Linux on IBM Z and IBM LinuxONE

Introducing IBM Secure Execution for Linux April 2024 update

SC34-7721-04

Note

Before using this document, be sure to read the information in “Notices” on page 81.

This edition applies to the IBM® Secure Execution for Linux® update in 2024 and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

Summary of changes

This revision includes maintenance and editorial changes. Technical changes or additions to the text are indicated by a vertical line to the left of the change.

What's new in the April 2024 edition

This edition, SC34-7721-04, contains the following changes compared to the previous edition.

New information

- The Resource Link URLs for host key documents have been updated, see Appendix C, "Obtaining a host key document from Resource Link," on page 77 and Appendix D, "Verifying the host key document," on page 79.
- The speed of booting and shutdown of KVM guests running in IBM Secure Execution mode is improved, see "Improved reboot and shutdown time" on page 3.
- A Secure Execution guest can now use Crypto Express adapters in Enterprise PKCS #11 coprocessor mode or accelerator mode, see Chapter 9, "Crypto Express adapters for secure-execution guests," on page 37.
- To protect EP11 secure keys, these keys can be associated with an association secret that is maintained by the ultravisor on behalf of the secure guest. To facilitate this protection, you can now submit secrets to the ultravisor, see "Submitting a secret to the ultravisor " on page 20 and "pvsecret - Create requests, add and list secrets, and lock the store of secrets" on page 70.
- A new command, **pvapconfig** implements cryptographic device configurations, see "pvapconfig - Implement an AP queue configuration " on page 63.

Changed information

- None

What's new in the November 2022 edition

This update, SC34-7721-03, contains the following changes compared to SC34-7721-02.

New information

- You can now ensure the identity, security, and integrity of a secure guest with attestation, see "Attestation on IBM Secure Execution" on page 2 and Chapter 7, "Attesting a KVM guest," on page 31.
- Update of the Linux distributions that include IBM Secure Execution, see "Required software" on page 14.
- The Workload owner tasks are updated with a new task in support of attestation, see "Extracting an IBM Secure Execution header" on page 25.
- In support of attestation, two new commands are available, see "pvattest - Create, perform, and verify attestation requests" on page 66 and "pvextract-hdr - Extract an IBM Secure Execution header" on page 69.

Changed information

- The host key document is now verified automatically by the commands **genprotimg** and **pvattest**, you no longer need a manual verification when "Securing the guest" on page 18 or Chapter 7, "Attesting a KVM guest," on page 31.
- The manual verification procedure has been moved to an appendix, see Appendix D, "Verifying the host key document," on page 79.

What's New in the September 2021 edition

This update, SC34-7721-02, contains the following changes compared to SC34-7721-01.

New information

- Requirements for guest memory are added, see “Guest memory requirements” on page 13.
- Linux distributions that include IBM Secure Execution are now listed, see “Required software” on page 14.
- New sysfs attributes indicate whether a Linux instance detects its environment as consistent with that of a secure guest or host, see “Enabling the KVM host for IBM Secure Execution” on page 50.
- If your environment supports it, you can now use a new element in the domain-XML to simplify secure guest setup, see “Starting the secure virtual server” on page 51.
- The **genprotimg** command now has options to manage the Permit CPACF Key Management Operations (PCKMO) support, see “genprotimg - Generate an IBM Secure Execution image” on page 60.

About this document

Learn about IBM Secure Execution for Linux, how you can benefit from it, how to set up a secure workload, and how the workload runs securely in a public, private, or hybrid cloud.

This publication describes IBM Secure Execution for Linux, which was introduced with IBM z15® and LinuxONE III. It describes how you can create encrypted Linux images that can run on a public, private or hybrid cloud with their in-use memory protected. The publication describes how to set up the KVM host, the secure guests, and how the security works.

For details about IBM tested Linux environments, see www.ibm.com/systems/z/os/linux/resources/testedplatforms.html.

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

```
https://www.ibm.com/docs/en/linux-on-systems?topic=overview-introducing-secure-execution-linux
```

Distribution independence

The information in this publication is Linux distribution independent.

Who should read this publication

For workload owners, this publication explains how IBM Secure Execution for Linux works, and what you must do to safely deploy your workload.

For cloud providers, this publication gives insights into how IBM Secure Execution for Linux works on z15®, and how the KVM host must be set up for workloads to benefit from it.

This publication assumes:

- Linux on Z or LinuxONE administrator skills.
- Familiarity with security concepts.

Terminology

IBM Secure Execution for Linux uses the terminology listed here.

boot image
A disk image that has been prepared as a boot device. It contains all data that is required to start a Linux instance. This data includes a kernel image, an initial RAM disk, kernel parameters, and a boot loader.

host key document
Contains the public host key in an X.509 certificate format, signed with an IBM key. A host key document is like a certificate with IBM as the trusted third party.

HSM master key
An HSM master key encrypts all other keys on that HSM. These are sometimes also called HSM wrapping keys or EP11 wrapping keys.

KVM virtual server, virtual server
Virtualized IBM Z resources that comprise processor, memory, and I/O capabilities as provided and managed by KVM. A virtual server can include an operating system.

KVM guest, guest, guest operating system
An operating system of a virtual server.

KVM host, host, hypervisor
The Linux instance that runs the KVM virtual servers and manages their resources.

master key verification pattern (MKVP)
An MKVP identifies the master key. These patterns are also sometimes called wrapping key verification patterns.

protected virtualization
An alternative name for IBM Secure Execution that still exists in some program code and, for example, in the name of the **genprotimg** command.

Secure guest ownership
The secure guest owner refers to the entity that possesses the secrets necessary for accessing and recognizing a secure guest, such as root passwords, TLS/SSH keys, and encryption keys. While the creator typically owns guests they've created, in cases where vendors sell pre-packaged secure guest images, ownership must transfer to the customer early in the guest's lifecycle. This transfer, known as "personalization", involves replacing vendor-installed secrets with those belonging to the guest owner.

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
- *Secure Boot for Linux on IBM Z and IBM LinuxONE*, SC34-7755
- *openCryptoki - An Open Source Implementation of PKCS #11*, SC34-7730
- *OpenSSL support for Linux on IBM Z and LinuxONE*, SC34-7732
- *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

Dynamic Partition Manager publications

Dynamic Partition Manager (DPM) publications are available from the following web pages:

- *IBM Dynamic Partition Manager (DPM) Guide*, SB10-7176-02 available from: https://www.ibm.com/docs/en/systems-hardware/zsystems/3932-A02?topic=library-dynamic-partition-manager-dpm-guide

Chapter 1. What is IBM Secure Execution?

IBM Secure Execution for Linux is a z/Architecture® security technology that is introduced with IBM z15 and LinuxONE III. It protects data of workloads that run in a KVM guest from being inspected or modified by the server environment.

In particular, no hardware administrator, no KVM code, and no KVM administrator can access the data in a guest that was started as an IBM Secure Execution guest.

Thus, IBM Secure Execution for Linux is a continuation and expansion of well-known security features of IBM Z and LinuxONE. It supplements pervasive encryption, which protects data at-rest and data in-flight, to also protect data in-use. With IBM Secure Execution for Linux, it is possible to securely deploy workloads in the cloud. The data of the workload can be protected everywhere:

- In flight with secure network protocols like TLS, SSH or IPsec
- At rest with volume encryption like dm-crypt or file system encryption like with IBM Spectrum® Scale
- In use in the memory of a running guest with IBM Secure Execution protection

Note: Since IBM Z 16 and LinuxONE 4, all memory is encrypted. This memory encryption is transparent to all firmware and software. It is intended to protect the IBM Z and IBM LinuxONE memory against physical attacks. This memory encryption is independent of any encryption performed in the context of IBM Secure Execution.

When a KVM guest runs in a cloud, be it in-house or third-party, security risks to the workload include:

- Intruders who might gain root privileges of the hypervisor due to some error in the security administration.
- Malicious hypervisor code that might be introduced by exploits, including zero-day exploits, or intruders.
- Malicious virtual machines that, hypothetically, can escape the control of the hypervisor, and gain hypervisor privileges.
- A malicious hardware operator who inspects the memory of an LPAR.

Intruders, malicious hypervisors, or malicious virtual machines are risks for both the cloud provider and the cloud customer, see Figure 1 on page 2.

To provide a secure hosting environment, a cloud provider might log every key stroke and conduct expensive audits to log any management action and deter any malicious actor.

With the introduction of pervasive encryption, all your data at rest could be encrypted with no application changes and at reasonable CPU cost.

With IBM Secure Execution, data is protected during processing. As a workload owner, your data in your KVM guest that is deployed in a cloud, which runs on IBM Z servers with IBM Secure Execution, are as safe as if you ran it in your own data center. In fact, it is safer. It is also protected from insider attacks. Only the workload owner can access the data.

Benefits of IBM Secure Execution

IBM Secure Execution comes with a number of benefits including technology-enforced security.

IBM Secure Execution provides the following benefits:

- Instead of relying on deterrence by using extensive audit tracks, IBM Secure Execution provides technology-enforced security rather than process or audit-based security.
- As a cloud provider that uses IBM Secure Execution, you can attract sensitive workloads that, formerly, were restricted to the workload owner's system.
- As a secure workload owner, you know that your workload is run in a secure manner, even outside your data center.

- As a secure workload owner, you can choose where to run your workload, independently of the security level required.

