GENERAL PAYMENT HSM INTEGRATION GUIDE

Applicable Devices:

Excrypt SSP Enterprise v.2

Excrypt Plus

TABLE OF CONTENTS

[1] DOCUMENT INFORMATION

[1.1] DOCUMENT OVERVIEW

In this document, we will describe the various payment applications and their use cases and how Futurex Payment HSMs can be utilized in multiple capacities throughout the payment process.

[1.2] PAYMENT APPLICATION USE CASES

In the payment space, there are two primary high-level types of applications: **Issuing** and **Acquiring**. Issuing focuses on issuing payment cards and provisioning mobile payment tokens. Acquiring focuses on the steps carried out between merchants and banks for processing credit and debit transactions, either through traditional card-based transactions or mobile payments.

Note: Due to regulatory requirements, payment acquiring and payment issuing processes are typically carried out inside separate HSMs.

[2] INITIAL SETUP

This section provides instructions for performing the minimum initial setup tasks required for all payment-related use cases on Futurex HSMs. These configurations can be made using either **Excrypt Manager** or **Futurex Client Tools (FXCLI)**.

- **FXCLI** (required)
  - Available for all operating systems
  - This tool can be utilized to perform all initial setup tasks, but it must be used to configure TLS mutual authentication between the HSM and the payment application you are integrating.
- **Excrypt Manager** (optional)
  - Available only for Windows
  - This tool provides a GUI option for performing most initial configurations on the HSM.

Section 2.1 covers FXCLI installation, and section 2.2 covers Excrypt Manager installation. Sections 2.4 and 2.5 cover the minimal setup tasks required, which are network configuration and loading major keys.

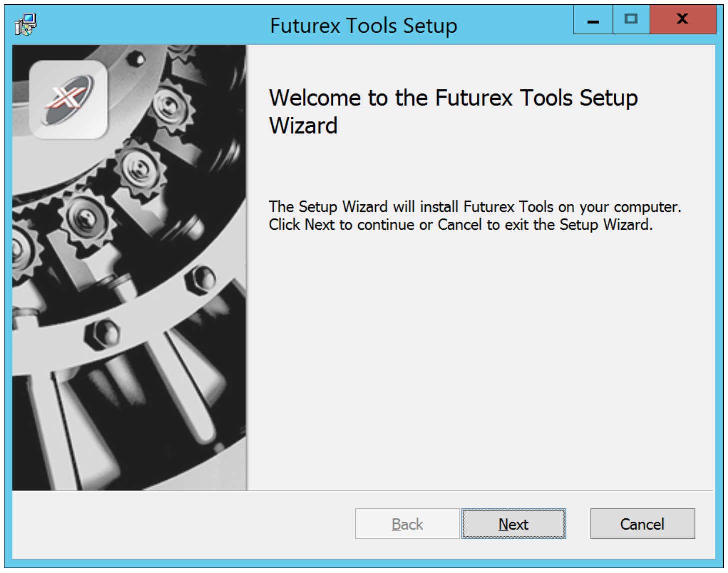
[2.1] INSTALL FUTUREX COMMAND LINE INTERFACE (FXCLI)

Note: Install FXCLI on the workstation you will use to configure the HSM.

[2.1.1] Installing FXCLI in Windows

On Windows, the easiest way to install **Futurex Command Line Interface (FXCLI)** on Windows is by installing the **FXTools** application, which includes FXCLI. You can download FXTools from the Futurex Portal.

To install FXCLI, run the Futurex Tools installer as an administrator and follow the prompts in the setup wizard to complete the installation.



Welcome to the Futurex Tools Setup Wizard

The Setup Wizard will install Futurex Tools on your computer. Click Next to continue or Cancel to exit the Setup Wizard.

FIGURE: FUTUREX TOOLS SETUP WIZARD

The setup wizard installs all tools on the system by default. You can override the defaults and choose not to install certain modules. The installation provides the following services:

- **Futurex Client Tools**:Command Line Interface (CLI) and associated SDK for both Java and C.
- **Futurex CNG Module**:The Microsoft Next Generation Cryptographic Library.
- **Futurex Cryptographic Service Provider (CSP)**:The legacy Microsoft cryptographic library.
- **Futurex EKM Module**:The Microsoft Enterprise Key Management library.
- **Futurex PKCS #11 Module**:The Futurex PKCS #11 library and associated tools.
- **Futurex Secure Access Client**:A client used to connect a Futurex Excrypt Touch to a local laptop through USB, which can then connect to a remote Futurex device.

[2.1.2] Installing FXCLI in Linux

Download FXCLI

You can download the appropriate FXCLI package files for your system from the Futurex Portal.

If the system is **64-bit**, select from the files marked **amd64**. If the system is **32-bit**, select from the files marked **i386**.

If running an OpenSSL version in the **1.0.x** branch, select from the files marked **ssl1.0**. If running an OpenSSL version in the **1.1.x** branch, select from the files marked **ssl1.1**.

Futurex offers the following features for FXCLI:

- Java Software Development Kit (**java**)
- HSM command line interface (**cli-hsm**)
- KMES command line interface (**cli-kmes**)
- Software Development Kit headers (**devel**)
- YAML parser used to parse bash output (**cli-fxparse**)

Install FXCLI

To install an rpm package, run the following command in a terminal:

```
$ sudo rpm -ivh [fxcl-xxxx.rpm]
```

To install a deb package, run the following command in a terminal:

```
$ sudo dpkg -i [fxcl-xxxx.deb]
```

Running FXCLI

To enter the HSM FXCLI prompt, run the following command in a terminal:

```
$ fxcli-hsm
```

After entering the FXCLI prompt, you can run **help** to list all of the available FXCLI commands.

[2.2] INSTALL EXCRYPT MANAGER (IF USING WINDOWS)

Note: Install Excrypt Manager on the workstation you will use to configure the HSM.

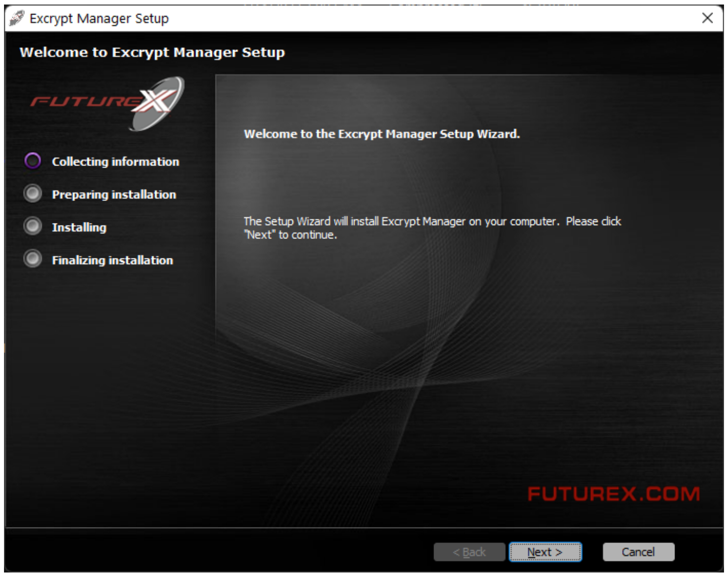
Sections 3 and 4 of this integration guide cover the installation of **Futurex Command Line Interface (FXCLI)** and **Excrypt Manager**. Excrypt Manager is a Windows application that provides a GUI-based method for configuring the HSM, while FXCLI provides a command-line-based method for configuring the HSM and can be installed on all platforms.

