Linux on IBM Z and LinuxONE

Enterprise Key Management for Pervasive Encryption of Data Volumes

IBM

SC34-7740-01

Note

Before using this document, be sure to read the information in "Notices" on page 65.

This edition applies to the zkey utility as included in s390-tools version 2.17.0, and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

What's New in the 2022 edition

This update, SC34-7740-01, contains the following changes compared to SC34-7740-00.

New information

- An overview of KMS plug-in integration steps was added, see "Overview of KMS plug-in integration steps" on page 2.
- You can now use a plug-in to integrate zkey with KMIP servers, see Part 3, "Using the KMIP plug-in," on page 33.

Changed information

- The tools for managing secure keys have changed to accommodate the KMIP plug-in, see Appendix B, "zkey kms - Managing secure keys with a KMS plug-in," on page 49.

About this document

Learn how to integrate the keys used for pervasive encryption in the EKMF Web enterprise key management (KMS) system by using the zkey utility in a Linux instance on IBM Z® or LinuxONE.

This publication describes how to manage the HSM-protected keys used with protected-key dm-crypt in an enterprise setting. Using the zkey utility, you can generate and manage keys in the EKMF Web or KMIP enterprise-key management systems and import these keys into the zkey repository on your Linux instance. The publication describes the setup required on both the Linux instance and in EKMF Web or KMIP. It also describes how to use EKMF Web or KMIP to generate a key for a new volume, recovering a zkey repository, and how to efficiently process a master key replacement.

You can find the latest version of this document on the IBM® Documentation at:

```
https://www.ibm.com/support/knowledgecenter/en/linuxonibm/liaaf/lnz_r_ck.html
```

How this publication is organized

The first part of this publication contains general and overview information.

Part two discusses how to set up zkey to be integrated with the EKMF Web key management system.

Part three discusses how to set up zkey to be integrated with a key management system that has a KMIP interface.

Appendix A explains how to change the master key, and applies to both EKMF Web key management systems and key management systems that have a KMIP interface.

Appendix B contains reference information for the **zkey kms** command and its subcommands.

Who should read this publication

For the key-management system administrator, this publication explains how to set up **zkey** to integrate with your key-management system.

For the security administrator who is responsible for pervasive encryption with zkey and the key-management system, this publication gives insights into how keys are managed, and what you can do to optimize your workflow.

Further, this publication is interesting for Linux administrators who are responsible for storage configuration, as well as security engineers who design an enterprise key management process.

This publication assumes that you:

- Have Linux® on IBM Z or IBM LinuxONE administrator skills.
- Are familiar with security concepts.

Terms and abbreviations

This publication uses the terms and abbreviations listed here.

CA
Certificate authority. A trusted entity, external or internal to your organization, that issues digital certificates. Digital certificates are data files used to prove the identity of a website, person, or device by certifying the ownership of a public key by the named subject of the certificate.

CBC
Cipher block chaining. A method of reducing repetitive patterns in ciphertext by performing an exclusive-OR operation on each 8-byte block of data with the previously encrypted 8-byte block before it is encrypted.

CCA mode
: Common Cryptographic Architecture. Crypto Express adapters can work in different modes, whereof CCA is one.

CCA mode	Common Cryptographic Architecture. Crypto Express adapters can work in different modes, whereof CCA is one.
CSR	Certificate signing request. An electronic message that an organization sends to a CA to obtain a certificate. The request includes a public key and is signed with a private key; the CA returns the certificate after signing with its own private key.
dm-crypt	dm-crypt is the device mapper crypto target of the Linux kernel crypto target. It is a disk encryption subsystem, and is part of the device mapper infrastructure.
EKMF	IBM Enterprise Key Management Foundation. EKMF provides centralized key management for IBM cryptographic products on multiple platforms.
EKMF Web	IBM Enterprise Key Management Foundation Web is a web application that you use to manage keys on IBM Z and LinuxONE systems.
HSM	Hardware security module. A tamper-protected cryptographic device that protects master keys from being inspected. IBM Crypto Express CCA coprocessors and EP11 coprocessors are certified as HSMs. Each domain of an IBM Crypto Express coprocessor constitutes a virtual HSM and maintains a domain-specific master key or a set of master keys.
HSM master key	Each domain of a Crypto Express cryptographic coprocessor can contain active master keys which are used to generate secure keys.
KMIP	Key Management Interoperability Protocol (KMIP) is a client/server communication protocol for the storage and maintenance of key, certificate, and secret objects.
KMS	Key management system
LUKS2	Linux Unified Key Setup version 2 is used for disk encryption management.
protected key	A protected key is a key encrypted by a firmware master key.
secure key	A secure key is a key encrypted by an HSM master key.
TLS	Transport Layer Security. A set of encryption rules that uses verified certificates and encryption keys to secure communications over the Internet. TLS is an update to the SSL protocol.
zkey repository	An access-controlled list of secure keys managed by the zkey utility.

Other publications

These publications might be of interest.

Publications for Linux distributions

For Linux on IBM Z documents that are adapted to a particular distribution, see one of the following web pages:

- SUSE Linux Enterprise Server documents at

```
ibm.com/docs/en/linux-on-systems?topic=distributions-suse-linux-enterprise-server
```

- Red Hat® Enterprise Linux documents at

```
ibm.com/docs/en/linux-on-systems?topic=distributions-red-hat-enterprise-linux
```

- Ubuntu Server documents at

```
ibm.com/docs/en/linux-on-systems?topic=distributions-ubuntu-server
```

These publications are available on IBM Documentation at
ibm.com/docs/en/linux-on-systems?topic=linuxone-library-overview

- *Device Drivers, Features, and Commands*
- *Using the Dump Tools*
- *How to use FC-attached SCSI devices with Linux on z Systems®*, SC33-8413
- *Networking with RoCE Express*, SC34-7745
- *KVM Virtual Server Management*, SC34-2752
- *Configuring Crypto Express Adapters for KVM Guests*, SC34-7717
- *Introducing IBM Secure Execution for Linux*, SC34-7721
- *openCryptoki - An Open Source Implementation of PKCS #11*, SC34-7730
- *libica Programmer's Reference*, SC34-2602
- *libzpc - A Protected-Key Cryptographic Library*, SC34-7731
- *Exploiting Enterprise PKCS #11 using openCryptoki*, SC34-2713
- *Secure Key Solution with the Common Cryptographic Architecture Application Programmer's Guide*, SC33-8294
- *Pervasive Encryption for Data Volumes*, SC34-2782
- *Enterprise Key Management for Pervasive Encryption of Data Volumes*, SC34-7740
- *How to set an AES master key*, SC34-7712
- *Troubleshooting*, SC34-2612
- *Kernel Messages*, SC34-2599
- *How to Improve Performance with PAV*, SC33-8414
- *How to Set up a Terminal Server Environment on z/VM®*, SC34-2596

EKMF Web publications

The following EKMF Web publications are available:

- *IBM Enterprise Key Management Foundation - Web Edition: EKMF Web UI Configuration and Operation Guide*, SC28-2022
- *IBM Enterprise Key Management Foundation - Web Edition: EKMF Web UI User's Guide*, SC28-2023
- IBM Redbooks: *Key Management Deployment Guide using the IBM Enterprise Key Management Foundation*, SG24-8181

For details about EKMF Web, see the web page at:

```
https://www.ibm.com/products/enterprise-key-management-foundation-web
```

Part 1. Introduction and general concepts

Key management is an essential aspect of managing secure and resilient systems. The IBM Enterprise Key Management Foundation (EKMF) is a key-management system that provides real-time, centralized secure management of keys and certificates. Many key-management systems are based on the Key Management Interoperability Protocol (KMIP), which is also supported on Linux.

Data protection is often driven by industry regulations. However, compliance with regulations is only the minimum protection level. One of the most effective ways to protect data is to encrypt it. With the encryption of data, encryption keys become the most sensitive pieces of data. Therefore, special care must be taken in managing such keys since both their disclosure and their loss can have harmful effects on the enterprise. The control of such keys is assigned to special experts (security officers) and they must follow guidelines that are imposed by the enterprise, or other regulatory bodies.

Linux on IBM Z® and IBM LinuxONE offers pervasive encryption for data volumes for data at-rest through **`dm-crypt`** using secure keys that are managed with the help of the zkey utility.

Key management on the enterprise level

On Linux, a secure key repository controlled by the zkey utility can be used to manage secure keys to encrypt Linux volumes. To encrypt volumes, you use **`dm-crypt`** with the protected-key cipher paes. The encryption keys are protected by a Crypto Express adapter. The **`zkey`** command manages those keys locally on the system it runs.

To manage keys across your enterprise from a central location, not separately on each system, you can use the zkey utility with the key-management system plug-in interface. Key-management systems can provide a plug-in and thus can integrate themselves into the zkey utility. The zkey utility does not need to know any details about the key-management system, it just uses the plug-in to interface with the key-management system.

Using an enterprise-wide key management system includes these benefits:

- You can centrally manage the full lifecycle of cryptographic keys, including creation, access, maintenance, decryption, and destruction.
- It helps ensuring industry compliance. As compliance rules change, a centralized system that adheres to a standardized key management policy that is implemented across the organization can simplify updates.
- You can share keys across Linux instances, if these instances share encrypted volumes.

Figure 1 on page 2 illustrates the concept of a central KMS. The figure also illustrates how to set up a backup system: Linux instance 1 and 2 have a hardware disk-mirroring connection through PPRC between their disks. The keys for such mirrored instances can be easily fetched from the KMS.

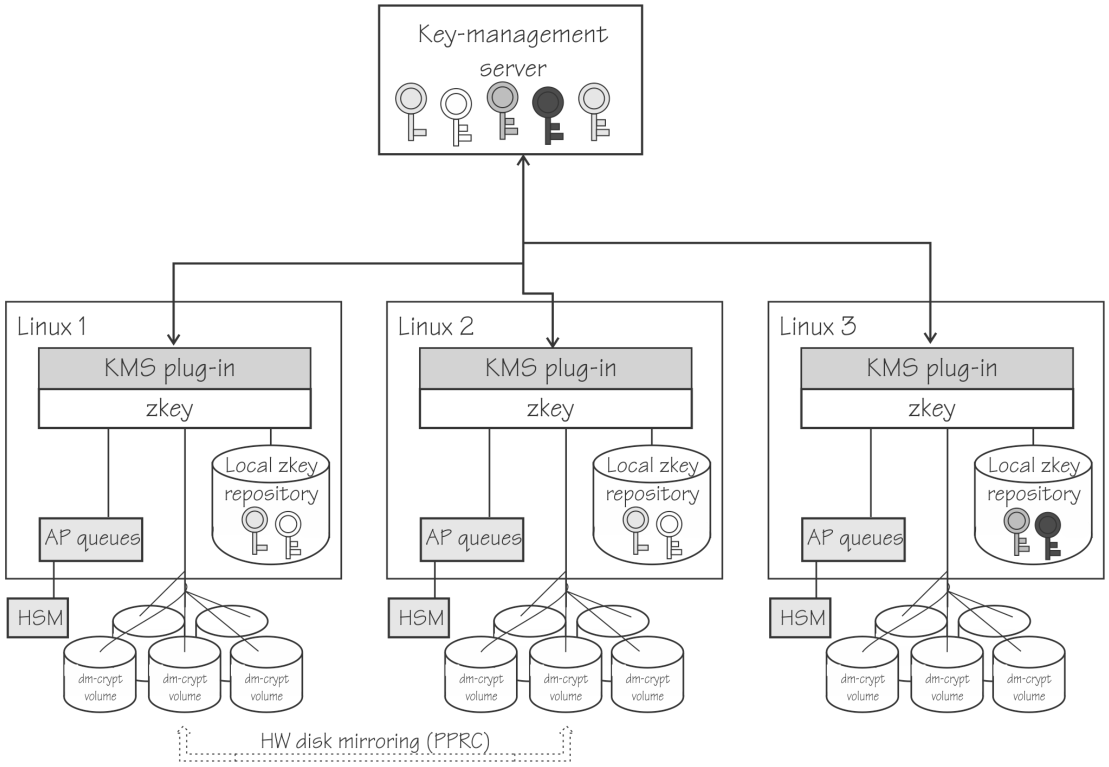


Figure 1. A central KMS serves multiple Linux systems. Keys can be populated in the local zkey repositories.

Overview of KMS plug-in integration steps

Key-management system plug-ins for EKMF Web and KMIP are available for Linux on IBM Z and LinuxONE.

The zkey utility can integrate with a key-management system. Thus, the key-management system can manage keys used by multiple Linux instances to encrypt volumes. The Linux instance must have access to a cryptographic adapter.

With EKMF Web or KMIP for Linux, security is emphasized. Keys are never exposed in plain text, and are only available to authorized parties, where both the key protection and the authentication of authorized parties is secured by HSMs.

The following steps provide an overview of the setup for a key-management system. For details, see Part 2, “Using the EKMF plug-in,” on page 5 or Part 3, “Using the KMIP plug-in,” on page 33.

Procedure:

1. Ensure the key-management system plug-in that you want to work with is available.
2. Bind the local key repository to the plug-in
3. Configure the plug-in.

   Configuring the plug-in usually entails:

   - Configuring AP queues.
   - Establishing communication with a server.
   - Defining supported key types

Figure 2 on page 3 shows a schematic overview of a KMS plug-in integrated with zkey and a KMS server. The secure connection between the plug-in and the server uses TLS and a transport key to protect the keys to be exchanged. Typically, with an enterprise key management system, keys are generated inside the KMS and can be downloaded to local zkey repositories of authorized clients.

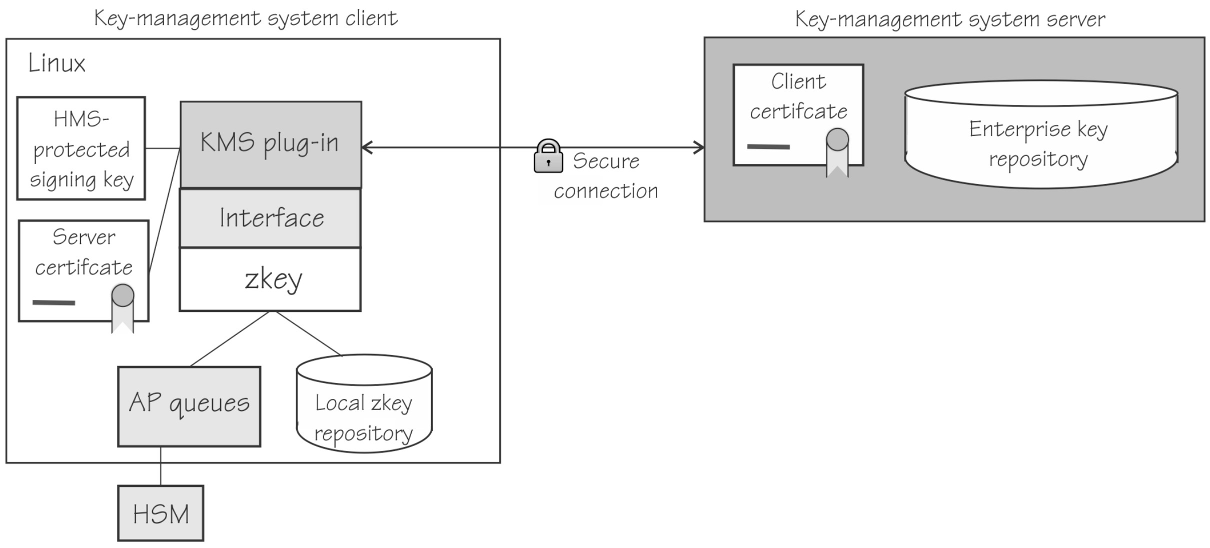


Figure 2. A KMS plug-in integrated with a KMS server

Once your set up is complete, you can use the zkey utility to perform all the tasks to manage your keys including:

- Generating keys in the KMS
- Modifying properties of keys in the KMS
- Listing keys of the client in the KMS
- Removing keys

The details of these operations are specific to the plug-ins and are described in detail in the plug-in specific part:

- Part 2, “Using the EKMF plug-in,” on page 5.
- Part 3, “Using the KMIP plug-in,” on page 33.

Part 2. Using the EKMF plug-in

The IBM Enterprise Key Management Foundation (EKMF) provides real-time, centralized secure management of keys and certificates. A key-management system plug-in for EKMF Web is available for Linux on IBM Z and LinuxONE.

Newest version

You can find the newest version of this book at: ibm.com/docs/en/linux-on-systems

Chapter 1. The EKMF Web plug-in

A key-management system plug-in for EKMF Web is available for Linux on IBM Z and LinuxONE.