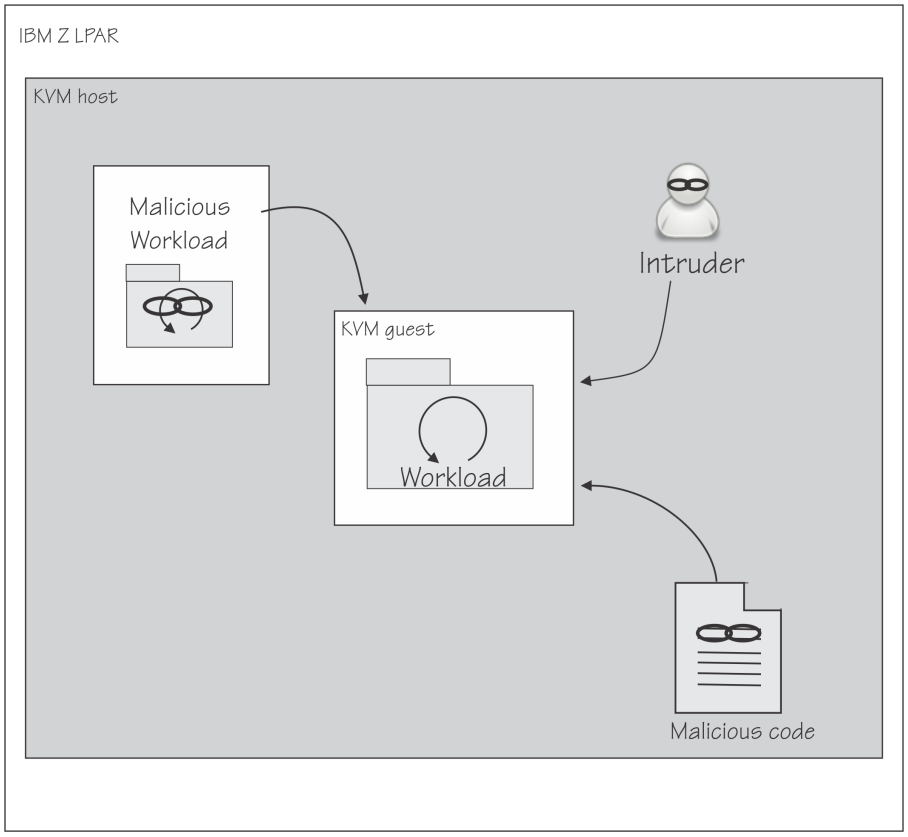


Figure 1. IBM Secure Execution protects workloads on clouds from intruders, malicious workloads, and malicious code

Attestation on IBM Secure Execution

Reasons for IBM Secure Execution attestation include auditing, image personalization, and aligning with other confidential computing architectures.

With cybersecurity threats developing and calling for mitigation, attestation is being integrated into workflows for cloud-based workloads. IBM Secure Execution as a superior security architecture provides an attestation function.

The following examples illustrate possible uses of attestation.

Auditing
Your organization might mandate that an attestation on cybersecurity be included in each department's annual report. That is, annually, a report must be created that shows that cybersecurity measures are in place. This report includes showing that all workloads that run at a cloud provider are safe.

Personalization
Assume that a KVM guest in secure execution mode runs a generic workload, for example Soda Company Recipe Store. This workload can be bought and used by different soda companies. These companies would want to personalize the KVM guest with individual secrets, such as replacing SSL or TLS keys. But before that they want to verify the integrity of the base image.

Unlocking data
A company provides data in the form of a file system encrypted with LUKS. A KVM guest running in secure execution mode is to process this data. An attester performs the attestation, and only sends the LUKS key to this guest after verifying its integrity. This procedure might be mandated by an external workflow.

For a description of how to attest a KVM guest, see Chapter 7, “Attesting a KVM guest,” on page 31.

Crypto Express adapters on secure guests

Use Crypto Express adapters on your secure guest for cryptographic operations. A secure guest can use domains of a Crypto Express adapter that are configured as accelerators or an EP11 coprocessor (hardware security modules).

On IBM Z and IBM LinuxONE, Crypto Express adapters with their virtual hardware security modules (HSM) are the natural choice for hardware-powered and -secured encryption.

Using a virtual HSM for a KVM guest that runs in IBM Secure Execution mode ensures that:

- Your guest can distinguish its HSM from a tampered substitute that holds a master key known to an attacker.
- You prevent a compromised hypervisor or peer guest from extracting sensitive data from your exchanges with your Crypto Express adapter.
- You prevent a malicious peer guest from using your HSM to decrypt data with a stolen secure key.

Secure Execution for Linux uses a special secret to associate a secure guest to an HSM. The untrusted provider of the host environment configures the HSM for the KVM guest, but cannot use it once it is associated with the secure guest.

For details of how to set up a virtual HSM for your guest, see Chapter 9, “Crypto Express adapters for secure-execution guests,” on page 37.

Improved reboot and shutdown time

Reboot time is improved for KVM guests running in IBM Secure Execution mode.

A feature of QEMU that allows guests to reboot quickly is available by default.

On the KVM host, the file `/sys/module/kvm/parameters/async_destroy` indicates whether the feature is active.

To enable shutdown improvement, add the async-teardown element to the guest's XML.

Support for async-teardown is included in libvirt if the element `async-teardown` is listed in domcapabilities. Confirm that this setting is available in your environment by checking that the `async-teardown` element attribute `supported` has the value `yes`. For example:

```
# virsh domcapabilities | grep async-tear
<async-teardown supported='yes'/>
```

You can configure your guest to use this feature by including the element `async-teardown` in the guest's domain XML features. An example domain XML, here for a SUSE Linux Enterprise Server instance, could look like:

```
<domain type='kvm'>
   <name>fast-shutdown</name>
   ...
   <memory unit='KiB'>1150976</memory>
   <vcpu placement='static'>2</vcpu>
   <os>
      <type arch='s390x' machine='s390-ccw-virtio'>hvm</type>
   </os>
   <features>
      <async-teardown enabled='yes'/>
   </features>
   <devices>
      <emulator>/usr/bin/qemu-system-s390x</emulator>
   ...
   </devices>
</domain>
```

For details about setting up features for and operating Linux on KVM instances, see *KVM Virtual Server Management*, SC34-2752.

Chapter 2. IBM Secure Execution components

To make your workload safe in the cloud, IBM Secure Execution for Linux provides technology-based mitigation for several security threats.

With IBM Secure Execution, your workload is run in a specially protected virtual server. When you start a guest in IBM Secure Execution mode, the following aspects are protected against being observed or modified by the hosting environment:

- The boot image
- The guest memory
- The guest state

Across its entire lifecycle, such a guest has its confidentiality and integrity protected, from the moment the image is built, through the boot process and the running of the virtual server, until its termination.

IBM Secure Execution is based on hardware and firmware features that became available with the IBM z15:

- Built-in private host key, see also private host-key handling
- Hardware memory protection
- Ultravisor

The ultravisor is trusted firmware that uses memory-protection hardware to enforce memory protection. The owner of a guest with IBM Secure Execution can securely pass secret information to the ultravisor by using the public host key, which is embedded in the host key document. To process the secret information, the ultravisor uses the matching private host key. The private host key is specific to an IBM Z server and is hardware protected.

A guest in IBM Secure Execution mode runs only on one or more specific, trusted servers that are provided as a cloud environment by your cloud provider. Only the workload creator can grant access to the data on it.

Private host-key handling: With IBM z15, IBM kept an HSM-protected copy of the private host key. As of IBM z16, IBM no longer keeps a copy. The only copy of the private host key is in the host-key bundle delivered with the Support Element (SE).

For IBM z16, use the Backup Critical Console Data task on the HMC to back up the host-key bundle that is delivered with your SE disk.

Boot image protection

The boot image of a secure virtual server must be prepared for the guest to run under the control of the ultravisor.

The preparation includes encrypting the boot image and computing a cryptographic hash of the image, as well as creating an IBM Secure Execution header (SE header) for this image. The header contains the image encryption keys and the hashes. The SE header itself is integrity protected, with its critical parts encrypted.

A host key document is specific to a host system on which a secure virtual server can run. A host can run the Linux instance if it can decrypt the customer root key (CRK) in the SE header. To ensure that only a particular host can run an instance, the CRK in the SE header is encrypted with that host's public key. To enable multiple hosts to run an instance, multiple encrypted copies of the CRK can be included in the SE header. Each copy is encrypted with the public key of a host where the instance can run.

Given that the boot image is encrypted, it can contain sensitive data that cannot be observed or modified by the components or operators that start the image. Typical examples of sensitive data that owners of secure virtual servers would want to place in a boot image include LUKS passphrases, root password hashes, or SSH certificates.

When the image boots, the ultravisor is given both the SE header and the encrypted image. Based on the information in the SE header the ultravisor verifies the integrity of the SE header and the image. It decrypts the image and starts the image only if all integrity checks succeed.

Memory protection

In a regular KVM setup, the KVM hypervisor can access the memory of the virtual servers. IBM Secure Execution isolates the guest memory from the KVM hypervisor by running secure virtual servers in secure memory.

Secure memory cannot be accessed from outside the secure virtual server or trusted firmware. In particular, secure memory cannot be accessed from the KVM hypervisor or any memory inspection function on the Support Element or the HMC. Hypervisor requests to access secure memory are rejected and redirected to the Ultravisor. Thus the workload is protected against data manipulation and extraction while it is running and at rest.

Once the boot image is decrypted by the ultravisor, it is stored in secure memory and all memory that is used by the secure virtual server continues to be secure.

State protection

Virtual servers need resources, such as memory and virtual CPUs to be functional. Such resources and their states are described by control blocks and comprise a large portion of the state of a virtual server. In addition, the state of a guest includes CPU registers, parts of cryptographic keys, and the program status word (PSW). Anyone who can access the state can learn something about the workload. Therefore, the ultravisor also protects all state information of a secure virtual server.