Note: If you will be configuring the HSM from a Linux computer, you can skip this section. If you will be configuring the HSM from a Windows computer, installing FXCLI in the next section is still required because FXCLI is the only method that can be used to configure TLS mutual authentication between the HSM and the payment application you are integrating.

Note: If you plan to use a Virtual HSM for the integration, all configurations will need to be performed using either FXCLI, the Excrypt Touch, or the Guardian Series 3.

Note: The Excrypt Manager version must be from the 4.4.x branch or later to be compatible with the HSM firmware, which must be 6.7.x.x or later.

To install Excrypt Manager, run the Excrypt Manager installer as an administrator and follow the prompts in the setup wizard to complete the installation.


Welcome to the Excrypt Manager Setup Wizard.

The Setup Wizard will install Excrypt Manager on your computer. Please click "Next" to continue.

FIGURE: EXCRYPT MANAGER SETUP WIZARD

The installation wizard prompts you to specify where you want to install Excrypt Manager. The default location is C:\Program Files\Futurex\Excrypt Manager\. After choosing a location, select **[Install]**.

[2.3] CONNECT AND LOG IN

For both **Excrypt Manager** and **FXCLI**, you must connect your laptop to the front USB port on the HSM. The initial login process described in this section uses the default Admin identities to log in under dual control.

User #1	User #2
User ID: Admin1	**User ID:** Admin2
Password: safe	**Password:** safe

[2.3.1] Logging in through Excrypt Manager

1. Open the Excrypt Manager application and select **[Refresh]** in the lower-right side of the **Connection** menu.
2. Select **USB Connection** and select **[Connect]**.
3. Login with both default Admin identities.
4. To be able to load major keys on the HSM in the next section, you must change the default passwords for both default Admin identities. To do so, navigate to the **Identity Management** menu.
5. Select the first default Admin identity and select **[Change Password]**.
6. Enter the default password (i.e., safe), enter the new password twice, and select **[OK]**.
7. Repeat steps 5 and 6 for the second default Admin identity.

[2.3.2] Logging in through FXCLI

1. Start the FXCLI application and run the following commands to connect and log in to the HSM:

   ```
   $ connect usb

   $ login user
   ```

   Note: The **login user** command prompts for the username and password. You need to run it twice because you must log in under dual control with both default Admin identities.
2. To be able to load major keys on the HSM in the next section, you must change the default passwords for both default Admin identities. To do so, run the following FXCLI commands:

   ```
   $ user change-password -u Admin1

   $ user change-password -u Admin2
   ```

   Note: The preceding **user change-password** commands prompt you to enter the old and new passwords for the identity you are updating.

[2.4] NETWORK CONFIGURATION

[2.4.1] Configuring network settings using Excrypt Manager

1. Navigate to the **Configuration** menu and modify the network settings as required.

[2.4.2] Configuring network settings using FXCLI

1. Run the **network interface modify** command to set a new IP for the HSM. An example is provided below to show the command syntax:

```
$ network interface modify --interface Ethernet1 --ip 10.221.0.10 --netmask 255.255.255.0 --
gateway 10.221.0.1
```

[2.5] LOAD MAJOR KEYS

The HSM requires you to load an **MFK (Master File Key)** before use. Depending on the intended use, you can also load a **PMK (Platform Master Key)**, **KEK (Key Encryption Key)**, and **FTK (Futurex Token Key)** at this point. The HSM allows you to load certain major keys through M of N fragmentation or a key wizard. With M of N key fragmentation, organizations can define a number of required key officers for a key ceremony that is less than the total number of key officers available. This allows organizations to maintain security while dramatically reducing the inconvenience of coordinating busy schedules around key ceremonies.

Section 2.5.1 shows how to load major keys through Excrypt Manager, and section 2.5.2 shows how to load major keys through FXCLI.

[2.5.1] Loading major keys through Excrypt Manager

1. Navigate to the **Key Management** menu and select **[Load]** next to the relevant key. You can load keys through M of N fragmentation or a key wizard. If this is the first HSM in a cluster, we recommend that you generate the key and save it to smart cards as M of N fragments.

[2.5.2] Loading major keys through FXCLI

1. If this is the first HSM you are setting up, you need to generate a random major key. Optionally, you can simultaneously load the generated key onto a smart card using the **-m** and **-n** flags.

```
$ majorkey random --mfk -m [number_from_2_to_9] -n [number_from_2_to_9]
```

2. If it's a second HSM that you're setting up in a cluster, then you will load the major key from smart cards using the following command:

```
$ majorkey recombine --key mfk
```

[3] ISSUING

Issuing focuses on issuing payment cards and provisioning mobile payment tokens.

[3.1] PIN AND OFFSET GENERATION

For both PIN generation methods described below, PINs are associated with an algorithm based on a particular key at the issuing bank. That key is a 3DES encryption key and is referred to as a **PIN Verification Key (PVK)**. A PIN is generated based on the customer's account/card number and the PIN Verification Key. This is referred to as the **natural PIN**.

In the past, issuers did not give customers the option to select their PIN. Instead, the bank would send the natural PIN to the customer in the mail, and they were forced to use the designated PIN. Now, most banks allow customers to select their own PIN. This is done by taking the PIN Verification Key, customer account number, and the chosen PIN, then sending that to an HSM to compare the natural PIN against the customer-selected PIN and determine the difference. The difference is referred to as the **PIN Verification Value** for the **VISA PVV** method and the **offset** for the **IBM 3624** method.

[3.1.1] VISA PIN Verification Value (PVV)

The VISA PIN Verification Value algorithm performs a multiple encipherment of a value, called the transformed security parameter (TSP), and a extraction of a 4-digit PIN verification value (PVV) from the ciphertext.

Note: Excrypt command **GVWW** can be used to generate a random VISA Working Key for use in the VISA Network.

[3.1.2] IBM 3624 (Offsets)

The IBM 3624 algorithm generates a n-digit PIN based on an account-related data or person-related data, namely the validation data. The assigned PIN length parameter specifies the length of the generated PIN.

Note: Excrypt command **GOFF** can be used to generate a PIN Offset for use in the IBM 3624 Network.