This allows the zkey utility to integrate with EKMF Web. Thus, EKMF Web can manage keys used by multiple Linux instances to encrypt volumes. The Linux instance must have access to a cryptographic adapter.

With EKMF Web for Linux, security is emphasized. Keys are never exposed in plain text, and are only available to authorized parties, where both the key protection and the authentication of authorized parties is secured by HSMs.

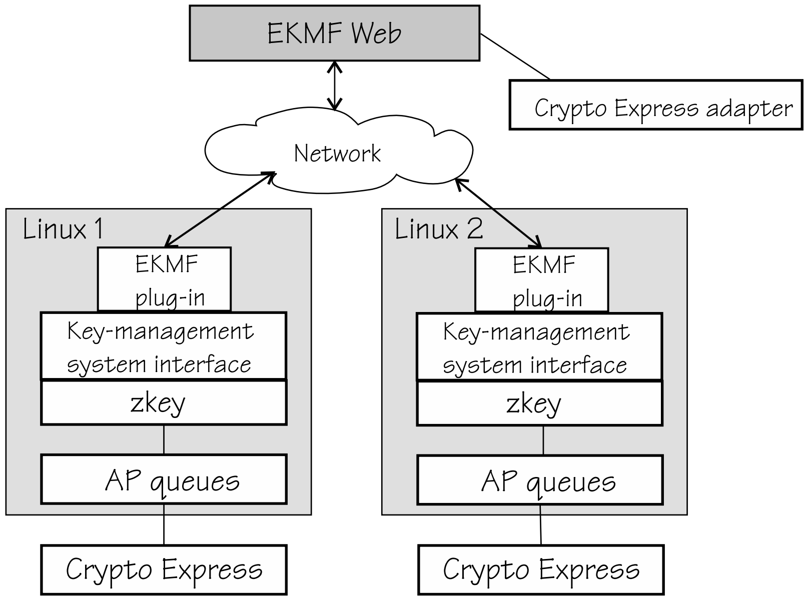


Figure 3. EKMF Web integrates through a plug-in with zkey on Linux

Figure 3 on page 7 illustrates how key management on multiple Linux instances, as z/VM guests, KVM virtual machines, or in LPAR mode, can be handled from a single key-management system.

Installing EKMF Web

Installing entails setting up a Web server for the EKMF Web interface, a database for the keys, cryptographic adapters, and user accounts.

For details about the installation, see the *EKMF Web UI Configuration and Operation Guide*, SC28-2022.

How EKMF Web works with zkey

On EKMF Web, templates for zkey keys must be prepared and then registered with zkey.

Cryptographic keys are the most important objects in a key management system. Apart from being the managed objects, keys are also used to establish a secure communication channel between the secure key repository and the EKMF Web server.

To use different key types for different purposes, these key types must be declared as *key templates* on the EKMF Web server.

On zkey, you must bind the zkey instance to EKMF Web.

These concepts are illustrated in Figure 4 on page 8.

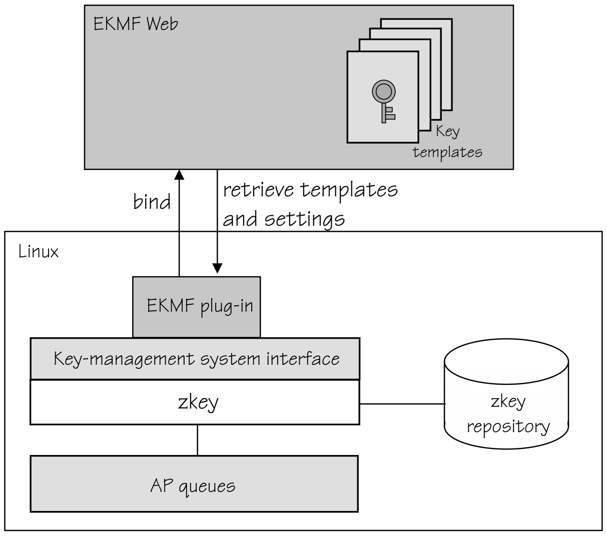


Figure 4. zkey and EKMF Web

EKMF Web software and hardware prerequisites

Deploying an EKMF Web and pervasive encryption solution on Linux on IBM Z and LinuxONE requires minimum levels of hardware and software on Linux.

For software and hardware prerequisites for EKMF Web, see the *EKMF Web UI Configuration and Operation Guide*, SC28-2022.

Hardware prerequisites

- IBM Z hardware as of IBM z13, or any LinuxONE system with the CPACF feature installed.
- A Crypto Express6S or later configured in CCA coprocessor mode.
- Volumes to be encrypted (for example, SCSI or DASD volumes). For DASD volumes, you can encrypt partitions only, not the complete DASD.
- The AES and APKA master keys must be set using the TKE (or the `panel.exe` program in a test environment).

Software prerequisites

- Linux kernel upstream version 5.4 or later for the support of secure keys of type CCA-AESCIPHER. Older versions where the required modules have been back-ported might also work.
- The **`cryptsetup`** utility version 2.0.3 or later is required to configure an encrypted volume.
- The **zkey** utility from the s390-tools package (as of upstream version 2.15.1) that contains the enhancements for EKMF Web. Both Red Hat Enterprise Linux 8.4 and SUSE Linux Enterprise 15.3 contain the correct version of **zkey**
- The CCA 6.0 package or later from the software-package selection page.

Access rights

The zkey user ID that is to be used for generating keys requires the following access rights:

- EKMF Web roles `keys:export`, `keys:generate`, `keys:pre_activation:activate`, `keys:write`, `keys:active:install`. See *EKMF Web UI Configuration and Operation Guide*, SC28-2022 for a full list of roles.
- In RACF, role-specific profiles must be defined. For each role, in the EJBROLE class and all user IDs must have READ access to the profiles corresponding to their required level of access. See *EKMF Web UI Configuration and Operation Guide*, SC28-2022 for examples.

Chapter 2. Setting up key templates

As an EKMF Web administrator, you need to set up key templates for use with zkey on EKMF Web for your organization, and register those templates with zkey.

About this task

A one-time setup is needed for key templates. The templates can then be used by any number of systems with zkey for key management.

You must define four key templates for zkey on EKMF Web:

- non-XTS - For volume encryption with an AES key in CBC mode.
- XTS part 1 - For volume encryption with AES in XTS mode, first part of an XTS key
- XTS part 2 - For volume encryption with AES in XTS mode, second part of an XTS key
- Identity key - Identifies the zkey system. This type of key is used for verifying the identity of zkey to EKMF Web.

By creating your own templates, you can decide on naming schemes and key properties such as the size of the key. For each template, define the algorithm, the size, the type, the state, and allow the key to be exported.

Creating pervasive encryption key templates

As an EKMF Web administrator, you must set up key templates for use with zkey on EKMF Web for your organization.

About this task

For pervasive encryption, you need to define three key templates:

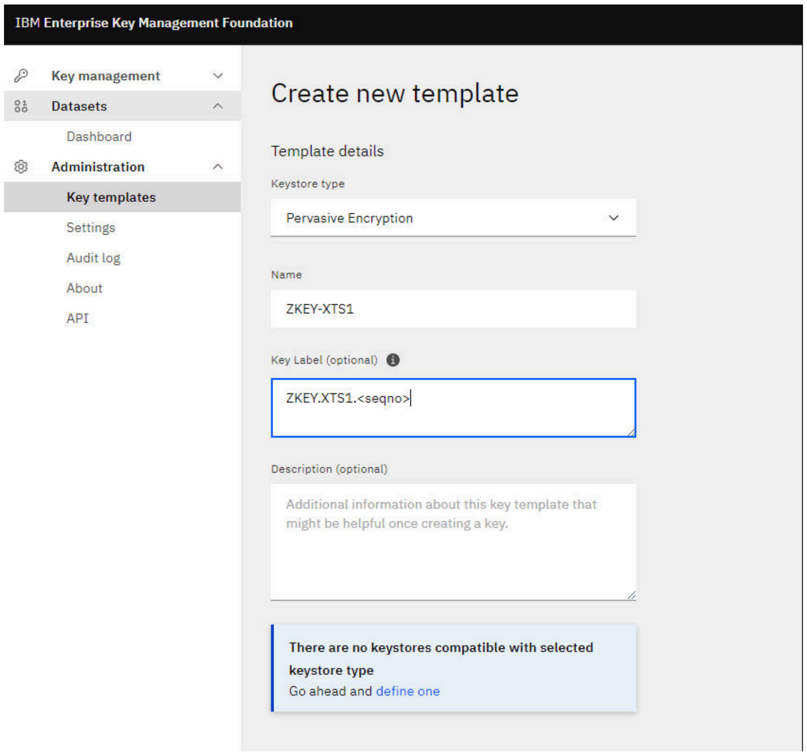
- non-XTS - For volume encryption with an AES key in CBC mode.
- XTS part 1 - For volume encryption with AES in XTS mode, first part of an XTS key.
- XTS part 2 - For volume encryption with AES in XTS mode, second part of an XTS key.

The two XTS templates must have the same properties.

Procedure

To create non-XTS, XTS1, and XTS2 key templates, follow these steps:

1. Log in to EKMF Web.
2. Go to **Administration** in the left navigation bar.
3. Click **Key templates**.
4. On the panel that opens, click the **Create** button on the right.
5. Select key type **Pervasive Encryption** from the drop-down menu.
6. In the **Name** field, enter the template name.
   The templates names can consist of up to 30 uppercase alphabetic characters, numerals, and hyphens. For example, assuming you want to remember that these key templates are for non-XTS zkey keys, the name can be ZKEY-NONXTS. An example of the first part of a template for the first part of an XTS key is show in Figure 5 on page 10



Create new template

Template details

Keystore type: Pervasive Encryption

Name: ZKEY-XTS1

Key Label (optional): ZKEY.XTS1.<seqno>

Description (optional): Additional information about this key template that might be helpful once creating a key.

There are no keystores compatible with selected keystore type
Go ahead and define one

Figure 5. Create new key template panel, part 1

7. In the **Key label** field, enter the pattern of the key names.

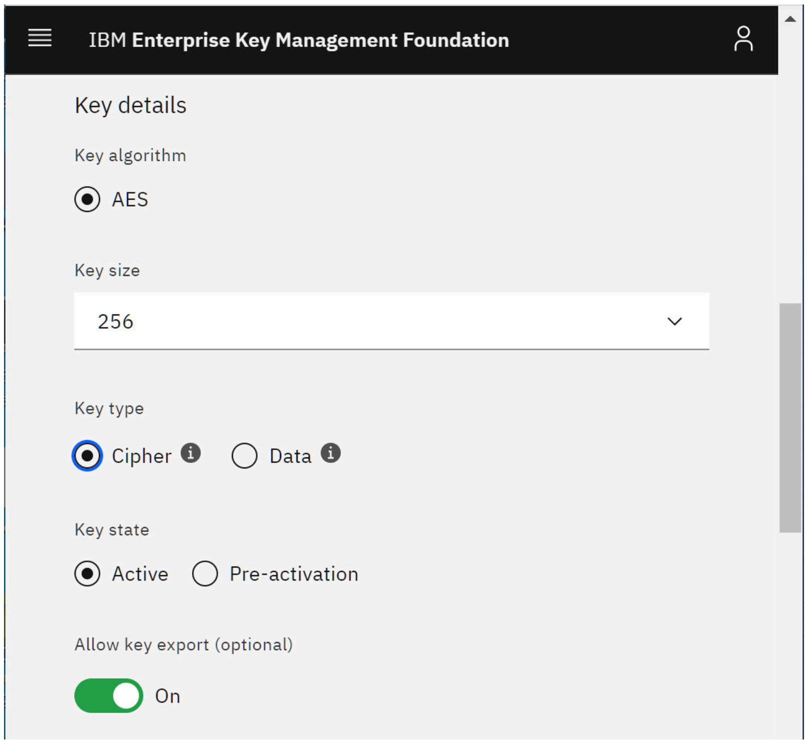
   All keys that are generated with this template have a name that follows this pattern.

   For example, assuming you want the keys to be named as the template and then have sequential numbering, enter:

   ```
   ZKEY.NONXTS.<seqno>
   ```

8. Select AES for the key algorithm. Only AES keys are eligible for pervasive encryption.
9. Select 256 for the key size.
10. Select Cipher for key type. For zkey, only cipher keys are possible.
11. For **Key state**, select Active.
12. Set **Allow key export** to on.

    This setting allows you to transfer the key to zkey. An example for the second part of a template for a key is shown in Figure 6 on page 11.


IBM Enterprise Key Management Foundation

Key details

Key algorithm

(●) AES

Key size

256

Key type

(●) Cipher ( ) Data

Key state

(●) Active ( ) Pre-activation

Allow key export (optional)

On

Figure 6. Create new key template, part 2

13. Optional: Set the key's active period.
14. Click **Save**.

Results

The template is created and shown on the **Key Template** page. The example template would be listed as ZKEY-NONXTS.

What to do next

Repeat the steps to create the non-XTS, XTS1, and XTS2 key templates. For information on how to create the identity key template, see "Creating an identity key template" on page 11.

Creating an identity key template

As an EKMF Web administrator, you need to set up an identity key template for use with zkey on EKMF Web for your organization.

Procedure

To create an identity key template, follow these steps:

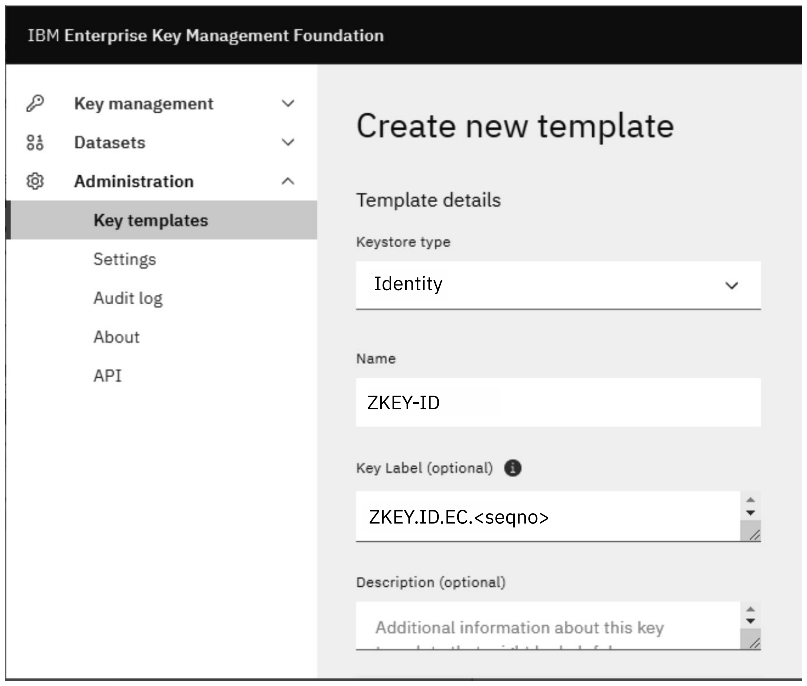
1. Log in to EKMF Web.
2. Go to **Administration** in the left navigation bar.
3. Click **Key templates**
4. On the window that opens, click the **Create** button on the right.
5. Select the key type **Identity** from the drop-down menu.
6. In the **Name** field, enter the template name.
   The templates names can consist of up to 30 uppercase alphabetic characters, numerals, and hyphens. For example, assuming you want to remember that these identity key templates are for zkey keys, the name can be ZKEY-ID.
7. In the **Key label** field, enter the pattern of the key names. In contrast to the name, the label can contain full stops, but no hyphens.

All keys that are generated with this template have a name that follows this pattern.

For example, assuming you want the keys to be named similar to the name, remember that it is an elliptic curve key, and have sequential numbering, enter:

```
ZKEY.ID.EC.<seqno>
```

An example for the first part of a template for an identity key is shown in Figure 7 on page 12.