Ultravisor

To protect against control block access, IBM Secure Execution introduces a new entity, the *ultravisor*, that controls execution instead of KVM. The ultravisor decrypts the workload, secures its memory, runs, and manages it securely.

The task of the ultravisor is to load the image of a secure guest, which includes its decryption and integrity verification. It turns all memory to be used by the secure guest into secure memory and protects the state of a secure virtual server.

The ultravisor takes over all sensitive work from KVM. KVM works through a special instruction with the ultravisor. The ultravisor performs a security check on KVM's requests, and runs them, while it gives KVM only necessary information.

IBM Secure Execution protects against manipulation of the workload and tampering with memory pages.

Figure 2 on page 7 shows three KVM guests. One guest operates in regular mode and the other two in IBM Secure Execution mode. The Ultravisor controls the IBM Secure Execution guests.

The hatched squares in the figure symbolize secure pages. The white squares represent memory pages that are voluntarily shared between the guests and the hypervisor to allow the hypervisor to perform I/O for the guest. The guest needs access to pages that are used to handle I/O. These pages are called bounce buffers.

The checkered squares represent memory pages encrypted by the ultravisor and are accessible to the hypervisor for page swapping operations. These encrypted pages are *not accessible to the secure guest*. If the hypervisor needs to swap out a page, the ultravisor stores a hash of the page and encrypts it before granting the hypervisor access to the page. When the hypervisor returns the page to the guest the ultravisor checks its contents against the stored hash.

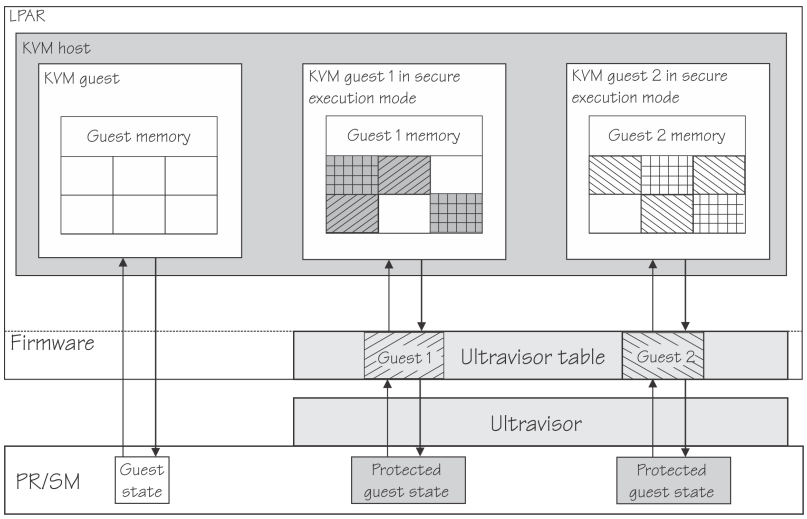


Figure 2. IBM Secure Execution protects guest memory and state

The hosting LPAR donates some memory to the ultravisor, which uses it to build a table to hold the security context data of the guest.

The secure memory that is used by a secure virtual server is labeled with the ID of the virtual server. Each virtual server can use only secure memory that is labeled with its own ID. Access to secure memory that is labeled with another ID is prevented by the IBM Z memory management hardware and firmware. This process strengthens the isolation of different guests that run in the same hypervisor. Such guest separation might be useful to cloud providers who, for example, have a legal requirement to keep workloads from different customers completely separated.

Summary

IBM Secure Execution protects the memory and state of each secure virtual server during its lifecycle.

Boot images for secure virtual servers must be encrypted and integrity protected. You use a command that automates the protection procedure. The boot image protection does not require any modification of existing applications.

It is possible to place sensitive data in the boot image that can never be observed or tampered with from the hosting environment of the secure virtual server.

A workload can only run on the hosts for which it was prepared.

Chapter 3. Securing a workload in the cloud

IBM Secure Execution encrypts the kernel image, the initial RAM file system, and the kernel parameter line. You are responsible for the application data encryption and its associated key management.

IBM Secure Execution keys

Every IBM z15 or LinuxONE III server is equipped with a private host key that is specific to that server. The key is protected by hardware and firmware. The cloud provider cannot access or manipulate the private host key. Cloud providers who run their cloud on z15 or LinuxONE III obtain a host key document from IBM. The host key document contains the public key associated with the private host key of that server. The cloud providers can distribute a host key document to cloud customers who want to run their workload in a z15 or LinuxONE III based cloud environment.

As a workload owner, you encrypt files that are necessary for booting by using the host key document of the cloud provider. The ultravisor uses the private host key, that is embedded in the hardware, to decrypt these files for the guest to boot in the cloud environment. These concepts are illustrated in Figure 3 on page 9

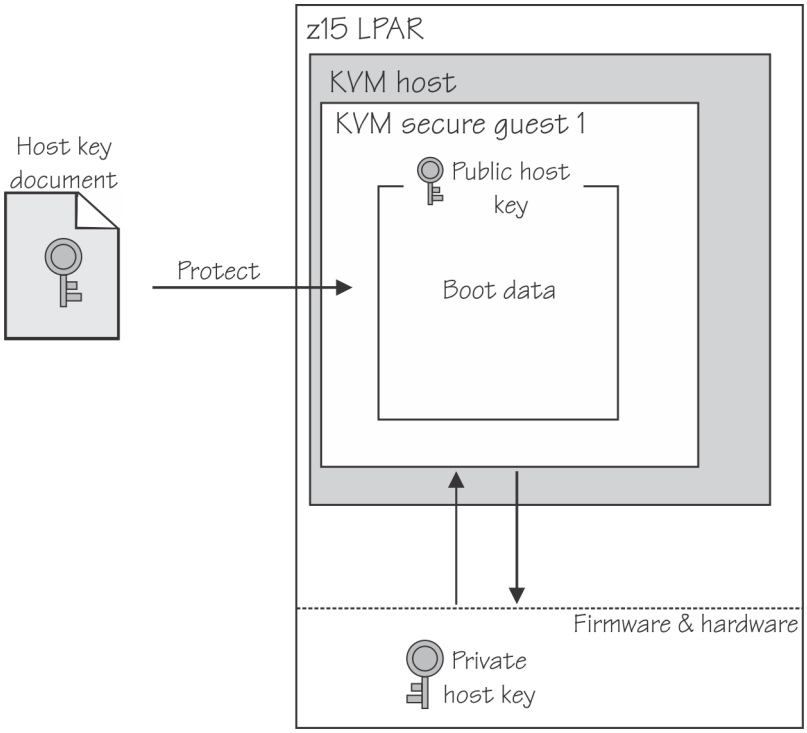


Figure 3. Encrypt files that are necessary for booting by using the host key document

Your application data is already encrypted, for example, with dm-crypt that uses LUKS volumes. This publication assumes that the data is encrypted with a symmetric key for faster encryption and decryption. The boot image that accesses the data needs access to the LUKS passphrase to use the data. Hence, you must copy the passphrase to the boot image.

IBM Secure Execution uses a cascade of encryption keys to ensure the security of the boot image. You only need to encrypt the data and use a command to secure the boot image. The cascade is shown in Figure 5 on page 11, and explained in the following.

Verify the host key document

Verifying the host key document is essential to ensuring the chain of trust for your workload, as shown in Figure 4 on page 10. The verification has two steps: first the host-key-signing-key certificate certificate

must be verified using the CA certificate. If that was successful, you can verify the host key document with the public key from the successfully verified host-key-signing-key certificate certificate. Then you can be sure that the host key document is valid.

These steps are done automatically for you when using the **genprotimg** and **pvattest** commands.

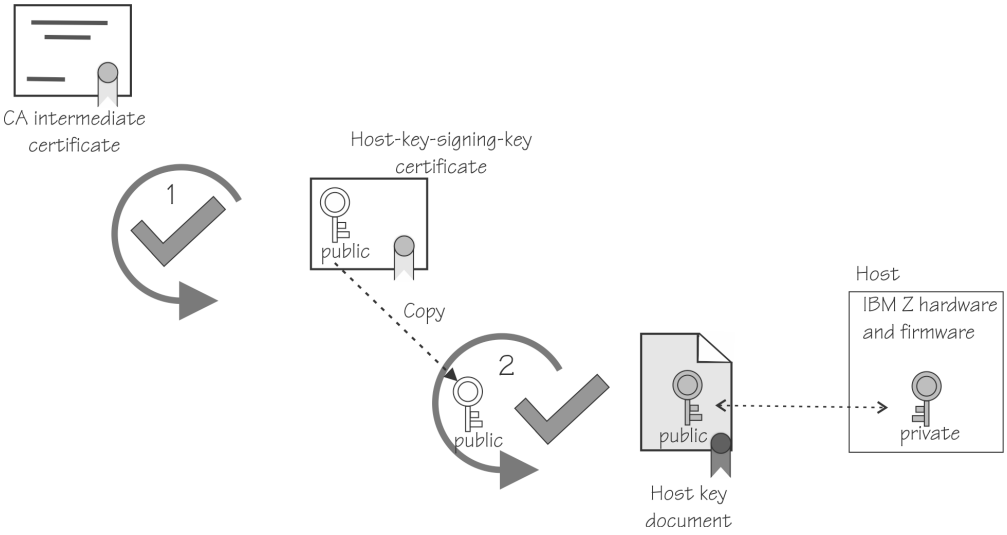


Figure 4. Chain of trust to verify the host key document

Encrypt the boot image

The boot image consists of the kernel, the kernel parameters and the initial RAM file system. The initial RAM file system contains the initial secrets needed to access the system, such as credentials needed for disk or file system encryption, password hashes, or SSH certificates. It also contains secrets that allows the system to identify itself, such as private SSH keys.