[3.1.3] Commonly used PIN and Offset Generation commands

Excrypt

- GNOF: Generate New Offset
- GOFC: Generate Offset of Clear PIN
- GOFF: Generate PIN offset value
- GPIN: Generate PIN (Diebold Method)
- GPIN: Generate PIN (IBM 3624 Method)
- GPIN: Generate PIN (Visa Method)

Standard

- 34: Generate Clear PIN and Offset
- 386: Generate MAC (DUKPT)
- 38C: Derive DUKPT Initial PIN Encryption Key
- 3D: Generate IBM 3624 Offset
- 3FA: Generate PIN and PVV

International

- BK: Generate IBM 3624 PIN Offset
- DE: Generate IBM PIN Offset
- DG: Generate Visa PIN Verification Value (PVV)
- EE: Derive PIN using the IBM Method
- FW: Generate Visa PIN Verification Value (of a customer selected PIN)
- JA: Generate Random PIN

[3.2] EMV KEY GENERATION AND DERIVATION

EMV stands for **Europay, Mastercard, and Visa**, the three companies that created the standard, and it is a payment method based on a technical standard for smart payment cards, payment terminals, and ATMs that can accept them.

EMV cards are smart cards, also called chip cards, which store their data on integrated circuit chips (ICCs), in addition to magnetic stripes for backward compatibility. These include cards that must be physically inserted (i.e., "dipped") into a reader and contactless cards that can be read over a short distance using near-field communication technology.

Payment cards that comply with the EMV standard are often called **chip and PIN** or **chip and signature** cards, depending on the authentication methods employed by the card issuer, such as a **personal identification number (PIN)** or **digital signature**. Outside the United States, the chip and PIN process is more common. It requires a secret four-digit PIN code known only by the cardholder to validate the EMV payment, making it significantly more secure. In the U.S., companies have opted for issuing chip and signature cards, weighing the risk of fraudulent transactions against the desire to make the purchasing process as seamless as possible for consumers.

[3.2.1] Commonly used EMV Key Generation and Derivation commands

Excrypt

- EMVG: Generate Master Key
- EMVK: Derive Key from Vendor Master Key and Derivation Data
- EMVM: Generate/Verify MAC
- GCIV: Generate a CVC IV
- GDAC: Generate a Data Authenticode Code (DAC)
- GDCV: Generate DCVC3
- GEMC: Generate EMV ICC Certificate
- GEMQ: Generate EMV Issuer CSR
- GIDN: Generate an ICC dynamic number (IDN)
- GOPC: Generate Offset and EMV PIN Change
- GVDC: Generate a Dynamic CVV
- OFPC: Perform EMV PIN Change Using Offset
- SSAD: Sign Static Authentication Data with Issuer Private Key

Standard

- 352: EMV Message Authentication Code (MAC) Generation
- 354: Generate Smart Card Master Key
- 368: Create Limited Use Key (LUK)

International

- KE: Generate an EMV Issuer CSR
- KI: Derive ICC key and encrypt under KEK
- KO: Generate an EMV ICC certificate and sign with issuer private key
- KU: Generate Secure Message with Integrity and optional Confidentiality

[3.3] MOBILE PAYMENT TOKEN ISSUANCE

Mobile payment tokens are governed by the pay brands (i.e., Google Pay, Apple Pay, Samsung Pay, etc.). For mobile payment tokens to be issued to a device, the card issuer (e.g., Wells Fargo, Chase, Bank of America, etc.) must have a relationship with the particular pay brand to which the mobile payment token will be issued. Each pay brand has specific data structures and encryption methods they require to communicate a token to a device, so the card issuer must support those methods for it to work.

[3.3.1] Commonly used Mobile Payment Token Issuance commands

Excrypt

- GHMC: Generate HCE Mobile Cryptogram
- GHMD: Generate HCE Magstripe Verification Value
- GHMK: Generate HCE Mobile Keys

Note: Mobile payment token issuance is not supported for the Standard and International command sets.

[3.4] CARD VERIFICATION VALUE (CVV) GENERATION

A **Card Verification Value (CVV)** is similar to a PIN, except it is not a secret value. A CVV is generated based on a **Card Verification Key (CVK)**. So the CVK is the base key, and the CVV value is based on that key and the customer account/card number.

Originally, CVV was meant to validate that a user has the card that's being used is not a cloned card.

Note: There is CVV generation and verification, but not translation because it is not encrypted between the hops.

[3.4.1] Commonly used Card Verification Value (CVV) Generation commands

Excrypt

- CAAV: Calculate Account holder Authentication Value
- GCAV: Generate CAVV
- GCSC: Generate American Express (Amex) CSC Value
- GCVC: Generate CVC and CVC2
- GCVV: Generate CVV/CVC Value
- GDDC: Generate Discover dynamic CVV
- GVDC: Generate dynamic CVV
- GIDN: Generate ICC dynamic number (IDN)

Standard

- 35B: Generate American Express (Amex) CSC Value
- 5D: Generate Card Verification Value (CVV)

International

- CW: Generate Visa Card Verification Value (CVV)
- RY: Generate Random CSCK
- RY (Mode 3): Generate Card Security Codes for CSCK

[4] ACQUIRING

Acquiring focuses on the steps carried out between merchants and banks for processing credit and debit transactions, either through traditional card-based transactions or mobile payments.

[4.1] PIN TRANSLATION AND VERIFICATION

When a user enters their card's personal identification number (PIN) on a device (i.e., an ATM or a payment terminal), the entered PIN is encrypted and then sent to an HSM in the bank's datacenter, along with the account/card number and the PVV or offset (depending on which method is used). The HSM then decrypts the PIN that the user entered, takes the account/card number, the bank's PIN Verification (PVK), and the PVV or offset and derives the actual PIN, then compares it to what the user entered and says, "yes" or "no" to the transaction.

The data structure for an encrypted PIN is called a **PIN block**. PINs are typically four characters, but you cannot encrypt only four characters with the DES algorithm. Instead, it must be an 8-byte block of data. This is one of the reasons why the concept of PIN blocks was created.

Embedded in a PIN block are the length of the PIN, the PIN itself, and padding. Then, all of that data is XORed with a portion of the account number. The result is the clear value of the PIN block. The clear PIN block value is then encrypted with a PIN Encryption Key (PEK).

Different standards exist for PIN blocks. ISO format 0 and ISO format 3 are currently supported. The new ISO format 4 standard supports an AES PIN encryption key. AES pin blocks were previously not supported due to issues with AES requiring a 16-byte block of data to encrypt.