Create new template

Template details

Keystore type: Identity

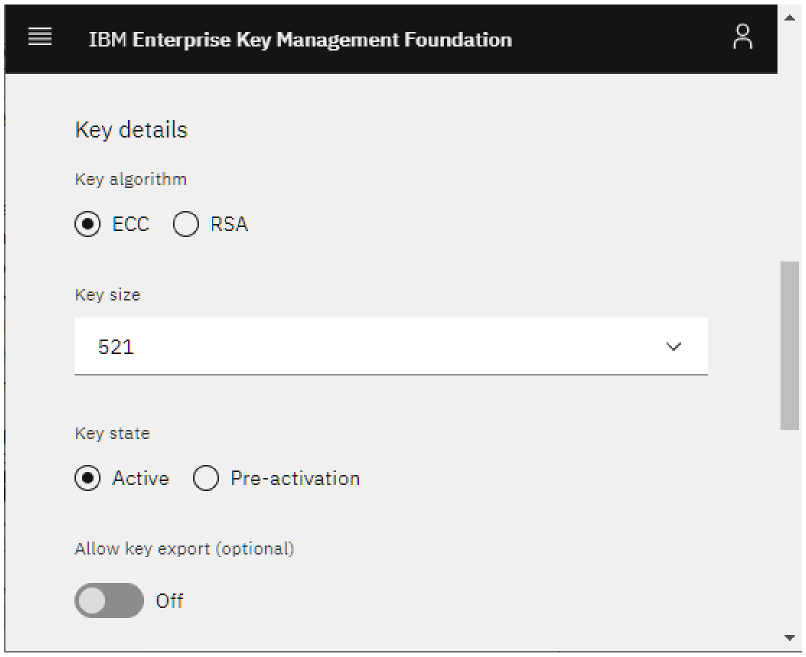
Name: ZKEY-ID

Key Label (optional): ZKEY.ID.EC.<seqno>

Description (optional): Additional information about this key

Figure 7. Create new identity key template, part 1

8. Select elliptic curve (ECC) or RAS for the key algorithm.
9. Select 521 for identity keys to use a prime 521 curve (ECC), or one of the supported RSA key sizes.
10. For **Key state**, select Active.
    An example for the second part of a template for an identity key is shown in Figure 8 on page 12.



Key details

Key algorithm: [✓ ECC] RSA

Key size: 521

Key state: [✓ Active] Pre-activation

Allow key export (optional): Off

Figure 8. Create new identity key template, part 2

11. Keep **Allow key export** off.
12. Optional: Set the key's active period.
13. Click **Save**.

Registering EKMF Web key templates with zkey

The key templates that you created must be made known to zkey.

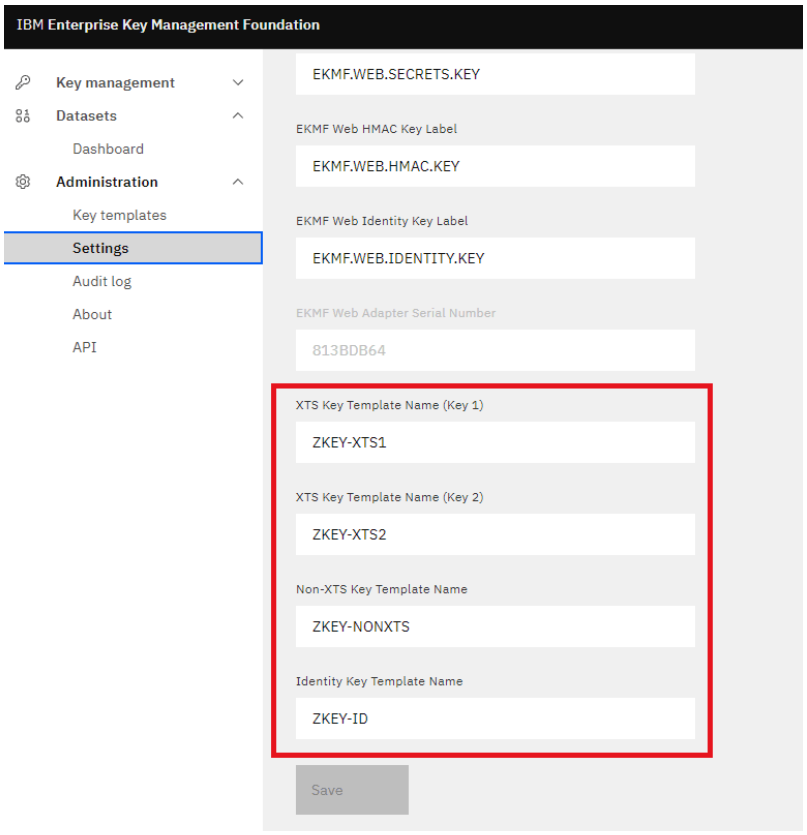
Procedure

1. On EKMF Web, go to **Administration** in the left navigation bar.
2. Click **Settings** in the left navigation
3. Enter the template names.
   Scroll down to find entry fields for:
   - XTS Key Template Name (Key1)
   - XTS Key Template Name (Key2)
   - Non-XTS Key Template Name
   - Identity Key Template Name

   Using the example from the key templates, you enter ZKEY-NONXTS in the **Non-XTS Key Template Name** field.
4. Click **Save**.

Example

In the screen capture that is shown in Figure 9 on page 13 all four templates are registered.



EKMF.WEB.SECRETS.KEY

EKMF Web HMAC Key Label
EKMF.WEB.HMAC.KEY

EKMF Web Identity Key Label
EKMF.WEB.IDENTITY.KEY

EKMF Web Adapter Serial Number
813BDB64

XTS Key Template Name (Key 1)
ZKEY-XTS1

XTS Key Template Name (Key 2)
ZKEY-XTS2

Non-XTS Key Template Name
ZKEY-NONXTS

Identity Key Template Name
ZKEY-ID

*Figure 9. Entering the template names on the **Settings** page*

Chapter 3. Setting up zkey

As a zkey user, you need to connect zkey to EKMF Web and configure the EKMF plug-in.

Connecting zkey with EKMF Web

After the EKMF Web administrator set up key templates and registered them for use with zkey, you can connect zkey to EKMF Web.

Before you begin

You need a Linux instance with the zkey utility installed. This instance of zkey must be able to connect to EKMF Web.

You need access to EKMF Web, that is, a user ID must be set up for you. For more details about access rights, see "Access rights" on page 8.

About this task

On your Linux system, use the **zkey** command to bind to the EKMF Web plug-in and configure it.

Procedure

1. Optional: Check which plug-ins are available.

   Issue the **zkey kms plugins** command, for example:

   ```
   # zkey kms plugins
   KMS-Plugin                Shared library
   ---------------------------------------------------
   EKMFWeb                   zkey-ekmfweb.so
   ```

   The example shows that the EKMF Web plug-in is available. It also shows the shared library that implements the plug-in.
2. To bind to the EKMF Web plug-in, issue the following command:

   ```
   # zkey kms bind EKMFWeb
   ```

   The local secure key repository is bound to a key management system of type EKMF Web.

   Note: A secure key repository can be bound to only one key management system.

Configuring an EKMF Web plug-in

After binding a key management system to the repository, the EKMF plug-in must be configured.

Before you begin

You require an EKMF Web user ID. For more details about access rights, see "Access rights" on page 8.

APKA and AES master keys must be in place on the AP queues that your Linux instance uses. Master keys are set through the TKE.

About this task

Use a command of this form to configure the plug-in:

```
# zkey kms configure <config_option>
```

The **zkey kms configure** command supports plug-in-specific options that the plug-in provides. To see which plug-in-specific options a plug-in provides or requires, use the command:

```
# zkey kms configure --help
```

You can supply all configuration options at once, or use the **zkey kms configure** command several times, supplying only one or a few configuration options each time. A useful command might be **zkey kms info**, which displays information about the key-management system to which zkey is bound. For example:

```
# zkey kms info
KMS-Plugin:              EKMFWeb
  Supported key types:  CCA-AESCIPHER
  APQNs:               (configuration required)
....
```

In the configuration phase, use the **info** command to see what must still be configured. The preceding example shows that APQNs, that is, AP queues, must be configured.

You must associate AP queues with the key management plug-in. AP queues perform secure key operations for the plug-in. Keys that are generated with the key management plug-in are automatically associated with these AP queues. If a plug-in supports different key types, for example CCA and EP11 keys, then at least one of the matching AP queues with this type must be associated. The EKMF Web plug-in supports only CCA type keys.

An overview of the setup process is shown in Figure 10 on page 16.

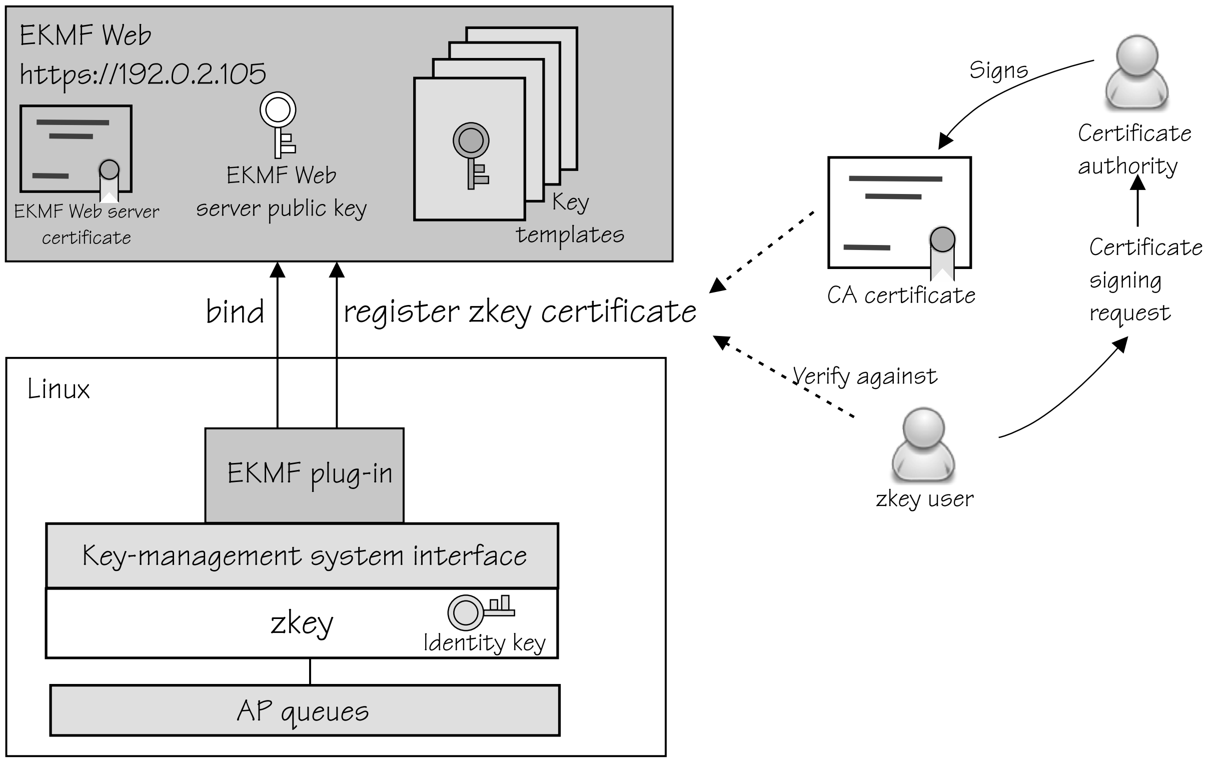


Figure 10. Exchanging configuration information between EKMF Web and zkey

Procedure

Specify the following settings.

1. To specify the AP queues to associate with the EKMF Web plug-in, use a command of the form:

```
# zkey kms configure --apqns <adapter1.domain1,adapter2.domain2,...>
```

The AP queue is defined by its adapter ID and domain ID. If you specify multiple AP queues, they must have the same master key.

For example, to add AP queues from adapter 08 and 09, both with domain 002f, issue:

```
# zkey kms configure --apqns 08.002f,09.002f
```

To see whether the AP queues are configured, use **`zkey kms info`**:

```
# zkey kms info
KMS-Plugin:              EKMFWeb
  Supported key types:   CCA-AESCIPHER
  APQNs:                 08.002f
                         09.002f
  EKMF Web server:       (configuration required)
```

In the next step, tell zkey which EKMF Web server to use.

2. Connect to the EKMF Web server.

   Use a command of the form:

```
# zkey kms configure --ekmfweb-url <URL>
```

The URL must start with `https://`, and can contain a port number that is separated by a colon. If no port number is specified, 443 is used for HTTPS. You can specify TLS-specific options to control the behavior of the TLS protocol and the validation of the EKMF Web server's certificate, see Appendix B, "zkey kms - Managing secure keys with a KMS plug-in," on page 49.

When the connection to the EKMF Web server is established, the EKMF Web settings, such as the EKMF Web server's public key and the key templates that are used by EKMF Web to generate keys are automatically retrieved from EKMF Web.

Tip: If the settings change in EKMF Web at a later time, use the `--refresh-settings` to refresh the settings in zkey.

```
# zkey kms configure --ekmfweb-url https://192.0.2.105:30140/
The EKMF Web server presented the following certificate to identify itself:
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 942492814 (0x382d4c8e)
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: C=us, O=ibm, OU=ekmf-web, CN=localhost
        Validity
            Not Before: Mar  17 11:37:54 2021 GMT
            Not After : Mar  17 11:37:54 2022 GMT
        Subject: C=us, O=ibm, OU=ekmf-web, CN=localhost
        Subject Public Key Info:
....
zkey: Is this the EKMF Web server you intend to work with [y/N]? y
```

Carefully check the certificate to ensure that you are connected to the correct server. For example, have the administrator of the web server send you the server certificate through a different method, and compare that certificate with what the **`zkey kms configure`** command gives you.

3. Log in to the EKMF Web server with the user ID given to you by the administrator.

```
zkey: EKMF Web user ID:
```

Go to the web browser as instructed and collect a one-time passcode for login. Using passcodes avoids sending passwords in the clear over the connection.

Now the zkey instance knows which EKMF Web server to work with. Using **`zkey kms info`** again, the example configuration now recognizes the key templates for this zkey instance:

```
# zkey kms info
KMS-Plugin:             EKMFWeb
  Supported key types:  CCA-AESCIPHER
  APQNs:                08.002f
                        09.002f
  EKMF Web server:      https://192.0.2.105:30140
  CA-bundle:            System's CA certificates
  Client certificate:   (none)
  Client private key:   (none)
  EBMF Web public key:  ECC (secp521r1)
  Key templates:
    Identity:           ZKEY-ID
      Label template:   ZKEY.ID.EC.<seqno>
    XTS-Key1:           ZKEY-XTS1
      Label template:   ZKEY.XTS1.<seqno>
    XTS-Key2:           ZKEY-XTS2
      Label template:   ZKEY.XTS2.<seqno>
    Non-XTS:            ZKEY-NONXTS
      Label template:   ZKEY.NONXTS.<seqno>
  Identity key:         ECC (secp521r1)
  Registered key label: (registration required)
```

Also, an identity key that is used to identify the zkey client to EKMF Web is automatically generated locally using the properties that are defined in the identity key template from EKMF Web.

In the next step, generate a certificate that identifies this zkey instance to EKMF Web.

4. Generate a certificate with which to register the zkey client with EKMF Web. The zkey utility automatically uses the identity key from the last step when generating the certificate.

   The registration certificate is an X.509 certificate that is generated with the secure identity key as follows:

   a. Use the `--gen-csr` option to generate a certificate-signing request (CSR) with the identity key. For example:

   ```
   # zkey kms configure --gen-csr csr.pem --cert-subject "CN=Example Name;O=Myorg;C=us"
   ```

   b. Pass the resulting CSR to a certificate authority (CA) to have it issue a CA-signed certificate for the EKMF Web plug-in.

   Alternative for test setups: You can use the `--gen-self-signed-cert` option to generate a self-signed certificate with the identity key for the EKMF Web plug-in. Self-signed certificates should be used for testing only. Use the `--cert-subject` to specify the certificate subject name, and optionally the `--cert-extensions` option to specify extensions.

   To renew an existing certificate, use the `--renew-cert` option. The subject name and extensions are then read from the certificate.

5. Register the zkey client with EKMF Web.

   a) Find out whether the identity key template uses label tags other than the `<seqno>` tag.

   Registering the client automatically generates an identity key on EKMF Web with the public key from the certificate. If the identity key template in EKMF Web uses label tags other then the sequential numbering, `<seqno>`, you must specify them using the `-T` option when you register the client. To find out what the label tags are, use the following command:

   ```
   # zkey kms info
   ```

   b) Register the client with EKMF Web.

   You receive a certificate file from the CA or from the self-signed certificate process. To use this certificate to register with EKMF Web, use a command of the form:

   ```
   # zkey kms configure --register <cert_file>
   ```

   For example, assuming the certificate file is called `cert.pem`:

```
# zkey kms configure --register cert.pem
```

Using **`zkey kms info`** again, you can see that no more configuration steps are open, and this instance of zkey is registered with EKMF Web with the key in the **Registered key label** field, for example:

```
# zkey kms info
...
Registered key label: ZKEY.ID.EC.00001
```

This key identifies this instance of zkey to EKMF Web. The certificate contains the public key of the zkey instance so that EKMF Web now knows the public key.