The secure execution technology uses the efficient crypto acceleration of CPACF to encrypt the entire boot image. Because it is encrypted, a boot image can hold secrets of any size at any location. An encrypted boot image also hides the nature of the workload, thus minimizing the attack surface for crypto analysis.

The symmetric key that you used to encrypt your data volume would now be in the clear on the boot image. The boot image must also be encrypted with another symmetric key, the image encryption key. To run the image, this key must be included in the IBM Secure Execution header.

Because the initial RAM file system is protected, you can in general keep secrets there, such as workload-specific keys, or network traffic protection keys.

When you secure the image header, the command you use for securing, **genprotimg**, creates an image encryption key for you and copies it to the correct location in the IBM Secure Execution header. Optionally, you can provide your own key as an argument to the command.

Encrypt the IBM Secure Execution header

The image encryption key, in turn, is now in the clear on the IBM Secure Execution header. To protect it, use the host key document to encrypt the IBM Secure Execution header. The only environment that can now decrypt and run the IBM Secure Execution header is the trusted hardware.

Figure 5 on page 11 shows a simplified view of the keys that are involved in all stages of securing the workload. The key used to encrypt the boot image can be automatically created and handled by the securing tool.

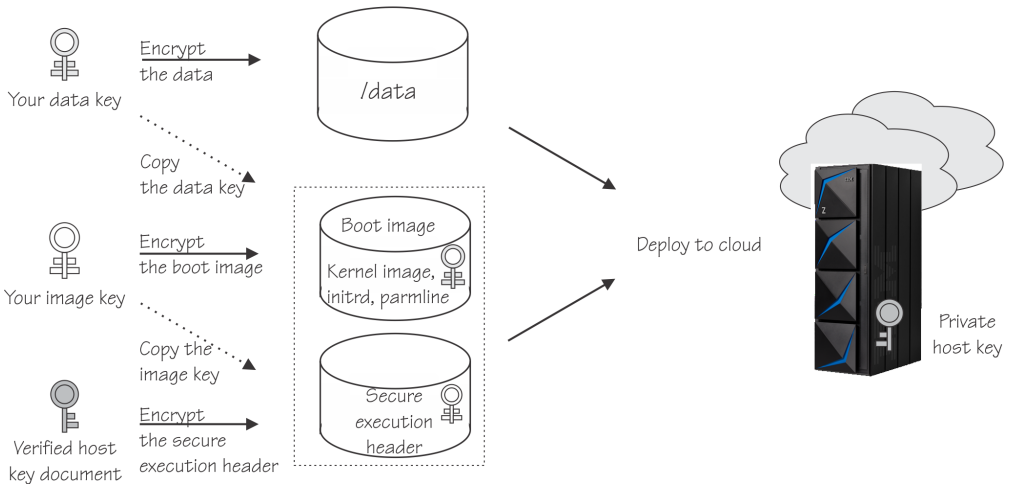


Figure 5. Conceptual key cascade for IBM Secure Execution

Chapter 4. What you should know

Before you start working with IBM Secure Execution, find out about prerequisites and restrictions.

IBM Secure Execution for Linux requires an IBM z15 or LinuxONE III or later models with the feature installed.

As the host is not allowed to access guest memory and state, certain KVM features are not supported, including:

- Live migration. Offline migration is possible, if the guest is built for more than one host. For more information about how to build for multiple hosts, see “genprotimg - Generate an IBM Secure Execution image” on page 60
- Save to and restore from disk.
- Hypervisor-initiated memory dump.
- Pass-through of host devices, for example PCI and CCW.
- Using huge memory pages on the host for backing guest memory.
- Memory ballooning through a virtio-balloon device.

In contrast to regular KVM guests, guests running in Secure Execution mode are limited to 247 virtual CPUs.

Guest memory requirements

KVM guests in IBM Secure Execution mode require more memory than the same guests in regular mode.

The following aspects increase the memory requirement of a guest:

- The extra memory needed by a bounce buffer. Add the swiotlb value multiplied by 2 KB.
- Disk encryption, if introduced with IBM Secure Execution.
- Setting up kdump if not already in place. Double the resulting figure of the preceding aspects.

Guests with a too narrowly computed memory assignment might not boot.

Example

Consider a guest that is configured to use kdump, but no disk encryption. The swiotlb value is set to 262144. Assuming swiotlb memory blocks of 2 KB, this results in 512 MB extra memory.

For a disk that is encrypted with LUKS2, use the **`cryptsetup luksDump`** *<LUKS_volume>* command to display the memory needed for the key derivation function. For example (output shortened):

```
# cryptsetup luksDump /dev/vda6
LUKS header information
Version:          2
Epoch: ...

Data segments:
  0: crypt
    ...

Keyslots:
  0: luks2
    Key:        512 bits
    Priority:   normal
    Cipher:     aes-xts-plain64
    Cipher key: 512 bits
    PBKDF:      argon2i
    Time cost:  4
    Memory:     270246
    Threads:    ...
...
```

The example shows that the memory needed is 270246 KB, or approximately 271 MB. The resulting increase in guest size is:

```
2 * (512 MB + 271 MB) = 1.6 GB
```

Tip: To reduce the amount of memory that needs to be reserved for kdump, change the LUKS2 key-derivation method from the default Argon2 to PBKDF2. Use the **`cryptsetup luksConvertKey`** command.

Required software

Certain distributions include IBM Secure Execution.

KVM guests in IBM Secure Execution mode are supported as of these distributions:

- Red Hat Enterprise Linux 9.0 with service
- Red Hat Enterprise Linux 8.4 with service
- Red Hat Enterprise Linux 7.9 with service
- SUSE Linux Enterprise Server 15 SP3 with service
- SUSE Linux Enterprise Server 12 SP5 with service
- Ubuntu Server 20.04 LTS with service

KVM hosts in IBM Secure Execution mode are supported as of these distributions:

- Red Hat Enterprise Linux 9.0 with service
- Red Hat Enterprise Linux 8.4 with service
- SUSE Linux Enterprise Server 15 SP3 with service
- Ubuntu Server 20.04 LTS with service

The attestation function is available on IBM z16 and LinuxONE Emperor 4 as of these distributions:

- Red Hat Enterprise Linux 9.1 with service
- Red Hat Enterprise Linux 8.7 with service

For other distributions, see the release notes for availability of the attestation function.

The use of Crypto Express adapters for guests running in secure-execution mode is supported as of these distributions:

- Ubuntu 24.04
- Red Hat Enterprise Linux 9.4 and 8.10
- SUSE Linux Enterprise Server 15 SP6

Required hardware

Certain IBM Secure Execution functions require specific hardware.

KVM guests that run in IBM Secure Execution mode on IBM z16 or IBM LinuxONE 4 with firmware bundle S30 (use bundle S31b or later to install S30) can use Crypto Express domains. A maximum of 12 AP queues per secure guest can be configured.

You can use Crypto Express8S adapters:

- Configured in accelerator mode.
- Configured in Enterprise PKCS #11 coprocessor mode. You require Enterprise PKCS #11 version 5.8.30.

Chapter 5. Workload owner tasks

As the owner of the secure workload, your tasks comprise preparing your workload and a bootable disk image that you can send to the cloud provider. The steps are described as manual steps, but can be integrated into a build pipeline.

Important: These tasks must be performed in a trusted environment. A sandbox or clean room that only you as the workload owner has access to are good options.

kdump consideration: If you configure kdump for the KVM guest, consider that sufficient memory must be reserved for the kdump kernel. If the memory is too small it cannot decrypt the root volume and is not functional. Configure the memory that is reserved for the crash kernel with the `crashkernel` command-line parameter.

Overview of steps

At a minimum, the following steps are required.

1. Encrypt the root file system.
2. Encrypt data volumes.
3. Modify the init RAM file system to mount the encrypted root file system .
4. Modify the root file system to mount the encrypted data volumes.
5. Generate a kernel parameter line.
6. Harden the workload..
7. Use the **genprotimg** command to generate a secure-execution protected image from the kernel, the kernel parameter line, and the initial RAM file system.

You can harden the workload before encrypting the root file system, then the root file system with the hardening changes must be encrypted.

These steps and additional tasks are described in the subsequent sections.

Encrypting the data volumes

Your goal is to prepare a workload for running as securely as possible in the cloud.

Before you begin

You require an encryption process of your choice for your data. Data here means everything except the boot image.

Important:

Do not use logical volumes together with encryption. If your distribution uses a logical volume setup by default, select a manual or expert storage setup to ensure that data is stored directly on LUKS volumes.

If logical volumes are required, use unique volume and non-predictable volume names. For example, use random names or UUIDs as generated with **uuidgen**. Multiple volumes with the same name can result in the wrong volume being mounted. With a known or easily guessed volume name, an attacker might be able to mount an unencrypted, malicious file system.

About this task

To prepare your workload for running securely in the cloud, you need to secure all parts of it. Start by securing the data volumes.

Procedure

Work in a trusted mainframe environment.

1. Prepare your data image.

   The data and the boot information can be on the same or different disk images.

   Encrypt the data partition of your disk with the encryption process of your choice.

   Tip: Use the operating system installer to encrypt the root filesystem, however, do not use the default of logical volumes with LUKS encryption, see Important note in Before you begin.
2. Ensure that the required keys and passphrases are available to the boot process.
   a) Save references to keys (plain format) or pass phrases (LUKS/LUKS2) for each volume in the `/etc/crypttab` configuration file.
   b) Include the `/etc/crypttab` configuration file in the initial RAM file system.