[4.1.1] Commonly used PIN Translation and Verification commands

Excrypt

- RPIN: PIN Change & Optional PIN Verification (IBM 3624)
- RPIN: PIN Change & Optional PIN Verification (IBM 3624 DUKPT)
- RPIN: PIN Change & Optional PIN Verification (Visa)
- RPIN: PIN Change & Optional PIN Verification (Visa DUKPT)
- TPCP: Translate Encrypted PIN Coordinates to a PEK for Generate New Map Collection
- TPDD: Allow an encrypted ANSI PIN block to be translated
- TPIN: Translate PIN blocks
- TPIN: Translate PIN blocks (DUKPT)
- TPIN: Translate PIN blocks (IBM 4736
- TRPN: Translate PIN from RSA to Symmetric PIN Block
- TSPN: Translate PIN from PIN block to RSA encryption

- VMAP: Verify MAC and PIN (Diebold Method)
- VMAP: Verify MAC and PIN (IBM 3624 Method)
- VMAP: Verify MAC and PIN (Visa Method)
- VPIN: Verify PIN
- WPIN: Weak PIN checking
- XPIN: ANSI to ANSI
- XPIN: ANSI to PIN Pad
- XPIN: Extended PIN Translation
- XPIN: IBM 3624 to IBM 3624
- XPIN: IBM 3624 to PIN Pad
- XPIN: IBM 4736 to IBM 4736
- XPIN: PIN Pad to PIN Pad

Standard

- 31: Translate PIN Block
- 32: Verify PIN
- 32: Verify PIN - ANSI
- 32: Verify PIN - Compare two PIN Blocks
- 32: Verify PIN - Diebold
- 32: Verify PIN - Diebold DUKPT
- 32: Verify PIN - IBM 3624
- 32: Verify PIN - IBM 3624 DUKPT
- 32: Verify PIN - IBM 4736/NCR
- 32: Verify PIN - Visa
- 32: Verify PIN - Visa DUKPT
- 335: Translate PIN Block
- 33: Extended PIN Translation
- 346: DUKPT PIN Translate
- 35: Translate PIN Block using Double Encryption
- 36: Verify PIN Block using Double Encryption
- 36: Verify PIN Block using Double Encryption (ANSI)
- 36: Verify PIN Block using Double Encryption (Compare two PIN Blocks)

- 36: Verify PIN Block using Double Encryption (Diebold - DUKPT)
- 36: Verify PIN Block using Double Encryption (Diebold)
- 36: Verify PIN Block using Double Encryption (IBM 3624 - DUKPT)
- 36: Verify PIN Block using Double Encryption (IBM 3624)
- 36: Verify PIN Block using Double Encryption (IBM 4736 - NCR)
- 36: Verify PIN Block using Double Encryption (Visa - DUKPT)
- 36: Verify PIN Block using Double Encryption (Visa)

International

- BC: Verify a Terminal PIN using the Comparison method
- BE: Compare PIN to Encrypted PIN Block
- CA: Translate PIN Block from TPK to PEK Encryption
- CC: Translate PIN Block from one PEK to another PEK
- CI: Translate PIN Block from BDK to PEK Encryption
- CK: Verify PIN using the IBM Method (DUKPT)
- CM: Verify PIN using the Visa PVV Method
- CU: Validate PIN and Generate new PVV
- DA: Verify Terminal PIN Block - IBM 3624
- DC: Verify Terminal PIN Block - Visa
- DU: Verify and Generate an IBM PIN Offset (of a customer selected PIN)
- EA: Verify PIN - IBM 3624
- EC: Verify PIN - Visa
- G0: Translate PIN from BDK to ZPK Encryption (3DES DUKPT)
- JC: Translate PIN from TPK to MFK Encryption
- JE: Translate PIN from ZPK to MFK Encryption
- JG: Translate PIN from MFK to ZPK Encryption
- SC: Verify Public Key Encrypted PIN

[4.2] EMV VALIDATION

When an EMV card is "dipped" at a terminal, the terminal performs a few validations, such as the type of product (i.e., debit/credit) and the cardholder verification method (i.e., chip and PIN/chip and signature).

After validations, the terminal requests an **Authorization Request Cryptogram (ARQC)** from the card's **integrated circuit chip (ICC)**.

[4.2.1] ARQC validation and ARPC generation

The chip generates an 8-byte **Application Cryptogram (AC)** using an **Application Cryptogram Master Key (MKAC)**. The chip performs the following actions to generate the Application Cryptogram (AC):

1. An **Application Cryptogram Session Key (SK_{AC})** is generated using the MKAC and an **Application Transaction Counter (ATC)** value. The session key is unique for each transaction.
2. Using the session key and the data, it generates an Application Cryptogram (AC) by applying 3DES or AES.

In the case of an online transaction, this authorization cryptogram is called an Authorization Request Cryptogram (ARQC). The chip sends the ARQC to the terminal, which in turn sends ARQC in the authorization message to the issuer host for authorization.

Upon receiving the ARQC in the authorization request, the issuer system validates the ARQC and generates an **Application Response Cryptogram (ARPC)** using a hardware security module (HSM).

If the ARQC verification is successful, an ARPC is generated using ARQC as one of the inputs. If the ARQC verification is unsuccessful, then the ARQC should not be used as input for ARPC generation.

There are two methods that can be used to generate the ARPC. Method 1 generates an 8-byte ARPC using an 8-byte ARQC and a 2-byte Authorization Response Code (ARC) as input. Method 2 generates a 4-byte ARPC using an 8-byte ARQC, a 4-byte Card Status Update (CSU), and a 0–8 byte Proprietary Authentication Data as input.

The ARPC is sent in an authorization response message from the issuer to the terminal via the acquirer.

[4.2.2] Commonly used EMV Validation commands

Excrypt

- EMPT: Translate PIN Block for EMV Personalization
- EMVA: Verify QRQC and optionally generate ARPC
- EMVP: PIN Change
- EMVR: EMV RSA Private Key or Component Translation to Encryption Under a Personalization Key
- EMVS: Translate an ICC Master Key to Encryption Under a Personalization Key
- EMVT: EMV Translate Sensitive Data
- VCAC: Verify EMV Mastercard CAP Token
- VCAV: Verify Cardholder Authentication Verification Value
- VDAC: Verify a Data Authentication Code (DAC)
- VDCV: Verify CVC3

- VDDC: Verify Discover Dynamic Card CVC3
- VEMI: Verify an EMV Issuer Certificate
- VIDN: Verifies an ICC Dynamic Number (IDN)
- VVDC: Verify a Dynamic CVV

Standard

- 350: EMV ARQC Validation
- 357: Dynamic CVV Validation
- 359: Dynamic CVC3 Validation
- 365: Verify Visa Cloud-Based Payments

International

- JS: UPI ARQC Verification and ARPC Generation
- KG: Validate a signed Issuer Certificate
- KQ: ARQC (or TC/AAC) Verification and/or ARPC Generation

[4.3] MOBILE PAYMENT ACCEPTANCE

Mobile payments work using a combination of specialty hardware and a mobile wallet (i.e., Google Pay, Apple Pay, Samsung Pay, etc.). Users hold their mobile devices up to the contactless payment terminal. This allows the two devices to communicate over a specific radio frequency, passing encrypted payment information back and forth to complete the transaction. The funds then leave the user's digital wallet and enter the business's point of sale system.

Tokenization is another critical piece in enabling mobile transactions and payments. When a user sets up a mobile wallet, details are sent to their issuing bank. The bank replaces these details with a token, a set of randomly generated numbers. This keeps your account details completely encrypted and protected from fraud.