Results

Now zkey and EKMF Web are set up, and you can generate keys.

Chapter 4. Using zkey with EKMF Web

With the zkey utility you can perform all the tasks on EKMF Web to manage your keys including generating, listing, changing properties, and removing keys.

Before you begin

The tasks described in the following assume that the zkey repository is bound to EKMF Web.

About this task

The following topics provide details about the tasks:

- "Generating keys" on page 21
- "Changing key properties" on page 22
- "Listing keys" on page 23
- "Renaming a key" on page 24
- "Refreshing keys" on page 25

These tasks might also be of interest:

- Chapter 5, "Sharing an EKMF Web key with another system," on page 27
- Chapter 6, "Recovering a secure key repository," on page 31
- Appendix A, "Changing the master key," on page 47

Generating keys

Keys are generated in EKMF Web, and stored in the zkey repository. Properties that you define for a key, such as the description or the volume, are transferred to EKMF Web.

Before you begin

You need to know:

- The volumes that you want to encrypt.
- The type of key you want to generate.

About this task

EKMF Web cannot import existing zkey keys. Keys that were generated locally before the repository was bound to EKMF Web are marked as local, and can be used only on the Linux instance on which zkey runs.

Procedure

1. Optional: Find out whether the key template uses label tags other than `<seqno>`.

   If the key template in EKMF Web uses label tags other then the sequential numbering, `<seqno>`, you must specify them using the `-T` option when you generate the key. To find out what the label tags are, use the following command:

   ```
   # zkey kms info
   ```

2. Use the **`zkey generate`** command. You must specify a name. Issue a command of the form:

   ```
   # zkey gen --name <name>
   ```

By default, the **zkey gen** command generates the new key in EKMF Web and imports it into the zkey repository.

To generate a key on the local zkey repository only, use the `--local` option. Local keys cannot be imported into EKMF Web.

Keys that are generated in EKMF Web are always of type CCA–AESCIPHER. The cryptographic size of the keys depends on the underlying EKMF Web template.

For example, assuming you want to encrypt a volume `/dev/dasdb1` with the device mapper name `enc_disk` and generate an XTS key for this encryption, issue:

```
# zkey gen --name emkf-dasdb1 --xts --volumes /dev/dasdb1:enc_disk --description "XTS key for
DASD B1"
```

Results

The key is saved in EKMF Web with its properties. You can reuse the key for another system.

After the key is generated you can use the **kms list** command to see its properties, such as the two parts of the XTS key:

```
# zkey list
Key                          : emkf-dasdb1
--------------------------------------------------------------------------------------
        Description          : XTS key for DASD B1
        Secure key size      : 272 bytes
        Clear key size       : 512 bits
        XTS type key         : Yes
        Key type             : CCA-AESCIPHER
        Volumes              : /dev/dasdb1:enc_disk
        APQNs                : 08.002f
                               09.002f
        Key file name        : /etc/zkey/repository/emkf-dasdb1.skey
        Sector size          : (system default)
        Volume type          : LUKS2
        Verification pattern : 709bc1e20e34f940362761141e094c65
                               d15bc6cc177d88e7c704577df96d1484
        KMS                  : EKMFWeb
        KMS key label        : ZKEY.XTS1.00002
                               ZKEY.XTS2.00002
        Created              : 2021-03-17 17:31:14
        Changed              : (never)
        Re-enciphered        : (never)
```

Changing key properties

To change a property of a key, use the **zkey change** command

About this task

Properties that you change with the **zkey change** command are updated in EKMF Web.

Some properties depend on the system, such the AP queues. These are not stored in EKMF Web.

Other properties represent the same physical entity, but need different names on different systems, for example, the same physical volume can be mounted to two Linux instances under different names. The same key can be imported on another Linux instance, to another zkey repository, under a different name.

For details about the **zkey change** command, see the **zkey** command reference in *Pervasive Encryption for Data Volumes*, SC34-2782, or the man page.

Procedure

1. Optional: List the key properties.
   For example, assuming the key generated before:

```
# zkey list
Key                          : emkf-dasdb1
-------------------------------------------------------------------------------------
        Description          : XTS key for DASD B1
        Secure key size      : 272 bytes
        Clear key size       : 512 bits
        XTS type key         : Yes
        Key type             : CCA-AESCIPHER
        Volumes              : /dev/dasdb1:enc_disk
        APQNs                : 08.002f
                               09.002f
        Key file name        : /etc/zkey/repository/emkf-dasdb1.skey
        Sector size          : (system default)
        Volume type          : LUKS2
        Verification pattern : 709bc1e20e34f940362761141e094c65
                               d15bc6cc177d88e7c704577df96d1484
        KMS                  : EKMFWeb
        KMS key label        : ZKEY.XTS1.00002
                               ZKEY.XTS2.00002
        Created              : 2021-03-17 17:31:14
        Changed              : (never)
        Re-enciphered        : (never)
```

You can change the description, the volume, the volume type, and the sector size. You cannot change the name with the **change** command. For how to rename a key, see "Renaming a key" on page 24.

2. Specify the **zkey change** and the name of the key, followed by the property you want to change. For example, to change the key description:

```
# zkey change -N emkf-dasdb1 -d "XTS key for some other DASD"
```

Results

The key now has the new description in the zkey repository as well as on EKMF Web:

```
# zkey list
Key                          : emkf-dasdb1
-------------------------------------------------------------------------------------
        Description          : XTS key for some other DASD

                ...
        Created              : 2021-03-17 17:31:14
        Changed              : 2021-03-18 12:08:10
        Re-enciphered        : (never)
```

Listing keys

Use the **zkey kms list** command to display eligible secure keys that are managed by EKMF Web. These keys can, but must not be in the zkey repository.

About this task

You can filter the displayed list by:

- Key label, option -B or `--label`
- Key name, option -N or `--name`
- Associated volumes, option `-l` or `--volumes`
- Volume type, option `-t` or `--volume-type`
- State, option `-s` or `--states`

Most of these options are the same as for the **zkey list** command. For details about the filter options, see Appendix B, "zkey kms - Managing secure keys with a KMS plug-in," on page 49, *Pervasive Encryption for Data Volumes*, SC34-2782, or the zkey man page.

Use the `--states` option to filter the list by the key state in EKMF Web. You can specify multiple states, separated by comma. By default, keys in ACTIVE state are displayed.

By default, only keys are displayed that this zkey client is allowed to use. This is controlled by export control options. When the export control options include the identity key of this zkey client as allowed exporting key, can the key be used by this zkey client. Specify the `--all` option to include keys that this zkey client is not allowed to use. The EKMF Web operator can change the export control options of a key to allow a certain zkey identity key to export the key.

Procedure

- To list all active keys the zkey instance can use, issue **`zkey kms list`**, for example:

```
# zkey kms list
Name                         : emkf-test
-------------------------------------------------------------------------------------
        Key label            : ZKEY.XTS1.00002
                               ZKEY.XTS2.00002
        Description          : A key generated in EKMF Web
        Key size             : 512 bits
        XTS type key         : Yes
        Key type             : CCA-AESCIPHER
        Volumes              : /dev/dasdb1:enc_disk
        Volume type          : LUKS2
        Sector size          : (system default)
        Addl. infos          : State: ACTIVE
                               Exporting keys: ZKEY.ID.EC.00001

Name                         : emkf-test2
-------------------------------------------------------------------------------------
        Key label            : ZKEY.XTS1.00003
                               ZKEY.XTS2.00003
        Description          : 2nd key generated in EKMF Web
        Key size             : 512 bits
        XTS type key         : Yes
        Key type             : CCA-AESCIPHER
        Volumes              : /dev/dasda1:enc_disk
        Volume type          : LUKS2
        Sector size          : (system default)
        Addl. infos          : State: ACTIVE
                               Exporting keys: ZKEY.ID.EC.00001
```

  For more information on how to make keys available to a zkey instance, see Chapter 5, “Sharing an EKMF Web key with another system,” on page 27.

- To filter the list by name, specify a name, or part of a name, for example:

```
# zkey kms list -N "ekmf*"
```

  This command would result in the same list as in the example before.

Renaming a key

To change the name of a key, use the **`zkey rename`** command.

About this task

The zkey name is specific to the Linux instance with the zkey repository you are working with. The name is stored in EKMF Web for this instance.

For more information about the **`zkey rename`** command, see the **`zkey`** command reference in *Pervasive Encryption for Data Volumes*, SC34-2782, or the man page.

Procedure

Specify the name of the key and the new name. Issue a command of the following form:

```
# zkey rename -N <name> -w <new_name>
```

For example, to rename the key with the name `emkf-dasdb1` into `emkf-dasdb2`:

```
# zkey rename -N emkf-dasdb1 -w emkf-dasdb2
```

Results

The key name is changed in the zkey repository and in EKMF Web. Use the **zkey list** command to check the name:

```
# zkey list
....
Name                         : emkf-dasdb2
....
```

Renaming a key changes only the name. EKMF Web still knows which key it is and any other changes feed through.

Refreshing keys

Use the **zkey kms refresh** command to refresh secure keys that are bound to EKMF Web.

About this task

Refreshing a key updates the secure key by reimporting it from EKMF Web.

You can filter the list of keys to be refreshed by:

- Key name, option `-N` or `--name`
- Key type, option `-K` or `--key-type`
- Associated volumes, option `-l` or `--volumes`
- Volume type, option `-l` or `--volume-type`

These options are the same as for other **zkey kms** commands. For details about the filter options, see Appendix B, “zkey kms - Managing secure keys with a KMS plug-in,” on page 49, *Pervasive Encryption for Data Volumes*, SC34-2782, or the zkey man page.

Procedure

- To refresh a key, issue a command of the form:

  ```
  # zkey kms refresh -N <name>
  ```

  You can use wildcards to refresh several keys.
  For example, to refresh all keys whose names start with "sec", issue:

  ```
  zkey kms refresh –N "sec*"
  ```

- To refresh key properties, use the `-P` option
  Refreshing updates the information on the zkey repository with the information from EKMF Web including the description, associated volumes, volume type, and sector size.

  Refreshing the properties is useful when key properties have changed through the EKMF Web UI, or by zkey on another system (for shared keys).

Chapter 5. Sharing an EKMF Web key with another system

A key that is generated in EKMF Web can be reused on other Linux instances. This sharing is useful for backup, recovery, and cloning setups.

Before you begin

The Linux instance that you want to share the key with, or reuse the key on, must be connected to the same EKMF Web instance as the original Linux instance from which the key was generated, see "Connecting zkey with EKMF Web" on page 15.

You need to know the label of the identity key of the second Linux instance. To find out what the label is, on that Linux instance issue the command **`zkey kms info`**, for example:

```
# zkey kms info
KMS-Plugin:             EKMFWeb
  ...
  Identity key:         ECC (secp521r1)
  Registered key label: ZKEY.ID.EC.00025
```

The identity key label is the last entry in the list of information.

About this task

A key can be imported to other Linux instances. Use this capability when you set up backup or recovery systems. System-specific properties, such as the volume, might need to be changed on the imported key.

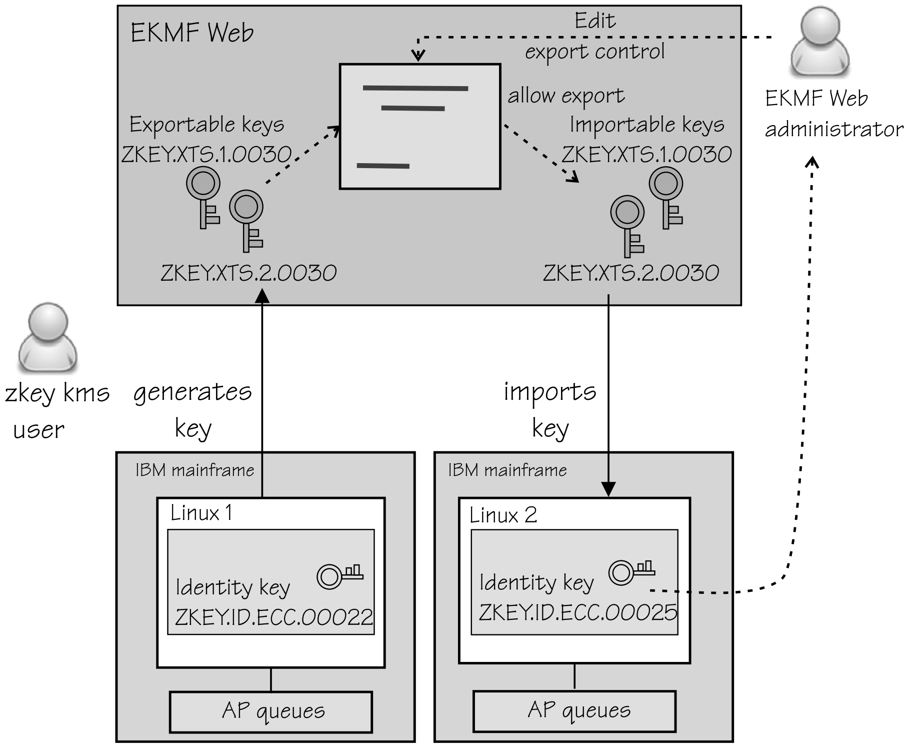
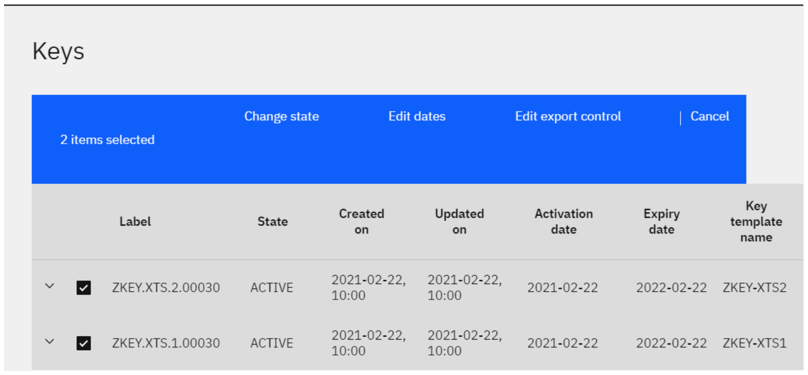


Figure 11. A key can be reused on another Linux instance

The imported key is protected by a transport key when it is sent to the second Linux instance, and then reencrypted with the master key of the AP queues the second instance uses.

Procedure

1. The EKMF Web administrator must allow the keys to be exported from EKMF Web.
   a) On EKMF Web, go to **Key management** on the left navigation bar.
   b) Go to **Keys**
   c) Select the keys for which to edit export control.
      For example, assume you want to export XTS keys ZKEY.XTS.1.0030 and ZKEY.XTS.2.0030 that were generated by the Linux instance with the identity key ZKEY.ID.ECC.0022 to the Linux instance with the identity key ZKEY.ID.ECC.0025. Select the keys as shown in Figure 12 on page 28.

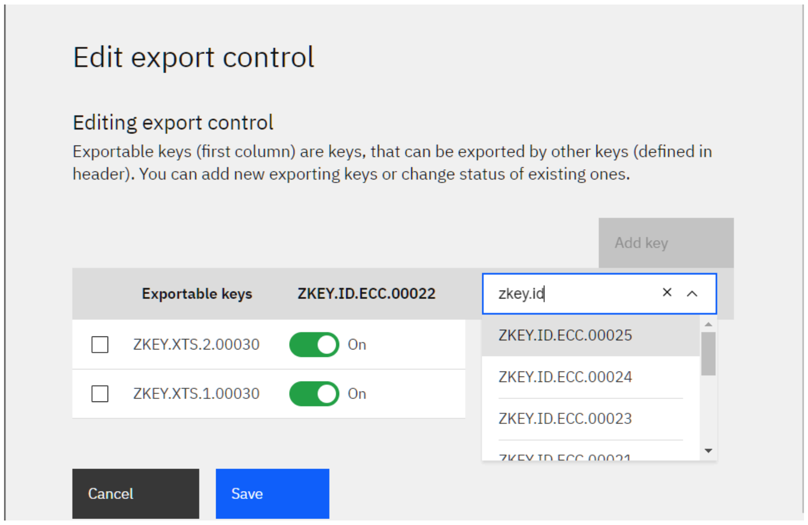

Keys

2 items selected | Change state | Edit dates | Edit export control | Cancel

		Label	State	Created on	Updated on	Activation date	Expiry date	Key template name
⌄	☑	ZKEY.XTS.2.00030	ACTIVE	2021-02-22, 10:00	2021-02-22, 10:00	2021-02-22	2022-02-22	ZKEY-XTS2
⌄	☑	ZKEY.XTS.1.00030	ACTIVE	2021-02-22, 10:00	2021-02-22, 10:00	2021-02-22	2022-02-22	ZKEY-XTS1

Figure 12. Select the keys for which you want to change export control

   d) Select **Edit export control**.
   e) On the Export control page, click **Add key**.
   f) In the field that opens, type the name of identity key you want to work with
      For example, assuming you want to export the keys to the Linux instance with the identity key ZKEY.ID.ECC.0025, as shown in Figure 13 on page 28.