      Because the initial RAM file system will be encrypted, it can hold keys and pass phrases without compromising security.

Results

As shown in Figure 6 on page 16, the workload data is encrypted, and the keys are stored in the bootable image.

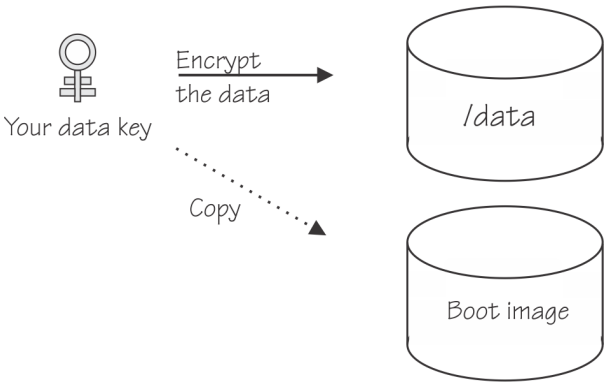


Figure 6. Data volumes for a workload need to be encrypted, using, for example, pervasive encryption

What to do next

Prepare a bootable disk image, see "Preparing the boot image" on page 16.

Preparing the boot image

Prepare a KVM guest for running in IBM Secure Execution mode. The guest that you create for running in a cloud must be adequately secured. Consider all access paths to it, including console logins.

Before you begin

To prepare the guest, you need the Linux boot components:

- Kernel
- An initial RAM file system
- Kernel parameters

About this task

Your starting point is a standard KVM guest. You can use QCOW2, FCP-attached disks, or DASD disks.

A secure execution boot image consists of the encrypted kernel, initial RAM file system, and kernel parameters. It also includes a header that can only be interpreted by an ultravisor of a host system for this secured workload. The header is integrity protected and contains the image encryption key.

Procedure

1. Install a standard Linux instance. This example uses an Ubuntu 20.04 instance.

   Accept the installer defaults, unless you want to use fixed IP addresses.

   In the package selection step, select OpenSSH to use SSH and SCP connections to your guest.

   Encrypt your root filesystem. See Important note in Preparing the boot image.
2. Prepare a kernel parameter file.

   Create a new file, called, for example `parmfile`.

   a. The boot configuration (`zipl.conf`, BLS entries, or `grub.cfg`) of the installed standard Linux instance contains a line that specifies the root device. Copy these parameters to the `parmfile`.

      Tip: Read `/proc/cmdline` to find out which parameters were used to start your Linux instance.

   b. Define a bounce buffer with the `swiotlb=` parameter.

      Tip: Use a setting of 262144 for best results.

      Add the `swiotlb=` parameter to the parameter line.

   Your `parmfile` might, for example, look like:

```
root=UUID=694fd9a4-4180-4c47-92e0-7aa4fe06d370 crashkernel=196M swiotlb=262144
```

   You can use virt-install to set up a Linux instance:

   a. Download the Ubuntu 20.04 CD-ROM image for IBM Z from http://cdimage.ubuntu.com/releases/focal/release/ into the directory `/var/lib/libvirt/images`

   b. Use a command like the following to set up secguest1 as an Ubuntu 20.04 instance with 4 GB of memory on an 8 GB QCOW2 disk with the default libvirt network:

```
# virt-install --name secguest1 --memory 4096 --disk size=8 \
--cdrom /var/lib/libvirt/images/ubuntu-<version>-live-server-s390x.iso
```

   Obtain the domain configuration-XML with the following command:

```
virsh dumpxml secguest1 > secguest1.xml
```

   Remember to modify the XML to allow bounce buffers with `iommu="on"`.
3. Mount the directories where the kernel, the initial RAM file system, and the kernel parameter file are located.
4. Disable root login on consoles.
   - Enforce secure remote login only.
     - Set up SSHD and SSH keys.
     - Disable login on consoles by disabling serial and virtual TTYs. For example, using systemd:

```
# cat /etc/systemd/system/serial-getty@.service.d/disable.conf
[Unit]
ConditionKernelCommandLine=allowlocallogin
# cat /etc/systemd/system/autovt@.service.d/disable.conf
[Unit]
ConditionKernelCommandLine=allowlocallogin
```

       The example shows how a `disable.conf` file defines a kernel parameter, `allowlocallogin`. With this configuration file, local logins are possible if the Linux instance is started with the

`allowlocallogin` kernel parameter in the parameter file that is used to build the image. Use this technique, for example, for debugging.

- Remove information leaks on the kernel console by setting `loglevel=0` and `systemd.show_status=0`.
- On Ubuntu Server: Edit the `/etc/securetty` to prevent console logins. Remove the contents of the file to not allow any logins. This prevents any logins from the hypervisor environment.

  For example, to remove all content in `/etc/securetty`, issue the following command:

```
# echo > /etc/securetty
```

- Disable the debug shell in initramfs by setting the `panic=` parameter.
- Disable debug, emergency, and rescue shells. For example, using systemd, mask the corresponding services:

```
# systemctl mask emergency.service
# systemctl mask emergency.target

# systemctl mask rescue.service
# systemctl mask rescue.target
```

5. Avoid using the virtio-rng.

   To defend against a possible malicious random-number generator on the host, exclude the virtio-rng. You can do this, for example, by using a module configuration file, `/etc/modprobe.d/virtio-rng.conf` , with the following content:

```
blacklist virtio-rng
```

6. Your guest runs in the context of a virtual server. The virtual server defines the virtual hardware. IBM Secure Execution has configuration requirements on the virtual server. See “Starting the secure virtual server” on page 51.

   Configure the QCOW2 image according to your needs. Pre-allocate it to optimize performance, or use a sparse setting to minimize size.

   For more information about the domain configuration-XML and how to configure virtual servers, see *KVM Virtual Server Management*, SC34-2752.

   Tip: Use **`virt-manager`** to work with the XML.

What to do next

Ensure that your guest boots and can performs its tasks, and make the guest secure with the **`genprotimg`** command as described in “Securing the guest” on page 18.

Test your image

Before you secure your image and send it to the cloud provider, test it to ensure that it boots and performs its tasks as expected.

Securing the guest

To convert the standard KVM guest into an IBM Secure Execution guest, run the **`genprotimg`** command. Also create a domain configuration-XML.

Before you begin

You require the **`genprotimg`** command from the s390-tools package. For more details about the **`genprotimg`** command, see “genprotimg - Generate an IBM Secure Execution image” on page 60.

The **`genprotimg`** command requires the following input:

- The original guest kernel.
- The original initial RAM file system.
- A file containing the kernel parameters.
- The public host key document.
- To verify the host key document, the IBM Z signing-key certificate, and the DigiCert intermediate certificate.
- The output file name of the resulting bootable image.

You must obtain the public host key document from your cloud provider. It must be available where you are preparing the guest.

Procedure

1. The **genprotimg** command is part of the s390-tools package. If it is not already installed, download the package into the file system on your Linux instance and install it.

   For example, on an Ubuntu system, use the following command to install the s390-tools package:

   ```
   # apt install s390-tools
   ```

2. Generate the secure image.

   Run the **genprotimg** command, specifying the kernel, initial RAM disk, parameter file, host key document, the IBM Z signing-key certificate, the Digicert intermediate certificate, and the resulting image name. Issue a command of the following form:

   ```
   # genprotimg -i <image> -r <ramdisk> -p <parm_file> \
    -k </path/to/host-key-doc>.crt --cert <ibm_signkey> --cert <digicert_intermediate> -o
   <output_image>
   ```

   where the host key document must match the host system for which the image is prepared. Specify multiple host key documents to enable the image to run on more than one host.

   For example:

   ```
   # genprotimg -i /boot/vmlinuz -r /boot/initrd.img -p parmfile  \
    -k HKD-8651-000201C048.crt --cert ibm_signkey.crt --cert digicert_intermediate.crt -o /boot/
   secure-linux
   ```

   In this example, the certificate revocation lists are downloaded automatically through an internet connection. If no internet connection is available, you can download the lists manually

3. Update your boot configuration.

   a) Edit `zipl.conf`

   For examples of boot configurations for different Linux distributions, see Appendix B, “Boot configurations,” on page 75.

   Add a new section for the IBM Secure Execution boot image and save. For example:

   ```
   # vi zipl.conf
   ...
   [secure]
   target=/boot
   image=/boot/secure-linux
   ...
   ```

   Specify the location of the mounted kernel, the initramfs and the kernel parameter file directories.

   b) Run **zipl -V**.

   The **zipl** command creates a bootable disk image.

Results

The kernel, initial RAM file system, and parameter file are encrypted. An integrity-protected IBM Secure Execution header is created that contains all information required for booting. The IBM Secure Execution header contains the image encryption key. The header is encrypted with the public host key.

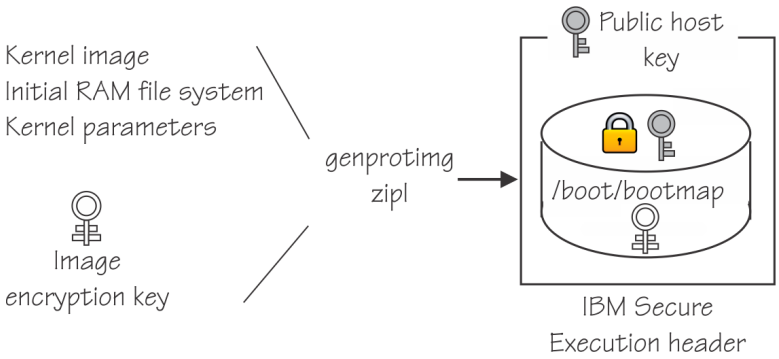


Figure 7. Boot data is consolidated and encrypted by IBM Secure Execution

What to do next

Extract the IBM Secure Execution header for later use with attestation. See Chapter 7, "Attesting a KVM guest," on page 31.