[4.3.1] Commonly used Mobile Payment Acceptance commands

Excrypt

- DAPT: Decrypt Apple Pay Token
- DGPT: Decrypt Google Pay Token
- DSPT: Decrypt Samsung Pay Token
- VHMC: Verify HCE Mobile Cryptogram
- VHMD: Verify HCE Magstripe Verification Value

Note: Mobile payment acceptance is not supported for the Standard and International command sets.

[4.4] CARD VERIFICATION VALUE (CVV) VALIDATION

Card issuers and banks use **card verification values (CVV)** to determine when a forged card is used in a card-not-present transaction. Using the card's expiration date, PAN data, and PIN, a hardware security module can cryptographically validate CVV (in its many forms, including CVC, CID, CSC, CVC2, and CVV2) and determine if the card data presented is authentic for online or contact-free payments.

[4.4.1] What CVV, CVC, and CVC2?

Card Verification Codes (CVC) accelerate the process of verifying card data in card-present transactions. The CVC is found within the magnetic stripe, and these codes are processed anytime a card is inserted into a card-reading point of sale terminal.

CVV and **CVC2** are commonly referenced for card-not-present transactions. They are usually in the form of a three or four-digit code on the back of the card, and they are often used in place of CVC for card-not-present transactions. The two terms are often used interchangeably.

[4.4.2] Reasons for validating CVV

In online transactions, CVV validation is one of two ways of verifying if cards are authentic (the other method is XML-based authentication). Storage of CVVs in any database is expressly prohibited, but businesses need to have access to the card's public key to decrypt and verify transactions. To secure the CVV, hardware security modules (HSMs) are used to validate and encrypt it.

Note: There is CVV generation and verification, but not translation because it is not encrypted between the hops.

[4.4.3] Commonly used Card Verification Value (CVV) Validation commands

Excrypt

- VAAV: Verify Account Holder Authentication Value
- VCSC: Verify American Express (Amex) CSC Value
- VCVC: Verify CVC and CVC2
- VCVV: Verify CVV
- VDCV: Verify Discover Dynamic Card CVV
- VDDC: Verify dynamic CVC value (CVC3)
- VEMI: Verify EMV Issuer Certificate

- VIDN: Verify ICC dynamic number (IDN)
- VVDC: Verify dynamic CVV

Standard

- 357: Dynamic CVV Validation
- 359: Dynamic CVC3 Validation
- 35A: Verify American Express (Amex) CSC Value
- 5E: Verify Card Verification Value (CVV)

International

- CY: Verify Visa Card Verification Value (CVV)
- RY (Mode 4): Verify Card Security Codes for CSCK

[4.5] MESSAGE AUTHENTICATION CODE (MAC) GENERATION AND VERIFICATION

Message Authentication Code (MAC), also referred to as a tag, is used to authenticate messages. In other words, MAC ensures that the message came from the stated sender (its authenticity) and has not been changed. The MAC value protects a message's data integrity by allowing verifiers (who also possess the secret key) to detect any changes to the message content, thus preventing man-in-the-middle attacks and fraudulent hosts or ATMs.

Currently, MACing is not very common in the U.S. for payment transactions. However, in Canada, a government-owned interbank network called Interac forces participating banks to MAC all payment transactions.

[4.5.1] Commonly used MAC Generation and Verification commands

Excrypt

- EMVM: Generate/Verify MAC
- GHPB: Generate HMAC and PBKDF2 Obfuscated Value
- GMAC: Generate Message Authentication Code (MAC)
- GPMC: General Purpose Symmetric MAC
- HMAC: Generate MAC Hash
- RKHM: Generate/Verify HMAC
- VMAC: Verify Message Authentication Code (MAC)
- VMAP: Verify MAC and PIN Diebold Method

- VMAP: Verify MAC and PIN IBM 3624 Method
- VMAP: Verify MAC and PIN Visa Method

Standard

- 56: Generate MAC (512 bytes or less)
- 57: Generate MAC (512 bytes or more)
- 5A: Verify MAC (512 bytes or less)
- 5B: Verify MAC (512 bytes or more)
- 5C: Verify and Generate MAC (DUKPT)
- 98: Generate MAC value based on MAC key and data
- 99: Verify MAC value based on MAC key and data
- 11D: Generate MAC and DEK keyblocks
- 304: Verify CMAC using TDES
- 305: Generate CMAC using TDES
- 324: Verify APACS 40 request MAC
- 348: Verify MAC using DUKPT key derived using the BDK and KSN
- 386: Generate MAC using DUKPT key derived using the BDK and KSN

International

- C2: Generate a MAC (AS2805)
- C4: Verify a MAC (AS2805)
- EO: Generate a MAC on a Public Key
- EQ: Verify a MAC on a Public Key
- GW: Generate or Verify a MAC (3DES DUKPT)
- L0: Generate an HMAC Secret Key
- LQ: Generate an HMAC on a block of data
- LS: Verify an HMAC on a block of data
- M6: Generate MAC using MAK (supports continuation mode)
- M8: Verify MAC using MAK (supports continuation mode)
- MA: Generate MAC using MAK
- MC: Verify MAC using MAK
- MK: Generate a Binary MAC

- MM: Verify a Binary MAC
- MQ: Generate a Binary MAC (extended length)
- MS: Generate a Binary or Hex MAC (extended length)
- MY: Verify and Generate MAC
- PQ: Generate an AS2805.4 MAC
- PS: Verify an AS2805.4 MAC
- RK: Verify MAC and return MAC Residue

[4.6] REMOTE KEY LOADING (RKL)

We all depend on encryption keys in one way or another. While few people outside the payments industry are aware of this, anytime you present your payment card at a Point of Sale (POS) terminal or use an ATM, an encryption key quickly goes to work to encrypt the PIN or the primary account number (PAN) associated with your card. This encryption obscures the data and protects against information theft as the transaction is sent back to the card issuer for validation. For this process to work, an encryption key must be securely loaded into that endpoint device, whether it be an ATM, a POS terminal, or even a commercial off-the-shelf device used for payment acceptance.

How does that encryption key find its way onto those devices? This has traditionally been done manually through a process known as direct key injection. For POS terminals and PIN entry devices, this involves bringing the devices to a key injection facility where key administrators manually inject each device. This can be time consuming and expensive. It requires the upfront cost of maintaining a validated Payment Card Industry (PCI) Level 3 key injection facility (KIF), and the operational costs of shipping devices to the KIF anytime they need to be rekeyed. For larger devices, like ATMs and gas station payment terminals, key administrators will often have to travel to each device in the field to load the necessary encryption keys. For organizations with widespread ATM or POS networks, this can be a significant operational expense with a high susceptibility to human error.

While the direct injection model has been sufficient for many organizations, others will find a remote key loading (RKL) solution more cost effective and efficient. With RKL, a remote key server establishes a secure, PKI-authenticated connection with each device and remotely distributes encryption keys without having to physically access the device. The ability to remotely rekey the device in the field without extended downtime is a powerful time and money saver for many organizations.

RKL allows organizations to manage keys for an entire infrastructure by sending cryptographically secure key exchanges from a centralized location. Better yet, devices can be rekeyed instantaneously with an absolute minimum of down time. Gone are the costs associated with maintaining an injection facility and manual injection.