Edit export control

Editing export control

Exportable keys (first column) are keys, that can be exported by other keys (defined in header). You can add new exporting keys or change status of existing ones.

Add key

	Exportable keys	ZKEY.ID.ECC.00022
☐	ZKEY.XTS.2.00030	On
☐	ZKEY.XTS.1.00030	On

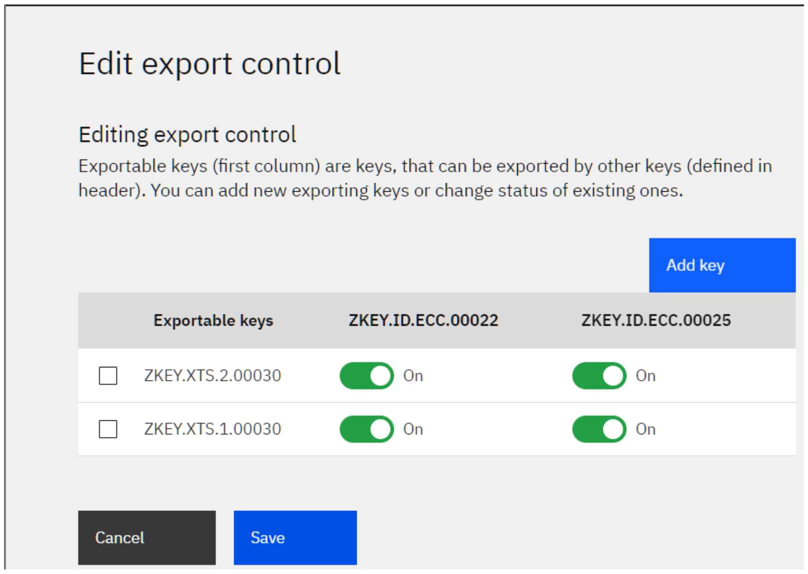
zkey.id

ZKEY.ID.ECC.00025
ZKEY.ID.ECC.00024
ZKEY.ID.ECC.00023
ZKEY.ID.ECC.00021

Figure 13. Add the identity key of the Linux instance on which you want to reuse the keys

   g) Click **Save**

   In the example shown in Figure 14 on page 29, the Linux instance with the identity key ZKEY.ID.ECC.0025 can now import the XTS keys.



Edit export control

Editing export control

Exportable keys (first column) are keys, that can be exported by other keys (defined in header). You can add new exporting keys or change status of existing ones.

	Exportable keys	ZKEY.ID.ECC.00022	ZKEY.ID.ECC.00025
☐	ZKEY.XTS.2.00030	On	On
☐	ZKEY.XTS.1.00030	On	On

Figure 14. The identity key of the Linux instance on which you want to reuse the keys is added

2. Optional: On the second Linux instance, check that the keys you want to use are available for importing.

   To check that the key you wanted to import is available, use the **zkey kms list** command.

   For example, the XTS keys the EKMF Web administrator allowed export for now show up in the list:

```
# zkey kms list
Name                           : emkf-dasdb1
------------------------------------------------------------------------------------
        Key label              : ZKEY.XTS1.0030
                                 ZKEY.XTS2.0030
        Description            : XTS key for DASD B1
        Key size               : 512 bits
        XTS type key           : Yes
        Key type               : CCA-AESCIPHER
        Volumes                : /dev/dasdb1:enc_disk
        Volume type            : LUKS2
        Sector size            : (system default)
        Addl. infos            : State: ACTIVE
                                 Exporting keys: ZKEY.ID.EC.00022
                                                 ZKEY.ID.EC.00025
```

   The keys you want to import must be in the ACTIVE state.

3. On the second Linux instance, use zkey to import the keys that you want to use. Issue the **zkey kms import** command and specify the key label.
   For example, to import the keys with labels starting with ZKEY.XTS, issue a command as follows:

```
# zkey kms import -B "ZKEY.XTS*"
```

   You can filter the list of keys to import, by specifying the `-N` (name), `-B` (label), `-l` (volume), and `-t` (type) options with the import command.

   For example, to import the key named emkf-dasdb1 of type LUKS2:

```
# zkey kms import --name emkf-dasdb1 -t LUKS2
```

4. Change system-specific properties.

   The volume and the name are system-specific and might have to be changed. AP queues are bound to the key management system, and are set automatically.

   If needed, change the volume with the **zkey change** command, for example:

```
# zkey change --name emkf-dasdb1 \
-l /dev/mapper/disk2:enc-disk2
```

To change the key name, use the **`zkey kms rename`** command, see "Renaming a key" on page 24.

Results

You can now use the XTS keys on the second Linux instance.

Chapter 6. Recovering a secure key repository

Restore a corrupted repository or create a backup repository.

Before you begin

This scenario assumes:

- Your organization uses EKMF Web
- You have a user ID and password for EKMF Web

About this task

Assume you have an EKMF Web and pervasive encryption solution on Linux on IBM Z, but your data center is flooded. Now your secure key repository is gone. But luckily, your encrypted volumes are safe, and you use EKMF Web, so your keys are safe.

You can recover a secure key repository by setting up a new repository on a new Linux instance, and reimporting the keys from EKMF Web.

Procedure

1. Install a new Linux instance with zkey. For example, you can use Red Hat Enterprise Linux as of version 8.4 or SUSE Linux Enterprise Server as of 15.3
2. On the new Linux instance, bind zkey to EKMF Web.

   For details, see "Connecting zkey with EKMF Web" on page 15.
3. On the new Linux instance, configure zkey.

   For details, see "Configuring an EKMF Web plug-in" on page 15.
4. On EKMF Web, the administrator must allow the keys from your old secure key repository to be exported from EKMF Web into the new secure key repository.
5. On the new Linux instance, import the keys by using the **zkey kms import** command.

   To import all eligible keys, issue the following command:

   ```
   # zkey kms import
   ```

   For details on how to refresh only certain keys, see "zkey kms import" on page 57.

Results

Your secure key repository is populated with the keys from EKMF Web, and you can resume working.

If the host name of the newly installed system remains the same, the key names and key properties of the imported keys also remain the same as on the original system.

If the host name is now different, some of the key properties might need adapting to the current system. Use the **zkey change** and **zkey rename** commands to adapt the key name and properties as needed.

Part 3. Using the KMIP plug-in

The Key Management Interoperability Protocol (KMIP) from OASIS aims to establish a standard for client/server communication for storage and maintenance of cryptographic keys, certificates, and secret objects. A key-management system plug-in for KMIP is available for Linux on IBM Z and LinuxONE.

Newest version

You can find the newest version of this publication at: ibm.com/docs/en/linux-on-systems

Chapter 7. The KMIP plug-in

A key-management system plug-in for the Key Management Interoperability Protocol (KMIP) is available for Linux on IBM Z and LinuxONE.

KMIP simplifies encryption key management. Keys can be created on the server and then retrieved, wrapped by transportation keys.

Keys are never exposed in plain text, and are only available to authorized parties, where both the key protection and the authentication of authorized parties is secured by HSMs.

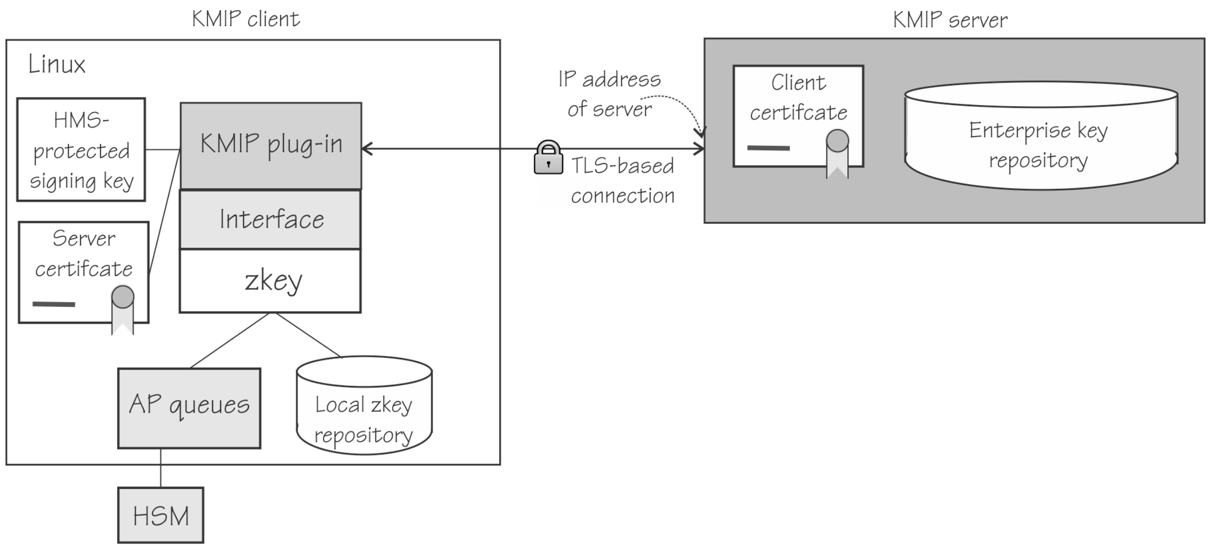


Figure 15. The KMIP plug-in integrates with zkey on Linux

Figure 15 on page 35 illustrates how the KMIP plug-in integrates with zkey on Linux over a secure connection.

Chapter 8. Configuring the KMIP plug-in

Configuring the KMIP plug-in entails configuring AP queues, identity key information, and the KMIP server information.

About this task

The following must be configured for a KMIP plug-in:

- AP queues
- Supported key types
- Identity key information (type, size, curve)
- Client certificate information
- KMIP server information (host name, vendor string, KMIP Version, transport/encoding)
- TLS settings
- (Optional) Wrapping key information (type, size, unique ID in KMIP)

For more details about the **zkey kms** command, see Appendix B, "zkey kms - Managing secure keys with a KMS plug-in," on page 49.

Procedure

1. Optional: Check which plug-ins are available. Issue the **zkey kms plugins** command, for example:

```
# zkey kms plugins
KMS-Plugin                 Shared library
---------------------------------------------------
KMIP                       zkey-kmip.so
```

   The example shows that the KMIP plug-in is available. It also shows the shared library that implements the plug-in.
2. To bind to the KMIP plug-in, issue the following command:

```
# zkey kms bind KMIP
```

   The local secure key repository is bound to a key management system of type KMIP.

   Note: A secure key repository can be bound to only one key management system.
3. Associate AP queues with the key-management plug-in.

   You can specify either CCA or EP11 AP queues, but not both. To specify the AP queues to associate with the KMIP plug-in, use a command of the form:

```
# zkey kms configure --apqns <adapter1.domain1,adapter2.domain2,...>
```

   The AP queue is defined by its adapter ID and domain ID. If you specify multiple AP queues, they must have the same master key.

   For example, to add AP queues from adapter 08 and 09, both with domain 002f, issue:

```
# zkey kms configure --apqns 08.002f,09.002f
```

   To see whether the AP queues are configured, use **zkey kms info**:

```
# zkey kms info
KMS-Plugin:             KMIP
  Supported key types:  CCA-AESCIPHER
  APQNs:                08.002f
                        09.002f
  KMIP server:         (configuration required)
```

4. Create a client certificate for communication with the KMIP server.

   You must register the certificate with the server. The KMIP server accepts communication with the client only after the client's certificate is registered.

   a) Generate a certificate-signing request (CSR).

      Issue a command of the form:

      ```
      # zkey kms configure  --gen-csr <request_file> --cert-subject  <subject>
      ```

      For example, to generate a certificate-signing request with a new identity key that uses the defaults ECC and secp521r1, and store it in a file that is named `csr.pem`, issue:

      ```
      # zkey kms configure --gen-csr csr.pem --cert-subject SUBJECT-RDNS
      ```

      The generated CSR is stored into the specified `csr.pem` file.

      Alternative for test setups: You can use the `--gen-self-signed-cert` option to generate a self-signed certificate with the identity key for the KMIP plug-in. Self-signed certificates should be used for testing only. Use the `--cert-subject` to specify the certificate subject name, and optionally the `--cert-extensions` option to specify extensions.

   b) Have a CA sign this request and issue a client certificate.

5. Register the client certificate to use for communication with the server. The detailed registration steps depends on the KMIP server. Refer to your server documentation for details and use the **`zkey kms info`** command to display information about the server and its configuration.

   Issue a command of the form:

   ```
   # zkey kms configure --client-cert <client_certificate>
   ```

   For example, assume that the client certificate is stored in a file called `CERT-PEM-FILE`. Issue:

   ```
   # zkey kms configure --client-cert CERT-PEM-FILE
   ```

6. Configure the KMIP server that you want to communicate with.

   This step assumes that the client certificate has been set up and registered with the KMIP server.

   Use the TLS options to pin the server, ensuring that it is always the same server in future communication.

   ```
   # zkey kms configure --kmip-server <KMIP_server> --tls-pin-server-pubkey
   ```

   You can specify a KMIP server profile. For more information about profiles, see "KMIP plug-in profiles" on page 39.

Results

The connection to the server is established, and information from the server is retrieved. An RSA key pair is generated automatically when the KMIP server connection is configured. The private key of this pair will be used as a wrapping key for retrieving keys from KMIP. The public key is retrieved by the KMIP plug-in, and stored in the zkey key repository, and is used for unwrapping the retrieved keys.

If you need to, you can generate a new wrapping key at any time. Use the command '**`zkey kms configure --gen-wrapping-key [--label <name>]`** to re-generate a wrapping key.

Use the `--label` option to provide a human-readable name for the key.

KMIP plug-in profiles

Use KMIP plug-in profiles to control parameters and settings for communicating with the KMIP server.

Your KMIP server might provide profiles for the plug-in. Profiles are located in `/etc/zkey/kmip/profiles` on your Linux instance. You can override the location with environment variable ZKEY_KMIP_PROFILES.

KMIP plug-in profiles are text files with the naming scheme `<product-name>.profile`. You can provide your own profile. For more information on what a profile must contain, see the zkey-kmip man page.

zkey applies the default profile unless there is a user-specified profile, or a profile can be matched based on the server information. zkey provides profiles for the following servers:

- IBM Security Guardium Key Lifecycle Manager (IBM SKLM)
- IBM Security Guardium Data Encryption (IBM GDE)

Chapter 9. Using zkey with KMIP

With the zkey utility you can perform all the tasks on KMIP to manage your keys including generating, listing, changing properties, and removing keys.

Before you begin

The tasks described in the following assume that the zkey repository is bound to KMIP.

About this task

The following topics provide details about the tasks:

- "Generating keys with KMIP" on page 41
- "Changing key properties" on page 42
- "Listing keys" on page 43
- "Renaming a key in the repository" on page 44
- "Refreshing keys on KMIP" on page 45
- "Importing secure keys from KMIP" on page 45

Key access rights are specific to the KMIP server implementation. See your server documentation for details about how to control access to keys.

For information about how to change a master key, see Appendix A, "Changing the master key," on page 47

Generating keys with KMIP

Keys are generated in the KMIP server, and stored in the zkey repository.

Before you begin

You need to know the type of key you want to generate.

The supported key types depend on the AP queue types bound to the plug-in, CCA or EP11.

When the KMS is bound to CCA AP queues, you can chose whether to generate CCA AES DATA keys or CCA AES CIPHER by using the `--key-type` option.

Procedure

Use the **`zkey generate`** command.

You can specify a name with the `--label` option. The label specifies the name of the key in the KMIP server, whereas `--name` specifies the name of the key in zkey.

Issue a command of the form:

```
# zkey gen --key-type <type> --label <name>
```

The type of the generated key depends on the chosen APQN types. For CCA APQNs, the command generates a CCA-AESDATA key by default, but for EP11 APQNs it generates an EP11-AES key. The new key is generated on the KMIP server and imported into the zkey repository.

To generate a key on the local zkey repository only, use the `--local` option. Local keys cannot be imported into KMIP.