Submitting a secret to the ultravisor

An IBM Secure Execution guest can submit a secret to the ultravisor. The secret must be contained in a an add-secret request.
The add-secret request is protected with the help of the host key and a random Elliptic Curve Diffie–Hellman (ECDH) key pair. Different types of secrets can be submitted, but only null-secrets and association secrets are supported.

Inserting a secret into the ultravisor of a guest is a prerequisite for associating an AP queue with the guest, see "Binding and associating an EP11 adapter AP queue using the chzcrypt command" on page 42.

Before you begin

You require the following hardware and software:

- IBM z16 with the latest firmware updates with Crypto Express support for secure execution on IBM Z.
- A Linux distribution that supports the uv device:
  - Ubuntu 24.04
  - Red Hat Enterprise Linux 9.4 and 8.10
  - SUSE Linux Enterprise Server 15 SP6
- s390-tools version 2.29 or later.

To create the add-secret request you also require:

- A trusted system - a system that can be trusted to not be compromised or tampered with. For example, an attested IBM Secure Execution guest or a local workstation.

  Important: The trusted system must not be the same guest as the one into which you want to insert the secret. This is to prevent the secret from being in cleartext on the target guest.
- The IBM Secure Execution header of the guest you want to work with. For details about how to extract a header, see "Extracting an IBM Secure Execution header" on page 25.
- One or more host key documents.
- An IBM-signing key certificate.

- An certificate authority (CA) certificate.

About this task

You can use the **pvsecret** command to:

- Create an add-secret request. You can create an add-secret request for an IBM Secure Execution guest without being its owner initially. However, you must be the owner of the IBM Secure Execution guest at the time of submitting the add-secret request to the ultravisor. See “Submitting an association secret” on page 21.
- Submit the secret to the ultravisor, see “Submitting an association secret” on page 21.
- Lock the store of secrets, see “pvsecret lock” on page 73.
- List secrets in the store, see “pvsecret list” on page 72.

For the complete command syntax and reference, see “pvsecret - Create requests, add and list secrets, and lock the store of secrets” on page 70.

Submitting an association secret

Create an add-secret request, transfer it to the secure guest, and then submit it to the ultravisor.

About this task

An add-secret request that contains an association secret must be created in a secure environment using the **pvsecret create** command with the **association** sub-command. This add-secret request must be transferred to the IBM Secure Execution guest that shall use the association secret. From within the guest, the add-secret request is submitted to the ultravisor using the **pvsecret add** command.

An association secret is a 32-byte value that can be referred to by a 32-byte secret identifier. If not specified, the command generates a random secret. The **pvsecret** command computes the secret identifier as an SHA-256 hash of a string that you specify.

Procedure

1. On a trusted system, generate an add-secret request. The request is written to a file that you specify.

   Issue a command of the following form:

   ```
   [trusted]# pvsecret create -k <host_key_document> --hdr <secure_exe_header> -o <request_file> \
   -C <CA_certificate> -C <IBM_signing_certificate> association <string>
   ```

   For example, to use a host-key document `z16.crt`, a guest header `se.hdr`, a CA certificate `DigiCert.crt`, and an IBM signing key `ibm-sign.crt`, issue the following command on a trusted system:

   ```
   [trusted]# pvsecret create -k z16.crt --hdr se.hdr -o addSecretReq \
   -C DigiCertCA.crt -C ibm-sign.crt association "myConfidentialSecret"
   ```

   The command creates an add-secret request and writes it to `addSecretReq`. It also creates an identifier for the request, consisting of an SHA-256 hash of the string "myConfidentialSecret".
2. On the KVM guest that is running in secure-execution mode, insert the secret.

   Issue a command of the following form:

   ```
   [se_guest]# pvsecret add <request_file>
   ```

   For example, to use the add-secret request that was created in the previous step, issue:

   ```
   [se_guest]# pvsecret add addSecretReq
   ```

Results

The secret is added to the store of secrets in the ultravisor. You can list all stored secrets by issuing:

```
[se_guest]# pvsecret list
Total number of secrets: 1

0 Association:

5468697320697320612076657279207365637265742073656372657420696 42e
```

The output of **pvsecret list** shows that one secret is listed which has index 0, is of type **Association**, and has the secret ID "546...42e".

As described so far, an add-secret request are limited to secure guests that are booted from a specific boot image. However, without further precautions, nothing prevents an attacker from replacing your add-secret request with their own containing a known secret. See “Preventing the misuse of add-secret requests” on page 22 for further protection.

Preventing the misuse of add-secret requests

You can protect the add-secret request against attacks.

About this task

There are two types of attacks against the secure use of add-secret requests:

- Theft of an add-secret request, which can then be used in a secure-guest instance that is controlled by an attacker.
- Deception, where an attacker substitutes their own add-secret request for the one you intended to use.

To counter the first attack, add-secret requests are bound to a secure-execution image. However, if the secure-execution image is generic, like an appliance from a software vendor, this defense is less effective, requiring further safeguards, such as an extension secret.

An extension secret can be the null secret (default), it can be derived from the CCK, or you can provide a specific extension secret to the add-secret request.

After you prime the ultravisor with an extension secret, all subsequent add-secret requests must include the same extension secret.

Using a simple extension secret

Use the `--extension-secret` option to use a 32-byte random secret to enhance the security of the **pvsecret create** command.

For a specific secure-execution guest instance, the ultravisor only accepts add-secret requests that share the same extension secret. You can specify the extension secret with the `--extension-secret` option to prevent your add-secret request from being used together with an add-secret request of an attacker.

After you prime the ultravisor with an extension secret, all subsequent add-secret requests must include the same extension secret.

Procedure

On a trusted system, add the extension secret to the **pvsecret create** command.

Specify a command of the following form:

```
[trusted]# pvsecret create -k <host_key_document> --se-hdr <secure_exe_header> -o <request_file> \
-C <CA_certificate> -C <IBM_signing_certificate> \
--extension-secret <ext_secret> association <string>
```

For example:

```
[trusted]# pvsecret create -k z16.crt --hdr se.hdr -o addSecretReq \
-C DigiCertCA.crt -C ibm-sign.crt \
--extension-secret myExtSecret association "myConfidentialSecret"
```

Using a customer-communication-key based extension secret

Use the `-cck` option to use an RFC-5869-HKDF derivative of the customer communication key (CCK) to enhance security of the **pvsecret create** command.

This ensures that only the builder of the secure-execution header, that is, someone who knows the CCK, can add add-secrets request to the guest instance.

About this task

During the creation of a secure execution boot image, the creator can specify that the extension secret must be derived from the customer communication key (CCK).

```
# genprotimg <other_genprotimg_options> --enable-cck-extension-secret --comm-key=cck
```

If that option was specified at image creation, the add-secret request must use the corresponding `-cck` *<cck_file>* option. For details about using **genprotimg** to create an image, see “Securing the guest” on page 18.

Note: The **genprotimg** command option is `--comm-key`, but the **pvsecret** command option is `--cck`.

The **pvsecret** command derives the extension secret from the CCK by performing an RFC-5869 HMAC-based extract-and-expand key derivation function (HKDF) operation with hashed message authentication code (HMAC) SHA-512 and "IBM Z Ultravisor Add-Secret" as information material. The firmware verifies that the extension secret was indeed derived from the CCK and aborts the operation if not.

Procedure

Add the `--cck` extension secret to the **pvsecret create** command.

Specify a command of the following form:

```
[trusted]# pvsecret create -k <host_key_document> --hdr <secure_execution_header> \
-o <request_file> -C <CA_certificate> -C <IBM_signing_certificate> \
--cck <cck_file> association <string>
```

For example:

```
[trusted]# pvsecret create -k z16.crt --hdr se.hdr \
-o addSecretReqCCK -C DigiCertCA.crt -C ibm-sign.crt \
--cck cck association "myConfidentialSecret"
```

Binding the request to a specific guest instance

Use the `--cuid` option to use the attestation response to enhance security of the **pvsecret create** command.

This ensures that your add-secret request can only be used for a specific instance of a secure-execution guest.

Before you begin

For the **pvattest** command, s390-tools version 2.29 or later is required.

About this task

Assume that during attestation of an image, the attestation verification saves the Configuration Unique ID (CUID) to, for example, `cuid.yaml`.

```
# pvattest verify <other_verify_options> -i attestationResponse --format=yaml -o cuid.yaml
```

Procedure

To generate a new add-secret request with a random secret, the hash value of "myConfidentialSecret" as identifier, and the CUID of `cuid.yaml`, issue:

```
[trusted]# pvsecret create -k z16.crt --hdr se.hdr -o addSecretReq -C DigiCertCA.crt \
-C ibm-sign.crt --cuid cuid.yaml association "myConfidentialSecret"
```

Results

The command writes the ID to `myConfidentialSecret.yaml` and the encrypted request to `addSecretReq`. If the CUID does not match the CUID from the attestation of the running guest instance, **pvsecret add** fails. The CUID is unique to each guest instance and changes with a reboot.

Proving the origin of an add-secret request to the secure guest

Use the `--user-sign-key` option of the **pvsecret create** command to sign an add-secret request.

Before you begin

You require a certificate and its corresponding private key. Supported key types are:

- RSA, 2048- or 3096-bit
- Elliptic Curve Digital Signature Algorithm (ECDSA) with secp521r1 curve (NIST P521)

To specify your private key, use a PEM or DER format.