Successful remote key loading (RKL) operations require collaboration and standardized communication protocols between the device manufacturer and the RKL provider. The backbone of RKL is trust at both ends of the key exchange, one end being the RKL provider and the other being the field-level device. This trust is established by a certificate authority, which provides both the endpoint terminal and the RKL platform with a digital certificate. This certificate serves as a private key in the public key infrastructure (PKI) used to facilitate secure key exchanges. The manufacturer’s role is to ensure that their devices have this certificate before deployment.

Furthermore, the endpoint devices and the RKL provider must use the same communication and encryption protocols, which furthers the manufacturer’s role in the process. While the most common and accepted encryption standard for RKL is TR-34, which was developed by ASC X9, there are many others in use depending on manufacturers, geographic location, and other factors. It is important for RKL providers to be accommodating in their platform design to allow integration with multiple manufacturers.

[4.6.1] Automated Teller Machines (ATMs)

ATMs are used by millions of people withdrawing cash every year. In 2016, the United States Federal Reserve noted that of the 91% of Americans with a credit, debit or other bank account, 75% use ATMs for cash withdrawals (1). With so many people depending on ATMs functioning properly, security is a major concern. ATMs rely on network protection and PIN encryption techniques to keep the customer customer’s PINs safe.

The encryption keys used to encrypt and validate PINs must be rotated on a regular basis to meet compliance mandates and maintain security. Before remote key loading became a viable option, key holders were required to visit each ATM in person to rotate keys across the network. This process was cumbersome and has grown increasingly infeasible as the number of ATMs continues to grow.

Furthermore, Payment Card Industry Data Security Standard (PCI DSS) regulations require that all PINs be encrypted upon capture at the terminal. RKL provides a secure, efficient, and cost-effective method for loading and managing ATM encryption keys across entire ATM networks.

[4.6.2] Point of Sale (POS) Terminals

POS terminals have double the encryption work. Like ATMs, they encrypt PINs for debit card transactions, but many merchants also require primary account numbers, commonly known as PANs, which are the account number associated with credit card payments, to also be encrypted. While PCI DSS regulations do to not currently require PAN encryption, its rapidly becoming the norm in the payments landscape. Recent years have seen high-profile data breaches that were traced back to a lack of PAN encryption. PAN encryption works similar to PIN encryption, but the technology surrounding PAN encryption is typically referred to as Point-to-Point Encryption (P2PE). Like PIN encryption, P2PE encrypts the PAN at the moment of capture in the POS device.

The encryption mechanisms behind PIN and PAN encryption differ slightly on the payment processing end, but they are the same for the purposes of RKL. Both processes require reliable access to encryption keys. Most POS terminals will have at least 2 key slots, with separate keys for both PIN and PAN encryption.

[4.6.3] Commonly used Remote Key Loading (RKL) commands

Excrypt

- PEDK: Key Request (Version 3 - Mode 1: Two Pass)
- PEDK: Key Request (Version 3 - Mode 2: One Pass TR-34)
- PEDK: Key Request (Version 3 - Mode 2: One Pass TR-34 with AES)
- PEDK: Key Request (Version 3 - Mode 3: One Pass)
- DRKI: Identification Request (Mode 1)
- DRKI: Identification Request (Mode 2)
- DRKK: Key Request (Mode 1)
- DRKK: Key Request (Mode 2)
- DRKV: Key Verification Request (Mode 1)
- DRKV: Key Verification Request (Mode 2)

Note: Remote Key Loading (RKL) is not supported for the Standard and International command sets.

[4.7] ATM NETWORK

An **ATM network**, also known as an **interbank network**, is a computer network that enables ATM cards issued by a financial institution that is a member of the network to be used to perform ATM transactions through ATMs that belong to another member of the network.

[4.7.1] Network Key Exchange

Every active ATM has a **PIN Encryption Key (PEK)**, **Master Key**, and **Key Encryption Key (KEK)** loaded on it through **Remote Key Loading (RKL)**. The way this is accomplished is, on the backend, the payment application sends a request to an HSM to generate a random KEK, encrypt it under the ATMs public key/certificate, and send it to the ATM. The ATM then decrypts the KEK using the private key associated with the public key/certificate that encrypted it. This key management scheme is called **Master/Session**.

On the ATM, the KEK (the "Master" key) is then used to encrypt a randomly generated Working Key (called the "Session" key). In this case, the session key is the PIN Encryption Key (PEK).

The KEK (the "Master key) is not changed out infrequently. In some cases, it is once a year, while in others, it is the life of the ATM. The PEK (the "Session" key) is changed out regularly. In some cases, once a day or once every 1,000 transactions.

[4.7.2] Foreign Transaction Example

Note: For this example, we will assume that an IBC Bank card is being used at a Chase Bank ATM to withdraw funds.

An IBC Bank customer goes to a Chase Bank ATM and inserts their IBC debit card in the card reader. They enter their PIN, which the ATM encrypts under its PIN Encryption Key (PEK) and sends to First Data Corporation. First Data hosts their STAR Network, which provides debit acceptance at more than 2 million retail POS, ATM, and Online outlets for nearly a third of all U.S. debit cards (**Note:** First Data is used in this example, but many other networks exist to provide debit acceptance services.)

Because First Data also has a relationship with IBC, they know that IBCs processor is Visa Debit Processing Services (DPS), and therefore forwards the request to Visa DPS to process the request and send the response back to First Data, which delivers the response to the Chase ATM.

Note: You may be wondering why Chase can't just send the request directly to IBC or Visa DPS. This is because it's not feasible for Chase to have relationships with every processor. So they go to a network, which is First Data in this case. First Data has relationships with all banks and processors or with other networks that do have a relationship with the issuing bank.

[4.7.3] PIN Translation

PIN translation plays a crucial role in all ATM foreign transactions. Between each of the hops in the example above, a PIN translation takes place using the TPIN Excrypt command. Each involved entity has its own PIN Encryption Key (PEK). In addition, between each link in the network chain, a shared PEK is established between the two entities. For example, between Chase and First Data, they have a shared PEK, which we'll call "CFD-PEK". This shared PEK is stored on an HSM in both companies' datacenters. So, after the IBC cardholder enters their PIN, the Chase ATM encrypts it under its PEK and sends it to the Chase HSM, requesting that it perform a PIN translation from the ATM PEK to the "CFD-PEK" and then send it to First Data. First Data then performs a PIN translation from the PIN being encrypted under the "CFD-PEK" to being encrypted under its own PEK. Then, the encrypted PIN is translated again to being encrypted under the PEK established between Visa DPS and them, and the process continues. To reiterate, a shared key must be established between each link in the chain for PIN translation to occur.

Note: Typically, it's not just a PEK that is involved. Most likely, there would also be an "MFK-CFD-KEK" key established, and the KEK would be used to share other keys so that the PEK could be swapped out periodically.