For example, assuming you want to encrypt a block device `/dev/dasdc1` with the device mapper name `enc_vol` and generate an XTS key for this encryption, issue:

```
# zkey gen --name seckey --volumes /dev/dasdc1:encvol --label TEST1:TEST2
```

The command generates a secure AES key in the KMIP server using the labels "TEST1" and "TEST2" and stores the XTS key in the secure-key repository using the name "seckey" and associates it with block device `/dev/dasdc1` and device-mapper name "encvol".

Results

The key is saved in KMIP with its properties. You can reuse the key for another system.

After the key is generated you can use the **`kms list`** command to see its properties:

```
# zkey list
Key                           : seckey
---------------------------------------------------------------------------------------
        Description           : AES key for DASD C1
        Secure key size       : 272 bytes
        Clear key size        : 512 bits
        XTS type key          : Yes
        Key type              : CCA-AESCIPHER
        Volumes               : /dev/dasdc1:encvol
        APQNs                 : 08.002f
                                09.002f
        Key file name         : /etc/zkey/repository/kmip-dasdc1.skey
        Sector size           : (system default)
        Volume type           : LUKS2
        Verification pattern  : 709bc1e20e34f940362761141e094c65
                                d15bc6cc177d88e7c704577df96d1484
        KMS                   : KMIP
        KMS key label         : TEST1
                                TEST2
        Created               : 2022-03-23 16:17:14
        Changed               : (never)
        Re-enciphered         : (never)
```

Changing key properties

To change a property of a key, use the **zkey change** command

About this task

Properties that you change with the **zkey change** command are updated in KMIP.

You can change the description, the volume, the volume type, and the sector size. You cannot change the name with the **change** command. For how to rename a key, see "Renaming a key in the repository" on page 44.

You cannot change the AP queues for a key that is bound to a KMIP plug-in. To change the AP queues, use the **`zkey kms configure`** command with the `--`apqns option.

Other properties represent the same physical entity, but need different names on different systems, for example, the same physical volume can be mounted to two Linux instances under different names. The same key can be imported on another Linux instance, to another zkey repository, under a different name.

For details about the **zkey change** command, see the **zkey** command reference in *Pervasive Encryption for Data Volumes*, SC34-2782, or the man page.

Procedure

1. Optional: List the key properties.
   For example, assuming the key generated before:

```
# zkey list
Key                           : emkf-dasdb1
------------------------------------------------------------------------------------
        Description           : AES key for DASD C1
        Secure key size       : 272 bytes
        Clear key size        : 512 bits
        XTS type key          : Yes
        Key type              : CCA-AESCIPHER
        Volumes               : /dev/dasdb1:enc_disk
        APQNs                 : 08.002f
                                09.002f
        Key file name         : /etc/zkey/repository/kmip-dasdc1.skey
        Sector size           : (system default)
        Volume type           : LUKS2
        Verification pattern  : 709bc1e20e34f940362761141e094c65
                                d15bc6cc177d88e7c704577df96d1484
        KMS                   : KMIP
        KMS key label         : TEST1
                                TEST2
        Created               : 2022-03-23 17:31:14
        Changed               : (never)
        Re-enciphered         : (never)
```

You can change the description, the volume, the volume type, and the sector size. You cannot change the name with the **change** command. For how to rename a key, see "Renaming a key in the repository" on page 44.

2. Use the **zkey change** and specify the name of the key, followed by the property you want to change. For example, to change the key label:

```
# zkey change -N seckey -l "This is my secret key"
```

Results

The key now has the new description in the zkey repository as well as on the KMIP server:

```
# zkey list
Key                           : seckey
------------------------------------------------------------------------------------
        Description           : This is my secret key

                ...
        Created               : 2022-03-17 17:31:14
        Changed               : 2022-03-18 12:08:10
        Re-enciphered         : (never)
```

Listing keys

Use the **zkey kms list** command to display eligible secure keys that are managed by KMIP. These keys can, but must not be in the zkey repository.

About this task

You can filter the displayed list by:

- Key label, option -B or --label
- Key name, option -N or --name
- Associated volumes, option -l or --volumes
- Volume type, option -t or --volume-type

Most of these options are the same as for the **zkey list** command. For details about the filter options, see Appendix B, "zkey kms - Managing secure keys with a KMS plug-in," on page 49, *Pervasive Encryption for Data Volumes*, SC34-2782, or the zkey man page.

The KMIP server implementation determines how keys are associated to certain clients or groups of clients, and controls who can see and access which keys. For more details about how to control access to keys in the KMIP server, refer to the documentation of your KMIP server.

Procedure

- To list all active keys the zkey instance can use, issue **zkey kms list**, for example:

```
# zkey kms list
Name                          : kmip-test
------------------------------------------------------------------------------------
        Key label             : TEST-KEY1
        Description           : A key generated in KMIP
...

Name                          : kmip-test2
------------------------------------------------------------------------------------
        Key label             : TEST-KEY2
        Description           : 2nd key generated in KMIP
        Key size              : 256 bits
        XTS type key          : No
 ...
```

- To filter the list by label, for example:

```
# zkey kms list --label "*LUKS2*"
```

  The command displays eligible secure keys managed by the KMIP server, where the label name contains the word "LUKS2".

Renaming a key in the repository

To change the name of a key, use the **zkey rename** command.

About this task

The zkey name is specific to the Linux instance with the zkey repository you are working with.

For more information about the **zkey rename** command, see the **zkey** command reference in *Pervasive Encryption for Data Volumes*, SC34-2782, or the man page.

Note: The key label as it is known in the KMIP server cannot be changed. Only the associated zkey name is updated.

Procedure

Specify the name of the key and the new name. Issue a command of the following form:

```
# zkey rename -N <name> -w <new_name>
```

For example, to rename the key with the name `kmip-dasdb1` into `kmip-dasdb2`:

```
# zkey rename -N kmip-dasdb1 -w kmip-dasdb2
```

Results

The key name is changed in the zkey repository. Use the **zkey list** command to check the name:

```
# zkey list
....
Name                          : kmip-dasdb2
....
```

Refreshing keys on KMIP

Use the **zkey kms refresh** command to refresh secure keys that are bound to KMIP.

About this task

Refreshing a key updates the secure key by reimporting it from the KMIP server.

The **zkey kms refresh** command can be useful if the secure keys were not been reenciphered properly after a CCA or EP11 master key change, and thus became invalid. The **zkey kms refresh** command reimports the secure key under the current CCA or EP11 master key. Hence, you can use this command as an alternative to the **zkey reencipher** command for keys that are bound to a KMIP plug-in.

You can filter the list of keys to be refreshed by:

- Key name, option `-N` or `--name`
- Key type, option `-K` or `--key-type`
- Associated volumes, option `-l` or `--volumes`
- Volume type, option `-l` or `--volume-type`

These options are the same as for other **zkey kms** commands. For details about the filter options, see Appendix B, "zkey kms - Managing secure keys with a KMS plug-in," on page 49, *Pervasive Encryption for Data Volumes*, SC34-2782, or the zkey man page.

Procedure

- To refresh a key, issue a command of the form:

```
# zkey kms refresh -N <name>
```

  You can use wildcards to refresh several keys.
  For example, to refresh all keys whose names start with "sec", issue:

```
zkey kms refresh –N "sec*"
```

- To refresh key properties, use the `-P` option
  Refreshing updates the information on the zkey repository with the information from the KMIP server. including the description, associated volumes, volume type, and sector size.

Importing secure keys from KMIP

Restore a corrupted repository or create a backup repository.

About this task

Assume you have an KMIP and pervasive encryption solution on Linux on IBM Z. You already have keys on the KMIP server, and you would like to import them into a zkey instance.

You can filter the keys by:

- Key label, option `-B` or `--label`
- Key name, option `-N` or `--name`
- Associated volumes, option `-l` or `--volumes`
- Volume type, option `-t` or `--volume-type`

These options are the same as for the **zkey list** command. For details about the filter options, see Appendix B, "zkey kms - Managing secure keys with a KMS plug-in," on page 49, *Pervasive Encryption for Data Volumes*, SC34-2782, or the zkey man page.

Procedure

1. Install a new Linux instance with zkey. For example, you can use Red Hat Enterprise Linux as of version 8.4 or SUSE Linux Enterprise Server as of 15.3
2. On the new Linux instance, bind zkey to KMIP and configure the plug-in.

   For details, see Chapter 8, "Configuring the KMIP plug-in," on page 37.
3. On KMIP, the administrator must give the new zkey system access to the key to import.
4. On the new Linux instance, import the keys by using the **zkey kms import** command.

   To import all eligible keys, issue the following command:

   ```
   # zkey kms import
   ```

   For details on how to refresh only certain keys, see "zkey kms import" on page 57.

Results

Your secure key repository is populated with the keys from KMIP.

If the host name of the newly installed system remains the same, the key names and key properties of the imported keys also remain the same as on the original system.

If the host name is now different, some of the key properties might need adapting to the current system. Use the **zkey change** and **zkey rename** commands to adapt the key name and properties as needed.

Appendix A. Changing the master key

Changing a master key requires reenciphering all secure keys that are enciphered with it. This includes all secure keys in the secure key repository, but also secure keys used by the plug-in, such as the identity key.

About this task

For EKMF Web: EKMF Web supports CCA master keys. In the following, only CCA applies to EKMF Web.

For KMIP: KMIP key-management systems support both CCA and EP11 master keys.

A new CCA or EP11 master key is set on the cryptographic coprocessor that the AP queues of your Linux instance uses.

- How to change a CCA master key is described in Setting a master key on the cryptographic coprocessor.
- How to change an EP11 master key is described in Setting a master key on the Crypto Express EP11 coprocessor

See also the zkey man page.

When a master key changed, you must reencipher all secure keys that are contained in the secure key repository that are associated with the AP queues for which you change the master key. For identity keys and other KMS plug-in keys, you must use the **`zkey kms reencipher`** command. For keys used to encrypt volumes, you can also use the **`zkey kms refresh`** command.

Procedure

1. Load the new master key into the NEW register using the TKE.
2. Reencipher the identity keys and other KMS plug-in keys of the KMS plug-in.

   Reencipher with the `--staged` option. For example, to reencipher an identity key and other KMS plug-in keys with the master key in the NEW register, issue:

   ```
   # zkey kms reencipher --to-new --staged
   ```
3. Reencipher the keys in the secure key repository.

   For example, to reencipher keys that use the AP queues 08.002f, and 09.002f, issue:

   ```
   # zkey reencipher --apqns 08.002f,09.002f --to-new --staged
   ```
4. On the TKE, make the new master key the active key by moving it into the CURRENT register.
5. Complete the reenciphering of the identity key.

   ```
   # zkey kms reencipher --complete
   ```
6. Complete the reenciphering of the keys in the secure key repository and the KMS keys.

   ```
   # zkey reencipher --apqns 08.002f,09.002f --complete
   ```

   Alternatively, use **`zkey kms refresh`**. The refresh command reimports the key from the KMS using the current master key. You can use the refresh command to reimport keys even if the master keys were already changed.

   Note: The refresh command does not reencipher local keys.
7. You must also re-encipher any secure AES volume keys when the AES master key changes. Use the **`zkey-cryptsetup`** command to do this.

For details about the **zkey-cryptsetup** command, see the command reference in *Pervasive Encryption for Data Volumes*, SC34-2782.

To re-encipher the secure key of the encrypted volume /dev/mapper/disk1 in staged mode and complete it later:

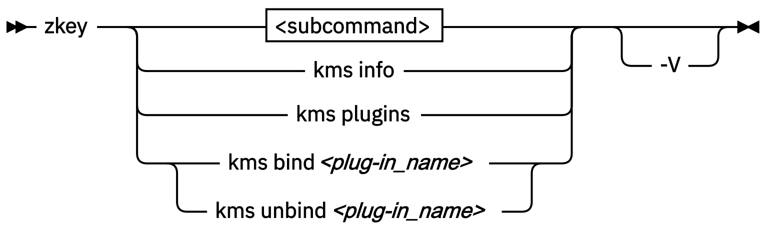
```
# zkey-cryptsetup reencipher /dev/mapper/disk1 --staged

Enter passphrase for '/dev/mapper/disk1': disk1pw
The secure volume key of device '/dev/mapper/disk1' is enciphered with the
CURRENT master key and is being re-enciphered with the NEW master key.
Staged re-enciphering is initiated for device '/dev/mapper/disk1'. After the NEW
master key has been set to become the CURRENT master key, run 'zkey-cryptsetup
reencipher' with option '--complete' to complete the re-enciphering process.

# zkey-cryptsetup reencipher /dev/mapper/disk1 --complete
```

Appendix B. zkey kms - Managing secure keys with a KMS plug-in

The **zkey** command has specific options for the EKMF Web and KMIP plug-ins. See the **zkey** command described in *Pervasive Encryption for Data Volumes*, SC34-2782 for the non-plug-in specific options. Also, see the zkey man page, the zkey-ekmfweb man page, and the zkey-kmip man page.

Syntax

zkey syntax for a KMS plug-in



<subcommand>
is described in the following sections:

- "zkey kms configure" on page 49
- "zkey generate" on page 56
- "zkey kms import" on page 57
- "zkey kms list" on page 58
- "zkey kms reencipher" on page 59
- "zkey kms refresh" on page 60
- "zkey remove" on page 61

kms info
displays information about the plug-in and its configuration.

kms plugins
displays the available KMS plug-ins.

kms bind *<plug-in_name>*
binds the zkey instance to the specified KMS plug-in.

kms unbind *<plug-in_name>*
releases the zkey instance from the specified KMS plug-ins.

-h or --help
displays help information for the command. Specify `zkey` *`<subcommand>`* `-h` to get help for a subcommand. This option also displays KMS-specific option, if any.

-V or --verbose
displays more details.

zkey kms configure

Use the **`zkey kms configure`** command to configure the key-management system plug-in.

In the following, the command is described for different plug-ins:

- For the EKMF Web version, see "zkey kms configure for the EKMF Web plug-in" on page 50.
- For the KMIP version, see "zkey kms configure for the KMIP plug-in" on page 53

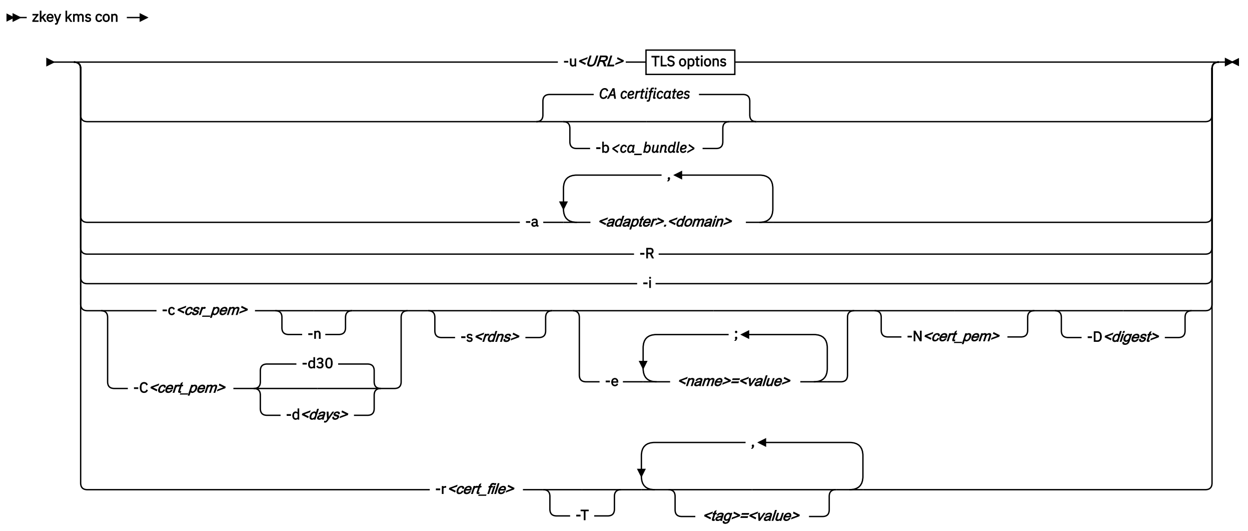
A key management system plug-in might offer plug-in specific options. Use **`kms configure --help`** to display the plug-in specific options and their meaning.

zkey kms configure for the EKMF Web plug-in

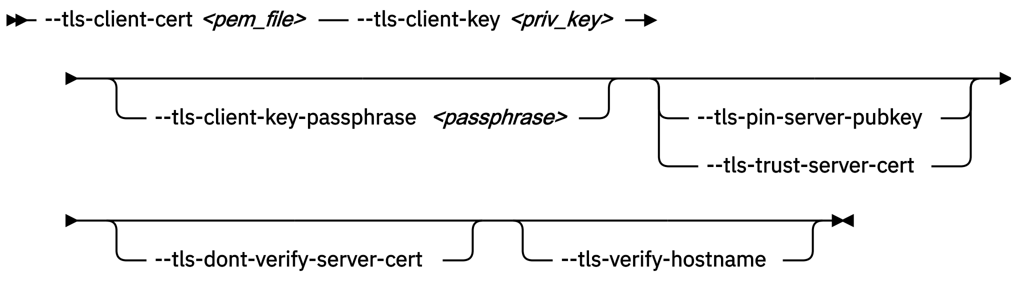
Use the **`zkey kms configure`** command to configure the key-management system plug-in.