About this task

A generic secure guest must only accept secrets from the owner to whom its ownership has been transferred. Here it is assumed that the owner is represented by a certificate linked to a private key known only to the guest's owner. For the guest to verify ownership of an add-secret request, the request must be signed by the owner's key, and the guest must have access to the owner's certificate.

To achieve this, use the `--user-sign-key` option of the **pvsecret create** command to sign an add-secret request. Within the secure guest, use the **pvsecret verify** command to validate the signature of the add-secret request before submitting it to the ultravisor.

Additionally, you can pass arbitrary user-key-signed and request-TAG verified user data to the secure-execution guest. The use of such data is up to you; other than the size restrictions of 128-512 bytes depending on the signing key, there is no limit in usage.

Procedure

1. On a trusted system, add the extension secret to the **pvsecret create** command.

   Specify a command of the following form:

   ```
   [trusted]# pvsecret create -k <host_key_document> --se-hdr <secure_exe_header> -o
   <request_file> \
   -C <CA_certificate> -C <IBM_signing_certificate> \
   --user-sign-key <user_key_file> association <string>
   ```

   For example:

   ```
   [trusted]# pvsecret create -k z16.crt --hdr se.hdr -o addSecretReq \
   -C DigiCertCA.crt -C ibm-sign.crt \
   --user-sign-key myPrivateKey.pem association "myConfidentialSecret"
   ```

Optionally, you can add user data from a file by specifying `--user-data <FILE>`. The maximum size of the user data depends on the signing key that is used. The data can be up to 512 bytes for no user-key, and as little as 128 bytes for an RSA 3096 key. For example:

```
[trusted]# pvsecret create -k z16.crt --hdr se.hdr -o addSecretReq \
-C DigiCertCA.crt -C ibm-sign.crt \
--user-sign-key myPrivateKey.pem --user-data user_data association "myConfidentialSecret"
```

The command generates a new association secret with a random secret and the hash value of `myConfidentialSecret` as secret identifier. The add-secret request is signed with your user-defined private key `myPrivateKey.pem`. The command writes the ID to `myConfidentialSecret.yaml` and the encrypted add-secret request to `addSecretReq`.

2. Verify that the request was signed with the guest owner's private key.

   A Secure Execution guest in possession of the guest owner's certificate can use the **pvsecret verify** command to verify that the request was signed with the guest owner's private key.

   This verification process should be conducted solely within the secure-execution guest. Only upon successful verification, submit the secret to the ultravisor using **pvsecret add**.

   For example, to verify that the request `addSecretReq` was signed with the private key that corresponds to the certificate `myCert.pem`, issue:

   ```
   [secguest]# pvsecret verify --user-cert myCert.pem -o user_data_out addSecretReq
   ```

   If you specified any user data, that user data is part of the output after a successful verification. The previous example saves it at `user_data_out`. By default, the user data is printed to `stdout`.

What to do next

If the verification fails, the guest must discard the add-secret request. Subsequently, depending on the application, the user data in stdout or user_data_out requires processing.

If neither the verification of the add-secret request fails nor the processing of the user data results in rejection of the add-secret request, then the add-secret request may be submitted to the ultravisor.

Extracting an IBM Secure Execution header

Use the **pvextract-hdr** script to obtain the IBM Secure Execution header of a KVM guest running in secure execution mode.

Before you begin

When **pvattest** is installed, a script called **pvextract-hdr** is also installed. Should this script be missing, you can obtain it from GitHub at:
https://github.com/ibm-s390-linux/s390-tools/blob/master/pvattest/tools/pvextract-hdr

About this task

The Secure Execution image, which is used to start a secure guest, includes a secure-execution header (SE-header). This SE-header holds metadata necessary for the ultravisor to validate the Secure Execution image's integrity and unpack it. The SE-Header need not be kept secret because it is safeguarded such that only the Ultravisor from a target host can verify its integrity and access the confidential data within the SE-header.

To create attestation and add-secret requests, you must provide the header as an argument to the respective request creation tools. Given a Secure Execution image, its SE-Header can be extracted on any Linux system when needed.

Procedure

Use the **pvextract-hdr** script to extract the header from the KVM guest.

Use a command of the following form:

```
[secguest]# pvextract-hdr -o <header_file> <path/to/image>
```

In the following example, the header file is written to `hdr.bin`:

```
[secguest]# pvextract-hdr -o hdr.bin /boot/seimage
```

Results

The **pvextract-hdr** script writes the header to a file that you specify. Use this file when attesting the image.

What to do next

Ensure that no non-secure partitions and files on disk are included. See “Guard against non-secure partitions and files on disk” on page 26.

Guard against non-secure partitions and files on disk

You have two options for sending your encrypted boot image to the cloud provider.

Procedure

- Option 1: If you send the boot image that you created in “Securing the guest” on page 18 separately from the volume image, destroy the boot partition before you create the volume image.
  For example, if `/dev/vda1` is the boot partition, use:

  ```
  # shred /dev/vda1
  ```

  For information about how to boot from a separate boot image file, see Step 3 "Configure for direct kernel boot" in “Starting the secure virtual server” on page 51.
- Option 2: If you send the boot image on the unencrypted boot partition, the cloud provider can read from and write to this unencrypted partition.

  To avoid security issues, ensure that:

  a) No sensitive content remains on the unencrypted partition. Use, for example, **sfill** to wipe any free space on the disk to ensure that no traces of confidential data remain. See also “Securely delete sensitive files from the unencrypted boot partition” on page 26.

  b) The unencrypted partition is not mounted by the secure guest.

  Edit `/etc/fstab` and remove `/boot` (or on SUSE Enterprise Linux Server, `/boot/zipl`).

  With no volume mounted at `/boot`, or `/boot/zipl`, potentially non-secure new kernels or kernel updates are written to the `/boot`, or `/boot/zipl`, sub-directories of the root file system, which is backed by an encrypted volume. Because you cannot boot from an encrypted volume, these untrusted kernels are not a threat.

Securely delete sensitive files from the unencrypted boot partition

Sensitive files include the original unencrypted kernel, RAM file system, and kernel parameter file as well as the related entries in the boot configuration. These files could potentially be used by an attacker to obtain secrets.

Use, for example, the **shred** command to remove these files. Then re-run the boot configuration update.

What to do next

Supply your setup details to your provider, see “Communicating your setup to the provider” on page 27.
Transfer the secure disk image and domain configuration-XML to the IBM Secure Execution host.

Communicating your setup to the provider

The cloud provider needs to know your expected disk layout and other information.

Configuration data

Your cloud provider needs to know the amount of disk storage, CPU, and network interfaces of the configuration your workload expects. Include the domain configuration-XML that you created when you send your image to the provider.

Assuming one disk, you need to specify, for example, these items:

Item	Value
Disk	Size and type
Memory	Amount:
No. of network interfaces	
Optional: MAC address	

An example is shown here:

Item	Value
Disk	Size and type: 8 GB, QCOW2
Memory	Amount: 4 MB
No. of network interfaces	2
MAC addresses	52:54:00:36:d3:75, 52:54:00:36:d3:88

Chapter 6. Preventing kernel dumps

Use **pvsecret create** with the `--disable-dump` option to prevent the hypervisor from creating guest kernel dumps.

About this task

If a secure execution image vendor creates a guest image with dumps enabled, any dumps are encrypted with the vendor's keys. However, a tenant who takes ownership of such a guest might prefer to prevent the hypervisor from dumping that guest by using the `--disable-dump` option.

You can use the **pvsecret create** command to only disable dumping, or add the disablement to an add-secret request.

You can enhance the security of the disable-dump request with methods that are described in "Preventing the misuse of add-secret requests" on page 22.

Procedure

- To prevent dumping, use **pvsecret create** with a meta secret to pass the `--disable-dump` flag to the guest. Issue a command of the form:

```
# pvsecret create -k <host_key_document> --hdr <secure_execution_header> -o <request_file> \
--crt <CA_certificate> --crt <IBM_signing_certificate> \
--flags disable-dump meta
```

  For example, to use a host-key document `z16.crt`, a guest header `se.hdr`, a CA certificate `DigiCert.crt`, and an IBM signing key `ibm-sign.crt`, issue the following command on a trusted system:

```
pvsecret create -k z16.crt --hdr se.hdr -o addNoDumpReq \
--crt DigiCertCA.crt --crt ibm-sign.crt \
--flags disable-dump meta
```

  The command prevents any memory dumps from being taken from this Linux instance. The command creates an add-secret request and writes it to addNoDumpReq.

- To prevent dumping and also create an association secret, issue a command of the form:

```
# pvsecret create -k <host_key_document> --hdr <secure_execution_header> -o <request_file> \
--crt <CA_certificate> --crt <IBM_signing_certificate> \
--flags disable-dump association <string>
```

  For example, to use a host-key document `z16.crt`, a guest header `se.hdr`, a CA certificate `DigiCert.crt`, and an IBM signing key `ibm-sign.crt`, issue the following command on a trusted system:

```
pvsecret create -k z16.crt --hdr se.hdr -o addSecretReq \
--crt DigiCertCA.crt --crt ibm-sign.crt \
--flags disable-dump association "myConfidentialSecret"
```

  The command prevents any dumps from being taken from this Linux instance. The command creates an add-secret request and writes it to addSecretReq. It also creates an identifier for the request, consisting of a hash of the association string "myConfidentialSecret".

Chapter 7. Attesting a KVM guest

Use attestation as evidence that the KVM guest runs in secure-execution mode. If the KVM guest was built for one particular IBM Z server, the attestation also verifies that the KVM guest runs on that specific server.

If the KVM guest was built for several servers, the attestation only verifies that the KVM guest runs on one of those servers.