[4.8] POINT-TO-POINT ENCRYPTION (P2PE)

Point-to-point encryption (P2PE) is an encryption standard established by the PCI Security Standards Council in 2012. It stipulates that cardholder information is encrypted immediately after the card is used with the merchant's point-of-sale terminal and is not decrypted until the payment processor has processed it.

P2PE encryption is not required by **PCI-DSS**, which is for a few reasons. First, PCI states that if you have a network where clear cardholder data is transmitted, you are within the scope of DSS compliance requirements. Because of this, when companies started encrypting cardholder data, they correctly assumed that they no longer fell within the scope of DSS. However, PCI caught wind of this and considered that if these companies were encrypting cardholder data, then they must need to securely manage encryption keys. This was when they came out with the **PCI-P2PE** security standard, making its requirements even more stringent than PCI-DSS.

In reality, there are primarily two types of companies that deploy P2PE in their payment processing transactions: 1) Retailers that want to reduce risk and don't want to be in the news, or 2) Acquirers that are trying to add value to their service.

Note: Before P2PE, there was no need for HSMs for acquirers because the only transactions going through an HSM were PIN-based transactions. Now, if an acquirer uses P2PE, every transaction goes through an HSM.

[4.8.1] Commonly used Point-to-Point Encryption commands

Excrypt

- DCDK: Decrypt Cardholder Data Using DUKPT
- ECDK: Encrypt Cardholder Data Using DUKPT
- ONGQ: Translate PAN Encrypted Under an Asymmetric Key Pair to a Different Trusted Public Key
- TCDK: Translate Cardholder Data Using DUKPT
- TDKD: Translate Cardholder Data Using DUKPT and Symmetric Keys
- TKDR: Translate DUKPT Data to RSA with Specific Output Data

Standard

- 388: 3DES DUKPT Encrypt/Decrypt Data
- 52: Data Translate

International

- M0: Encrypt a block of data
- M2: Decrypt a block of data

[5] FUTUREX PAYMENT HSM'S

[5.1] THE EXCRYPT PLUS AND EXCRYPT SSP ENTERPRISE V.2

The **Excrypt Plus** and **Excrypt SSP Enterprise v.2** hardware security modules (HSMs) handle cryptographic processing and key management for various payments-related use cases. Futurex HSMs protect data in transit, in use, and at rest through various physical and logical security measures. In addition, the HSM is validated under **FIPS 140-2 Level 3** and **PCI HSM** standards and complies with all relevant regulatory requirements for payment transaction processing, including those set out by Accredited Standards Committee X9.

The **Secure Cryptographic Device (SCD)** contained within the Excrypt Plus and Excrypt SSP Enterprise v.2 HSMs handles all sensitive operations and supports common algorithms used in the payments industry such as **3DES**, **AES**, **RSA**, **ECC**, as well as a range of key derivation and wrapping methods, message authentication algorithms, and more.

Most payment applications communicate with the Excrypt Plus or Excrypt SSP Enterprise v.2 using one of Futurex's four payments APIs, which are referred to in this document as the **Excrypt**, **Standard**, **International**, and **REST** APIs. The **Excrypt Universal Interface**, comprising these three APIs, integrates directly with virtually all major payment application providers and allows for easy replacement of legacy HSMs without application code changes.

Excrypt Plus or Excrypt SSP Enterprise v.2 utilization typically falls under one of two payment use cases covered in this document: **Issuing** or **Acquiring**.

[5.2] API'S SUPPORTED

For the most part, the **Excrypt**, **Standard**, and **RESTful (JSON)** APIs are quite similar in their syntax. For all three, the HSM knows when the command starts and stops based on opening and closing delimiters, and then in between each field in the command are values of a specific format (e.g., binary, hex, decimal) and length or length range. International commands have no delimiters, and the HSM determines the value for each field based on the length of the field and the offset of the field. An alternative method for International commands is for the HSM to determine when a command has ended based on a timer. For example, 15 ms after data has stopped being received, the HSM will assume that the entire command has been sent.

If you need more information about the different command sets listed below, please refer to the User Guide for either the Excrypt SSP Enterprise v.2 or the Excrypt Plus.

[5.2.1] Excrypt

The Excrypt Universal Interface (UI) is an Application Programming Interface (API) that runs on the Excrypt family of Futurex products.

The UI interfaces directly with any host transaction processing application, whether it is vendor-supplied or developed in-house, and it enables support for common domestic and international message formats for PIN translation, PIN verification, and key management.

The Excrypt UI is a unique offering among industry Application Programming Interfaces (API) because it can interpret a range of different command syntaxes used in host applications. It's benefits are numerous, including:

- Extensible design
- Simple command syntax
- Wide-spread compatibility

[5.2.2] Standard

Support for the Standard command set on the Excrypt line of Futurex HSMs enables customers to get up and running quickly with minimal to no changes to their existing code.

[5.2.3] International

Support for the International command set on the Excrypt line of Futurex HSMs enables customers to get up and running quickly with minimal to no changes to their existing code.

[5.2.4] RESTful (JSON)

The Excrypt command set is based on tags/values (e.g., AO is the tag and ECHO is the value) and there are beginning and ending delimiters for the entire command (i.e., [and]), and there are delimiters for each field (i.e., a semicolon). JSON works the same way. It has beginning and end delimiters, tags, tokens, and field delimiters. Except JSON is typically communicated over an HTTP call.

Futurex added JSON support to our HSMs because many customers want to be able to interface with the HSM using RESTful. This is because RESTful libraries are plentiful, and it makes it easy for developers because they don't have to worry about things like socket management or TLS. So now we expose the Excrypt command set in a RESTful JSON data structure over an HTTP POST call.

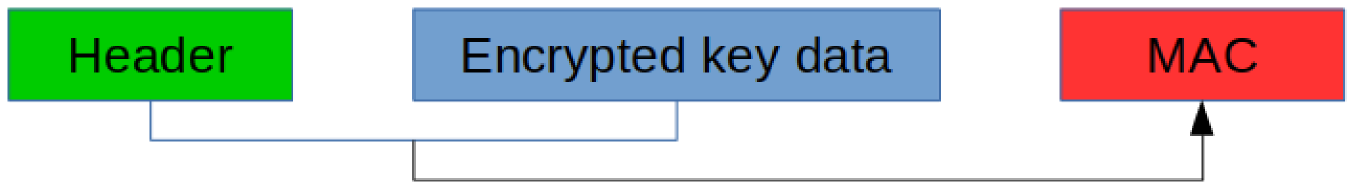
[5.3] KEY BLOCKS SUPPORTED

[5.3.1] Variant / Modifier-based Cryptograms

A cryptogram is simply an encrypted blob of data. Aside from the encryption itself, no additional security mechanisms are baked in. Instead, Futurex recommends using TR-31 key blocks to manage keys. The advantages of using TR-31 Key Blocks are explained further in the TR-31 section below.

[5.3.2] TR-31

TR-31 Key Blocks are described in the ANSI X9.24-1-2017 specification. The key block structure consists of three parts (Header, Encrypted key data, and MAC).