In the following, the command is described as it applies when working with the EKMF Web plug-in. For the KMIP plug-in context, see "zkey kms configure for the KMIP plug-in" on page 53.

zkey kms configure syntax for EKMF Web



TLS options


--tls-client-cert <pem_file>
--tls-client-key <priv_key>
--tls-client-key-passphrase <passphrase>
--tls-pin-server-pubkey
--tls-trust-server-cert
--tls-dont-verify-server-cert
--tls-verify-hostname

where:

-u or --ekmfweb-url *<URL>*
: Specifies the URL of the EKMF Web server. The URL starts with `https://`, and can contain a port number, which is separated by a colon. The default port number is 443 for HTTPS.

-a or --apqns <adapter.domain, adapter.domain,...>
: Associates cryptographic adapters (APQNs) in CCA coprocessor mode (APQNs) with the EKMF Web plug-in. You can specify multiple APQNs as a comma-separated list. Each APQN consists of an adapter and domain number separated by a period. All APQNs that you want to set, add, or replace must be online.

To add an APQN to an existing list, prefix the APQN with a plus sign (+).

To remove an APQN from the associated APQNs, prefix the APQN with a minus sign (-).

To set or replace the APQN association do not specify a prefix. You cannot mix + and - in one specification. You can either add, remove, or set the associations with one command.

-b or --tls-ca-bundle *<ca_bundle>*
: Specifies the CA bundle PEM file, or the directory that contains the CA certificates that are used to verify the EKMF Web server certificate during the TLS handshake. If the option specifies a directory path, then this directory must be prepared with OpenSSL's **`c_rehash`** utility.

  The default is to use the system CA certificates.

--tls-client-cert *<pem_file>*
: Specifies the PEM file that contains the client's TLS certificate for use with TLS client authentication.

--tls-client-key *<priv_file>*
: If the PEM file is protected by a pass phrase, use this option to specify the pass phrase to unlock the PEM file that is specified with the `--tls-client-key` option.

--ts-client-key-passphrase *<pass_phrase>*
: Specifies the PEM file that contains the client's private key for use with TLS client authentication.

--tls-pin-server-pubkey
: For CA-signed EKMF Web server certificates: Pins the public key of the EKMF Web server. With a pinned key, it is verified that every connection uses the same EKMF Web server certificate as the one used when the connection to the EKMF Web server was configured.

--tls-trust-server-cert
: Trusts the EKMF Web server's certificate even if it is a self-signed certificate, or it was not verified due to other reasons. Use this option instead of the `--tls-pin-server-pubkey` option when you are using self-signed EKMF Web server certificates.

--tls-dont-verify-server-cert
: For self-signed key-management server certificates used in test environments: Bypasses the server certificate verification by default. For CA-signed server certificates, the default is to verify them.

  This option overrides `--tls-trust-server-cert`.

--tls-verify-hostname
: Verifies that the server certificate's **Common Name** field or a **Subject Alternate Name** field matches the hostname that is used to connect to the server.

-R or --refresh-settings
: Refreshes the EKMF Web server settings. The settings are automatically refreshed when the connection to the EKMF Web server is configured or reconfigured. Use this option when the settings of the configured EKMF Web server changed.

-i or --gen-identity-key
: Generates an identity key for the plug-in. An identity key is automatically generated for you when you configure the EKMF Web server connection. Use this option to generate a new identity key. If you regenerate the identity key you must also regenerate a registration certificate with the newly generated identity key, and reregister this zkey client with the EKMF Web server.

-c or --gen-csr *<csr_pem>*
: Generates a certificate signing request (CSR) with the identity key and store it into the specified PEM file. You pass this CSR to a CA to have it issue a CA signed certificate for the plug-in. You need to register the certificate with the EKMF Web before you can access it.

-C or --gen-self-signed-cert *<csr_pem>*
: Generates a self-signed certificate with the identity key and store it into the specified PEM file. You need to register the certificate with the EKMF Web server before you can access it.

-s or --cert-subject *<rdns>*
: Specifies the subject name for generating a CSR or self-signed certificate, in the form `<type>=<value>(;<type>=<value>)*[;]` with types recognized by OpenSSL.

-e or --cert-extensions *<name>=<value>*
: Specifies the certificate extensions for generating a CSR or self-signed certificate, as a semi-colon-separated list of the form `<name>=[critical,]<value>(;<name>=[critical,]<value>),...` with extension names and values recognized by OpenSSL. You can optionally include the `critical` attribute for any tag.

-N or --renew-cert *<cert_pem>*
Specifies an existing PEM file that contains the certificate to be renewed. The certificate's subject name and extensions are used to generate the CSR or renewed self-signed certificate.

-n or --csr-new-header
Adds the word NEW to the PEM file header and footer lines on the CSR. Some software and some CAs need this marking.

-d or --cert-validity-days *<days>*
Specifies the number of days the self-signed certificate is valid. The default is 30 days.

-D or --cert-digest *<digest>*
Specifies the digest algorithm to use when you generate a certificate-signing request or self-signed certificate. If this specification is omitted, the OpenSSL default is used.

-r or --register *<cert_file>*
Registers the zkey client with EKMF Web by generating an identity key in EKMF Web by using a certificate from the specified file. Supported certificate files formats are `.pem`, `.crt`, `.cert`, `.cer`, and `.der` (that is, either base64 or DER encoded).

To register a self-signed certificate that you are about to generate by using the `--gen-self-signed-cert` option, specify the same certificate file name here, and the generated certificate is registered immediately.

-T or --label-tags *<tag>=<value>*
Specifies the label tags for generating the identity key in EKMF Web when you register the zkey client. The label tags are a comma-separated list of tags and values, in the form *`<tag>=<value>,<tag>=<value>),...`* with tags as defined by the key template. Use the **`zkey kms info`** command to display the key templates. For registration, the template for identity keys is used.

Examples

- To connect to the EKMF Web server on `my.ekm-fweb.server`, issue:

```
# zkey kms configure -u https://my.ekmfweb.server
```

- To configure the connection to the EKMF Web server on `my.ekmfweb.server`, first pin the server's public key from the server's TSL certificate. Then verify that the hostname matches the server's **Common Name** in the certificate, issue:

```
zkey kms configure -u https://my.ekmfweb.server --tls-pin-server-pubkey --tls-verify-hostname
```

- To generate a certificate-signing request with the identity key and the specified subject name, and store it in a file named `csr.pem`, issue:

```
zkey kms configure -c csr.pem -s "CN=my.zkey.client;OU=Example;C=US"
```

- To generate a certificate-signing request with the identity key to renew the existing certificate in the file named `cert.pem` and store it in a file named `csr.pem`, issue:

```
zkey kms configure -c csr.pem -N cert.pem
```

- To generate a self-signed certificate with the identity key and the specified subject name and a validity of 50 days, and store it in a file named `cert.pem`, issue:

```
zkey kms configure -C cert.pem -s "CN=my.zkey.client;OU=Example;C=US" -d 50
```

- To generate a self-signed certificate with the identity key and the specified subject name and a certificate extension to limit the key usage, and store it in a file named `cert.pem`, issue:

```
zkey kms configure -C cert.pem -s "CN=my.zkey.client;OU=Example;C=US" -e "keyUsage=critical,
digitalSignature,keyAgreement"
```

- To register the zkey client with EKMF Web by using the certificate in the file named `cert.pem`, issue:

```
zkey kms configure -r cert.pem
```

- To register the zkey client with EKMF Web by using the certificate in the `cert.pem` file, and the label tags ENV=TEST and APP=LINUX for the identity key, issue:

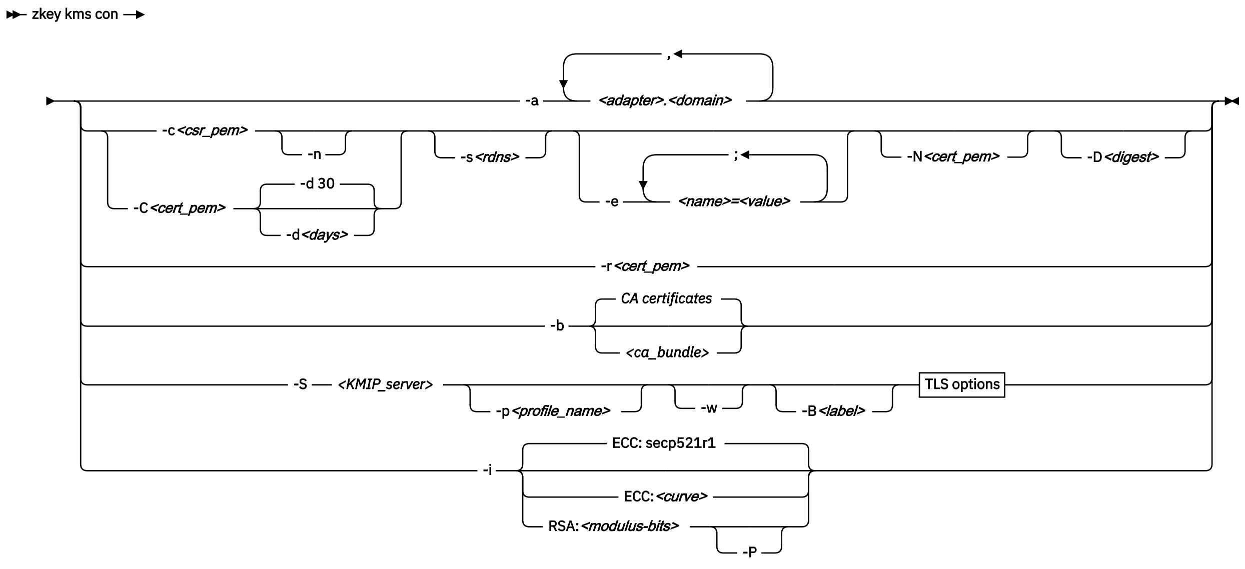
```
zkey kms configure -r cert.pem -T "ENV=TEST,APP=LINUX"
```

zkey kms configure for the KMIP plug-in

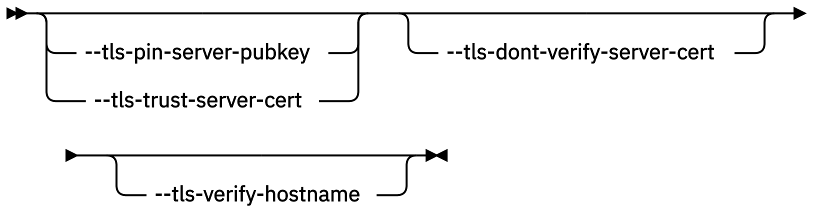
Use the **`zkey kms configure`** command to configure the KMIP plug-in.

In the following, the command is described as it applies when working with the KMIP plug-in. For the EKMF Web context, see "zkey kms configure" on page 49.

zkey kms configure syntax for KMIP



TLS options



where:

-a or --apqns <adapter.domain, adapter.domain,...>
Associates cryptographic adapters (APQNs) in either CCA or EP11 coprocessor mode with the key-management system plug-in. You can specify one or multiple APQNs in either CCA coprocessor mode or in EP11 coprocessor mode, but not both. Specify multiple APQNs as a comma-separated list. Each APQN consists of an adapter and domain number separated by a period. All APQNs that you want to set, add, or replace must be online.

To add an APQN to an existing list, prefix the APQN with a plus sign (+).

To remove an APQN from the associated APQNs, prefix the APQN with a minus sign (-).

To set or replace the APQN association, do not specify a prefix. You cannot mix plus and minus signs in one specification. You can either add, remove, or set the associations with one command.

-c or --gen-csr *<csr_pem>*

Generates a certificate signing request (CSR) with the identity key and stores it into the specified PEM file. You pass this CSR to a CA to have it issue a CA signed certificate for the KMIP plug-in. You need to register the certificate with the KMIP server before you can access it. Registering a client certificate with the KMIP server is a manual procedure, and is specific to the KMIP server used. The KMIP server accepts communication with the KMIP plug-in only after the certificate is registered. The CA-signed certificate is required for communicating with the KMIP server. Use the `--client-cert` option to specify it.

-n or --csr-new-header

Adds the word NEW to the PEM file header and footer lines on the CSR. Some software and some CAs need this marking.

-C or --gen-self-signed-cert *<csr_pem>*

Generates a self-signed certificate with the identity key and store it into the specified PEM file. You need to register the certificate with the key-management system before you can access it. You need to register the certificate with the KMIP server before you can access it. Registering a client certificate with the KMIP server is a manual procedure, and is specific to the KMIP server used. The KMIP server accepts communication with the KMIP plug-in only after the certificate is registered.

-d or --cert-validity-days *<days>*

Specifies the number of days the self-signed certificate is valid. The default is 30 days.

-s or --cert-subject *<rdns>*

Specifies the subject name for generating a CSR or self-signed certificate, in the form `<type>=<value>(;<type>=<value>)*[;]` with types recognized by OpenSSL.

-e or --cert-extensions *<exts>*

Specifies the certificate extensions for generating a CSR or self-signed certificate, in the form `<name>=[critical,]<value>(;<name>=[critical,]<value>)*[;]` with extension names and values recognized by OpenSSL.

A certificate used to authenticate at a KMIP server usually needs the TLS Web client authentication extended-key-usage certificate extension. Additionally, the Common Name field or the Subject Alternate Name extension must match the host name or IP address of the client system. If no extended-key-usage extension is specified, then a TLS Web client authentication extension (`'extendedKeyUsage=clientAuth'`) is automatically added. If no Subject Alternate Name extension is specified, then a Subject Alternate Name extension with the system's host name (subjectAltName=DNS:hostname) is automatically added.

-D or --cert-digest *<digest>*

Specifies the digest algorithm to use when you generate a certificate-signing request or self-signed certificate. If this specification is omitted, the OpenSSL default is used.

-i or --gen-identity-key ECC:*<curve>* or RSA:*<modulus-bits>*

Generates an identity key for the KMIP plug-in. The identity key is a secure ECC or RSA key. An identity key is automatically generated with the default values ECC with curve secp521r1 when a CSR or self-signed certificate is to be generated and no identity key is available.

Use this option to generate or regenerate the identity key with specific parameters. You must regenerate a client certificate with the newly generated identity key and reregister this client certificate with the KMIP server.

–P or ––cert–rsa–pss

For identity-key type RSA: Uses the RSA–PSS algorithm to sign the certificate signing request (CSR) or the self–signed certificate.

–r or ––client–cert *<cert_pem>*

Uses a CA-signed certificate to authenticate the KMIP plug-in with the KMIP server. The certificate must be registered with the KMIP server. The detailed registration steps depends on the KMIP server. Refer to your server documentation for details and use the **`zkey kms info`** command to display information about the server and its configuration.

–S or ––kmip–server *<KMIP_server>*
Specifies the hostname or IP address of the KMIP server, and an optional port number separated by a colon. The default port number is 5696. To use HTTPS transport, specify the URL, starting with `https://`, followed by the hostname or IP address of the KMIP server, an optional port number, and a URI, for example `/kmip`.

–p or ––profile *<profile_name>*
Specifies the name of the KMIP plug-in profile to use with the KMIP server connection. If no profile name is specified, the KMIP plug-in queries the KMIP server information and attempts to match a profile to the information. If no profile matches, the default profile, `default.profile` is used. Profiles are contained in the directory `/etc/zkey/kmip/profiles`. You can set the location of the profiles with the environment variable ZKEY_KMIP_PROFILES.

-N or --renew-cert *<cert_pem>*
Specifies an existing PEM file that contains the certificate to be renewed. The certificate's subject name and extensions are used to generate the CSR or renewed self-signed certificate.

-b or --tls-ca-bundle *<ca_bundle>*
Specifies the CA bundle PEM file, or the directory that contains the CA certificates that are used to verify the key-management server certificate during the TLS handshake. If the option specifies a directory path, then this directory must be prepared with OpenSSL's **`c_rehash`** utility.