Header: is the least sensitive part of the key block. It defines the key block type, key usage, and key type.

Encrypted key data: Contains all the key sensitive data, including the actual key value and its size. It can optionally contain the ciphering mode used and data padding options.

MAC: (The Message Authentication Code) is an integrity check of the Header and Key data and ensures that the key block has not been modified.

TR-31 key blocks are used by Futurex HSMs for external key escrow and key transport. Futurex recommends using TR-31 key blocks to manage keys instead of cryptograms because key blocks safeguard against unauthorized substitution, replacement, or misuse of cryptographic keys by embedding information about a key within the key and data itself. Cryptograms, however, do not provide this extra level of security.

[5.3.3] Atalla Key Block (AKB)

A key that is secured by the Atalla Key Block consists of three parts:

1. An 8-byte header containing the attributes of the key (header);<
2. A 48-byte key field containing the Triple-DES cipher block chaining (CBC) mode ciphertext of the key (encrypted key field);
3. A 16-byte field containing the Triple-DES message authentication code (MAC) computed over the header and the encrypted key field.

[5.3.4] International Key Block

The International Key Block Structure defines four blocks instead of the three blocks defined by the TR-31:

- Header (16 bytes)
- Optional header
- Encrypted key data
- Authenticator

The main difference lies in adding an optional header block that allows more leeway in the management of the key.

[5.3.5] Key Table

Keys can also be stored with the HSM. These are generated on the internal HSM, wrapped with a major key, and stored on the HSM's internal storage table in one of the 25,000 available key slots. These keys can only be utilized with cryptographic operations on the internal HSM.

It is required to use this method of key storage if utilizing the PKCS #11 library, which does not support external key escrow. For this reason, this method is favored by those using the HSM in a General Purpose environment. Note that this method is not available through the International API, but can be managed through the Excrypt and Standard APIs, as well as through Excrypt Manager and Web Portal interfaces.

[5.4] KEY STORAGE METHODS

When considering different key storage methods, two primary variables come into play:

1. Whether keys are stored on the HSM or off the HSM
2. In what format are encrypted keys stored

These two variables are explained in further detail below.

[5.4.1] On vs Off the HSM

For most payments-related use cases, keys are stored off the HSM rather than inside the key table on the HSM. When keys are kept off the HSM, they are encrypted with a master key that is stored on the HSM.

[5.4.2] Encrypted Key Format

Encrypted keys are generally in one of the following two formats:

- Cryptogram
- TR-31 Key Block

Note: Key block formats other than TR-31 exist, but they are generally more proprietary. TR-31 Key Blocks were developed by the American Nation Standards Institute (ANSI) and therefore have more widespread support.

[5.5] KEY TRANSPORT METHODS

Various options are available for transporting keys from an external source to a Futurex HSM. The process to use depends greatly on the key source (i.e., where you are transferring the keys from), the key type (i.e., symmetric versus asymmetric), and the number of keys that need to be moved.

[5.5.1] Exporting keys from third-party/non-Futurex HSM's or Key Management Servers

Typically third-party HSMs and Key Management Servers support exporting keys, including private keys, under a wrapping key (e.g., KEK). In some cases, it is required to put the HSM or Key Management server in a special export mode. Please refer to the documentation specific to each third-party HSM or Key Management Server for details.

[5.5.2] Exporting keys from software sources

Exporting keys from software sources is often a more straightforward process than exporting from HSMs due to the ability for keys to be transferred in PKCS #12 format. PKCS #12 defines an archive file format for storing many cryptography objects as a single file. It is commonly used to bundle a private key with its X.509 certificate or to bundle all the members of a chain of trust.

A PKCS #12 file can also be generated using OpenSSL if you have the clear private key and its corresponding certificate using the command below:

```
openssl pkcs12 -export -out bundle.p12 -inkey myKey.pem -in cert.pem
```

[5.5.3] Encrypted Key Import

The following methods are available for importing encrypted keys into a Futurex HSM:

- **PKCS #12 import** using **Futurex Command Line Interface (FXCLI)**
- **PKCS #8 import** using the **RSTE Excrypt command**
- Using a **Key Exchange Key (KEK)**

Note: The PKCS #12 and PKCS #8 import options apply to asymmetric keys, and using a KEK for key import applies to symmetric keys.

[5.5.4] Clear Key Import

The methods listed below are available for importing clear keys into **Futurex HSMs**.

- **Full clear key import using Excrypt Manager:** If you have the full clear key value, it can be imported into the Futurex HSM by logging in under **dual control** through **Excrypt Manager** and then loading the key using either the **Symmetric** or **Asymmetric Key Loading Wizard**.
- **Component import using either Excrypt Manager or FXCLI:** Clear keys can also be loaded as **components**. In this scenario, more than one person would possess the clear key values for different parts of a key. Component holders would then need to log in to the Futurex HSM under dual control (using either Excrypt Manager or FXCLI), load each of the key components, and then the key parts would be XOR'd together and stored on the HSM.

  Note: This option is most common in the payment space.
- **Converting to KEK for batch import:** Another common use case is for a customer to need to import a large number of keys, making logging in under dual control and loading every individual key unfeasible. In this situation, all the keys could be encrypted under a single KEK and then batch imported into the Futurex HSM using the **TWKS** Excrypt command.

[5.6] TLS (SERVER-SIDE AUTHENTICATION), MUTUAL TLS (MTLS), CLEAR SOCKET

Mutually authenticating to the HSM using client certificates is recommended, but server-side authentication is also supported. To enable server-side authentication, enable the “Allow Anonymous” setting for the Excrypt

Port.

In a non-production setting, clear socket can be used for connections to the Excrypt HSM. However, this setting should not be used in production because all traffic is unencrypted and in the clear.

APPENDIX A: XCEPTIONAL SUPPORT

In today's high-paced environment, we know you are looking for timely and effective resolutions for your mission-critical needs. That is why our Xceptional Support Team does whatever it takes to ensure you have the best experience and support possible. Every time. Guaranteed.

- 24x7x365 mission critical support
- Level 1 to level 3 support
- Extremely knowledgeable subject matter experts

At Futurex, we strive to supply you with the latest data encryption innovations as well as our best-in-class support services. Our Xceptional Support Team goes above and beyond to meet your needs and provide you with exclusive services that you cannot find anywhere else in the industry.

- Technical Services
- Onsite Training
- Virtual Training
- Customized Consulting
- Customized Software Solutions
- Secure Key Generation, Printing, and Mailing
- Remote Key Injection
- Certificate Authority Services

Toll-Free: 1-800-251-5112

E-mail: support@futurex.com

ENGINEERING CAMPUS

864 Old Boerne Road

Bulverde, Texas, USA 78163

Phone: +1 830-980-9782

+1 830-438-8782

E-mail: info@futurex.com

XCEPTIONAL SUPPORT

24x7x365

Toll-Free: 1-800-251-5112

E-mail: support@futurex.com

SOLUTIONS ARCHITECT

E-mail: solutions@futurex.com