The default is to use the system CA certificates.

--tls-pin-server-pubkey
For CA-signed server certificates: Pins the public key of the server. With a pinned key, it is verified that every connection uses the same key-management server certificate as the one used when the connection to the key-management server was configured.

--tls-trust-server-cert
Trusts the key-management server's certificate even if it is a self-signed certificate, or it was not verified due to other reasons. Use this option instead of the `--tls-pin-server-pubkey` option when you are using self-signed key-management server certificates.

--tls-dont-verify-server-cert
For self-signed key-management server certificates used in test environments: Bypasses the key-management server certificate verification by default. For CA-signed key-management server certificates, the default is to verify them.

This option overrides `--tls-trust-server-cert`.

--tls-verify-hostname
Verifies that the key-management server certificate's **Common Name** field or a **Subject Alternate Name** field matches the hostname that is used to connect to the key-management server.

–w or ––gen–wrapping–key
Generates a new wrapping key (key–encrypting key) based on the settings in the profile and registers it with the KMIP server. A wrapping key is automatically generated when the KMIP server connection is configured. Use this option to generate a new wrapping key later.

–B or ––label *<label>*
Specifies a human-readable identifier of the wrapping key that is stored in the Name KMIP attribute of the key. KMIP names must usually be unique within the KMIP server.

Examples

- To configure the connection to the KMIP server on `my.kmip.server`, issue:

```
zkey kms configure ––kmip–server my.kmip.server
```

- To configure the connection to the KMIP server on `my.kmip.server` using HTTPS, issue:

```
zkey kms configure ––kmip–server https://my.kmip.server/kmip
```

- To configure the connection to the KMIP server on `my.kmip.server` and use the KMIP plug-in profile ABC, issue:

```
zkey kms configure --kmip-server my.kmip.server --profile ABC
```

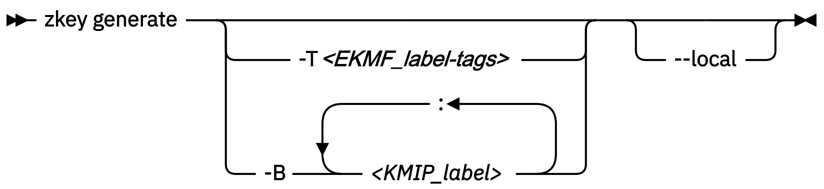
- To configure the connection to the KMIP server on `my.kmip.server` and pin the server's public key from the server's TSL certificate as well as enable verification of the hostname to match the server's Common Name in the certificate, issue:

```
zkey kms configure --kmip-server my.kmip.server --tls-pin-server-pubkey --tls-verify-hostname
```

zkey generate

Use the generate command to generate a secure AES key on a key-management system.

zkey generate syntax



where:

-T or --label-tags *<label-tags>*

EKMF Web only. Specifies the label tags for generating a secure key in EKMF Web, in the form *<tag>=<value>(,<tag>=<value>)*[,]* with tags as defined by the key template. Use the **`zkey kms info`** command to find out which label tags the configured key template uses.

–B or ––label <labels]>

KMIP only. Specifies an optional human-readable identifier of the key or keys stored in the `Name` KMIP attribute of the key. KMIP names must usually be unique within the KMIP server. For XTS type keys, two different labels must be specified, separated by a colon.

--local

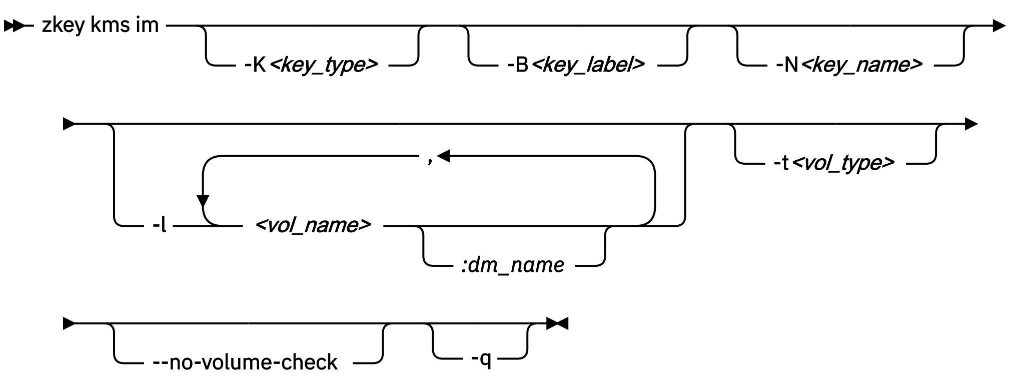
Generates keys in the local secure key repository. These keys cannot be used in a key-management system.

For a complete list of options for the **`zkey generate`** command, see the zkey man page.

zkey kms import

Use the **`zkey kms import`** command to import secure keys from a key-management system into the secure key repository on your Linux instance. The default is to import all eligible keys.

zkey kms import syntax



Where:

-K or --key-type *<key_type>*
: **KMIP only.** Specifies the type of the key to import. Possible values are:

- CCA-AESDATA
- CCA-AESCIPHER
- EP11-AES

The key type must match the type of APQNs associated with the KMIP plug-in.

When cryptographic adapters in CCA coprocessor mode are associated with the KMIP plugin, secure keys of type CCA-AESDATA, and CCA-AESCIPHER are supported. The default type is CCA-AESDATA.

When cryptographic adapters in EP11 coprocessor mode are associated with the KMIP plugin, secure keys of type EP11-AES are supported. The default type is EP11-AES.

-B or --label *<key_label>*
Specifies the label of the secure key in the KMS. Use wildcards to select multiple secure keys. If you use wildcards, enclose the value in quotation marks.

-N or --name *<key_name>*
Specifies the key name of the secure key.

-l or --volumes *<vol_name>*
You can associate volumes with a key. Each volume association specifies the name of the block device, for example `/dev/mapper/disk1`, and the device mapper name separated by a colon.

Separate multiple volume associations with a comma, for example:

```
# zkey kms import -l /dev/mapper/disk1:enc-disk1,/dev/mapper/disk2:enc-disk2
```

-t or --volume-type *<vol_type>*
Specifies the volume type of the associated volumes used with dm-crypt. Possible values are PLAIN or LUKS2

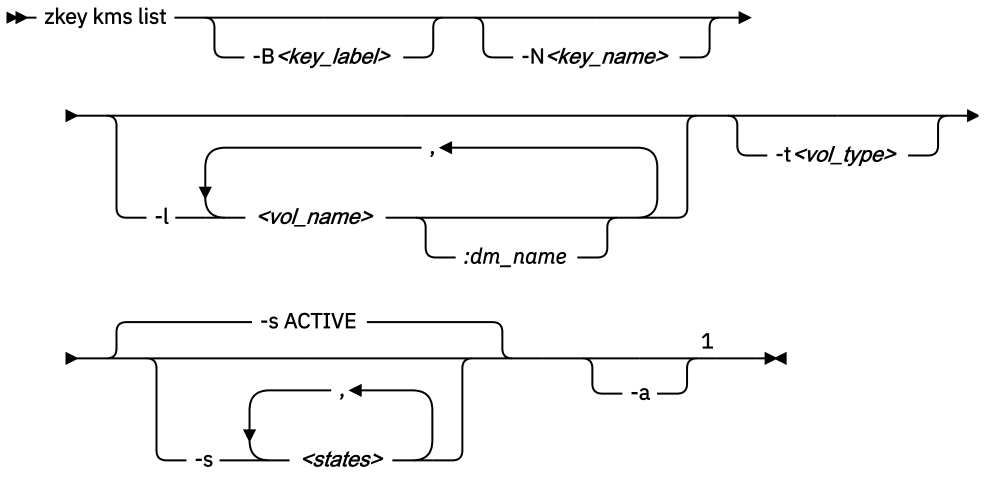
--no-volume-check
Omits the volume check, and imports the keys even if the associated volumes do not exist.

-q or --batch-mode
Suppresses prompts for names of existing keys. Keys with an existing name are skipped.

zkey kms list

Use the **`zkey kms list`** command to list encryption keys.

zkey kms list syntax



Notes:

[1] The default lists keys that the current secure key repository can use.

where:

-B or --label *<key_label>*
Specifies the label of the secure key in the KMS. Use wildcards to list multiple secure keys. If you use wildcards, enclose the value in quotation marks.

-N or --name *<key_name>*
Specifies the key name of the secure key.

-l or --volumes *<vol_name>*
You can filter the list by the volumes that are associated with a key. Each volume association specifies the name of the block device, for example `/dev/mapper/disk1`, and the device mapper name separated by a colon.

Separate multiple volume associations with a comma, for example:

```
# zkey kms list -l /dev/mapper/disk1:enc-disk1,/dev/mapper/disk2:enc-disk2
```

-t or --volume-type *<vol_type>*
Filters the list by volume type of the associated volumes used with dm-crypt. Possible values are PLAIN or LUKS2

EKMF Web only:

-s or --states *<states>*
Filters the list by key states. Separate multiple states with a comma. Possible states are PREACTIVATION, ACTIVE, DEACTIVATED, COMPROMISED, DESTROYED, and DESTROYED-COMPROMISED.

The default is to list the ACTIVE keys.

-a or --all
Lists all keys that can be used for volume encryption.

By default, keys that can be exported to this secure key repository are listed.

Examples

- To list secure keys managed by the key-management system, regardless of whether the zkey client is allowed to use it:

```
# zkey kms list --all
```

- Using an EKMF Web plug-in, to list secure keys managed by EKMF Web, which this zkey client is allowed to use and are in state ACTIVE:

```
# zkey kms list
```

- Using an EKMF Web plug-in, to list secure keys managed by EKMF Web, which this zkey client is allowed to use, and are in ACTIVE or DEACTIVATED state:

```
# zkey kms list --states ACTIVE,DEACTIVATED
```

zkey kms reencipher

Use the **`zkey kms reencipher`** command to reencipher a secure key in a key-management system.

There can be plug-in specific options. Use **`zkey kms reencipher --help`** to see which plug-in specific options a plug-in provides or requires.

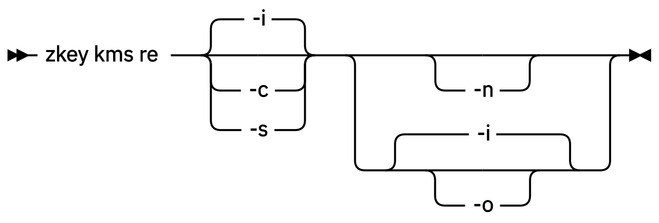
For KMIP: Use the **`zkey kms reencipher`** command to reencipher the identity key and the wrapping key with a new master key. These keys must be must be reenciphered when the master keys of the associated AP queues change:

- For cryptographic adapters in CCA coprocessor mode, this is the APKA master key.
- For cryptographic adapters in EP11 coprocessor mode, this is the EP11 master key.

Note: The **`zkey kms reencipher`** command does not reencipher secure keys that were generated by, or have been imported from, the KMIP server, and are now stored in the secure key repository. Use the regular **`zkey reencipher`** command to reencipher those secure keys.

For EKMF Web: The **`zkey kms reencipher`** command re-enciphers the secure-identity key of the EKMF Web plug-in with a new master key. The secure-identity key must be re-enciphered when the APKA master key of the CCA cryptographic adapter changes.

zkey kms reencipher syntax



where:

-n or --to-new
Reenciphers a secure key with the master key in the NEW register.

-o or --from-old
Reenciphers a secure key that is currently enciphered with the master key in the OLD register with the master key in the CURRENT register.

If both `-n` and `-o` are specified, a secure key that is currently enciphered with the master key in the OLD register is reenciphered with the master key in the NEW register.

-i or --in-place
Forces an in-place re-enciphering. This is the default for `--from-old`.

-s or --staged

Stores the key in a file, `<key-name>.renc`, in the secure key repository. The key in `<keyname>.skey` is still valid. Once a new master key has been set, you must rerun the reencipher command with option --complete. This copies the file <key-name>.renc to <key-name>.skey and thus completes the staged re-enciphering. Re-enciphering from CURRENT to NEW is by default done in staged mode.

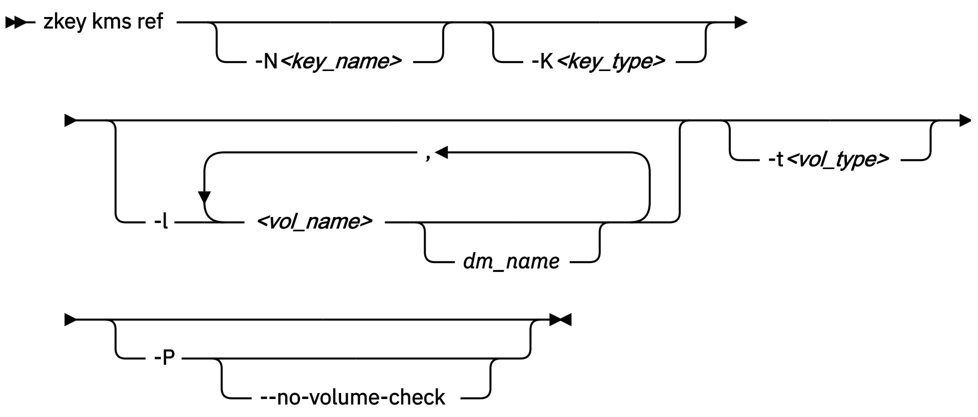
-c or --complete

Completes a staged re-enciphering.

zkey kms refresh

Use the **`zkey kms refresh`** command to reimport all, or a selection of encryption keys, or refresh key properties.

zkey kms refresh syntax



where:

-N or --name *<key_name>*

Specifies the key name of the secure key. Use wildcards to refresh multiple secure keys. If you use wildcards, enclose the value in quotation marks.

-K or --key-type *<key_type>*

Refreshes keys with the specified key type. Possible values are CCA-AESDATA, CCA-AESCIPHER, or EP11-AES.

-l or --volumes *<vol_name>*

You can filter the list of keys to refresh by the volumes that are associated with a key. Use wildcards to refresh keys for multiple volumes. If you use wildcards, enclose the value in quotation marks.

-t or --volume-type *<vol_type>*

Refreshes keys with the specified volume type. Possible values are PLAIN or LUKS2.

-P or --refresh-properties

Updates the associated information, such as the textual description, associated volumes, volume type, and sector size, with the information stored in the key management system.

--no-volume-check

Omits checking if the volumes that are associated with the secure keys to be refreshed are available, or are already associated with other secure keys in the repository. This option has an effect only if specified together with the `--refresh-properties` option.

Examples

- To refresh secure keys from EKMF Web whose name starts with "sec".

```
# zkey kms ref -N "sec*"
```

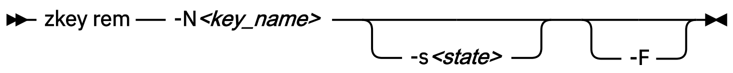
- To refresh the secure key with the name "seckey" from EKMF Web including its properties:

```
# zkey kms ref -N seckey -P
```

zkey remove

Use the **`zkey remove`** command to remove encryption keys from the local repository, and optionally set a new key state for the key in the key-management system.

zkey remove syntax



where:

-N or --name *<key_name>*
Specifies the key name of the secure key.

-s, --state *<state>*
Specifies the new state for the key on the key-management system server after removing the secure key from the local secure key repository. Possible states are DEACTIVATED, COMPROMISED, DESTROYED, and DESTROYED-COMPROMISED.

The default is to remove the key from the local secure key repository, but leave the state in the key-management system unchanged.

-F or --force
Forces the removal of the key

Example

- To remove the secure key named `seckey` from the repository and set the state of the key to DEACTIVATED in the key-management system:

```
# zkey remove --name seckey --state DEACTIVATED
```

Accessibility

Accessibility features help users who have a disability, such as restricted mobility or limited vision, to use information technology products successfully.

Documentation accessibility

The Linux on IBM Z and LinuxONE publications are in Adobe Portable Document Format (PDF) and should be compliant with accessibility standards. If you experience difficulties when you use the PDF file and want to request a Web-based format for this publication send an email to eservdoc@de.ibm.com or write to:

IBM Deutschland Research & Development GmbH
Information Development
Department 3282
Schoenaicher Strasse 220
71032 Boeblingen
Germany

In the request, be sure to include the publication number and title.

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

IBM and accessibility

See the IBM Human Ability and Accessibility Center for more information about the commitment that IBM has to accessibility at

```
www.ibm.com/able
```

Notices

Trademarks

Index

SC34-7740-01