Before you begin

You require the following access rights:

- Access to the secure execution header of the KVM guest to be attested.
- Access to the KVM guest to be attested.
- On the KVM guest to be attested: You need to be able to send and receive requests and responses to the KVM guest.

You require the following input files:

- A host key document of the IBM Z system where the guest to be attested runs.
- The IBM Z signing-key certificate (also called a host-key-signing-key certificate) used by IBM to sign the host key document.
- An intermediate CA certificate used to sign the IBM Z signing-key certificate.
- The IBM Secure Execution header from the KVM guest to be attested, see “Extracting an IBM Secure Execution header” on page 25.

  Using a trusted Linux instance, extract the header from the KVM guest image before you submit it to the cloud provider.

You require the **pvattest** command, which is included in your distribution. For details about the command, see “pvattest - Create, perform, and verify attestation requests” on page 66.

About this task

Two different Linux instances are involved:

- The KVM guest to be attested. In the examples in this section, this KVM guest is called `secguest`.
- A trusted Linux instance. In the examples in this section, this Linux instance is called `trusted`. This instance can be:
  - A Linux instance running on IBM Z hardware. If you want to use IBM Z, the Linux instance should be a previously attested secure execution guest or be on your premises and managed by trusted personnel.
  - A Linux instance running on x86 hardware. No special setup is required. If you want to use a local workstation, it must adhere to the security policies of your organization. For information about building the **pvattest** command, see Chapter 8, “Building pvattest on Linux on x86 hardware,” on page 35.

  The `trusted` Linux instance requires the **pvattest**. The first attestation is typically done on a system you fully control, such as your laptop.

From the trusted Linux instance you send an attestation request from user space to the KVM guest to be attested. The ultravisor processes the request and creates a response. The response is an answer document that you retrieve, and that you must validate. This process is illustrated in Figure 8 on page 32.

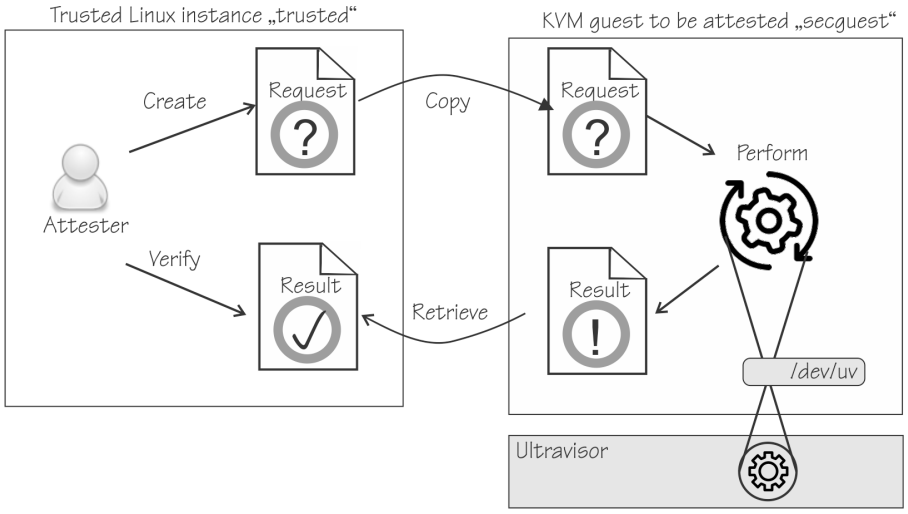


Figure 8. The process of attestation

Procedure

1. On your trusted Linux instance, prepare the attestation request:
   a) Use the **pvattest create** command to create an attestation request.
   Use a command of the following form:

```
# pvattest create -k <host_key_doc>.crt --cert <CA_certificate>.crt \
--cert <IBM_sign_key_cert>.crt --arpk <request_protection_key> -o <request>.bin
```

   For example, using a Linux instance called `trusted`:

```
[trusted]# pvattest create -k hkd.crt --cert CA.crt --cert ibmsk.crt \
--arpk arp.key -o attreq.bin
```

   In this example, the attestation request is written to `attreq.bin`. The command generates a key that protects the request and writes it to a file, here `arp.key`.
   b) Send the attestation request to the KVM guest to be attested.
   Use, for example, the secure-copy (**scp**) command to transfer the request file. For example:

```
[trusted]# scp attreq.bin seguest:
```

2. On the KVM guest to be attested, perform the attestation.
   a) Ensure that the `/dev/uv` device is available.
   For example, assuming that the KVM guest to be attested is called `secguest`:

```
[secguest]# ls /dev/uv
/dev/uv
```

   If the `uv` device is not available, use **modprobe** to load the `uvdevice` module.

```
[secguest]# modprobe --first-time uvdevice
```

   b) Perform the attestation.
   Use a command of the following form:

```
# pvattest perform -i <request>.bin -o <response>.bin
```

   In the following example, the response is written to `attresp.bin`:

```
[seguest]# pvattest perform -i attreq.bin -o attresp.bin
```

3. On your trusted Linux instance:
   a) Retrieve the response.
      For example, to copy the response `attresp.bin` to the current directory on `trusted`:

```
[trusted]# scp seguest:attresp.bin .
```

   b) Obtain the header file or the guest image from the owner of the guest.
      For details about how to extract the header from the image, see "Extracting an IBM Secure Execution header" on page 25. Store the header in a file, for example `hdr.bin`.
   c) Verify the attestation using the **`pvattest verify`** command.
      Use a command of the following form:

```
[trusted]# pvattest verify -i <resp>.bin --arpk <req_prot_key> --hdr <se_guest.hdr>
```

      The command uses the request protection key that was generated during the creation and the IBM Secure Execution header to calculate a corresponding measurement. It then verifies that the two are the same.

      For example:

```
[trusted]# pvattest verify -i attresp.bin --arpk arp.key --hdr hdr.bin
```

Results

If the result of the **`pvattest perform`** command and the calculated results match, the command ends with exit code 0. You can check this by displaying the $? variable with the **`echo`** command:

```
[trusted]# pvattest verify -i attresp.bin --arpk arp.key --hdr hdr.bin
[trusted]# echo $?
0
```

If the result of the **`pvattest perform`** command and the calculated results do not match, the command ends with an error and exit code 2:

```
[trusted]# pvattest verify -i wrongmeas.bin --arpk arp.key --hdr hdr.bin
ERROR: Attestation measurement verification failed:
Calculated and received attestation measurement are not the same.
[trusted]# echo $?
2
```

Chapter 8. Building `pvattest` on Linux on x86 hardware

To build the command, download a source tarball from GitHub, unpack it on your x86 Linux system, switch to the pvattest folder, and run make.

Before you begin

Check if the package is already installed. The package name depends on your distribution:

- Ubuntu or SUSE Linux Enterprise Server: s390-tools
- Red Hat Enterprise Linux or Fedora: s390utils

If you need to install the package, you need a x86 machine that you have full control over, and that adheres to your organization's security standards.

The following example assumes that the s390-tools version is s390-tools 2.23.0.

Check the s390-tools/README for any dependencies. If the dependencies are not fulfilled, the build fails but tells you which packages are required.

Procedure

1. Download a tarball from https://github.com/ibm-s390-linux/s390-tools/releases
   For example:

```
$ wget https://github.com/ibm-s390-linux/s390-tools/archive/refs/tags/v2.23.0.tar.gz
```

2. Unpack the tarball.
   For example:

```
$ tar xf s390-tools-2.23.0.tar.gz
```

3. Switch to the pvattest folder in the unpacked tarball directory.
   For example:

```
$ cd s390-tools-2.23.0/pvattest
```

4. Run the **make** command to compile the **pvattest** command.
   For example:

```
$ make
```

Results

The **pvattest** binary now resides in `s390-tools-2.23.0/pvattest/src`.

Chapter 9. Crypto Express adapters for secure-execution guests

You can use Crypto Express adapters for KVM guests that run in secure-execution mode.

Each adapter is divided into multiple domains. Each domain acts as an independent cryptographic device, for example, as a hardware security module (HSM), with its own state, including its own HSM master key. In Linux, cryptographic adapter resources are managed as AP queues. An AP queue corresponds to a specific cryptographic domain on a specific cryptographic adapter and is denoted by the pair of adapter ID and domain ID in hexadecimal format, for example, 27.0014, 28.0014, or 28.0015. This is also called an AP queue number (APQN).

Prerequisites

You need a secure-execution boot image that supports the insertion of secrets into the ultravisor, see "Submitting an association secret" on page 21.

You can use Crypto Express8S adapters:

- Configured in accelerator mode.
- Configured in Enterprise PKCS #11 coprocessor mode. You require Enterprise PKCS #11 version 5.8.30.

The adapter domains must be configured in passthrough mode (dedicated) for Crypto Express8S adapters. A maximum of 12 adapter domains per secure guest can be configured.

Binding

Both accelerator and Enterprise PKCS #11 coprocessor mode AP queues must be *bound* to the secure guest.

For a Crypto Express adapter in accelerator mode, binding is all you need to do. For details, see "Crypto Express adapter in accelerator mode" on page 38.

Associating

To use HSMs, that is Crypto Express adapters in Enterprise PKCS #11 coprocessor mode, you must also *associate* corresponding AP queues with a secret. A secure-execution guest must submit the secret to the ultravisor before it can be associated with an AP queue.

Before an AP queue is associated with an association secret, you should verify that the adapter domain addressed by the AP queue is configured as expected. In particular, confirm that the master key verification pattern of the AP queue is the expected one.

Associating a secret with an AP queue that is configured with the wrong HSM master key might lead to security issues.

All requests that do not involve a secure key can be submitted to an AP queue that is bound, but not yet associated. Such requests include querying the properties of an EP11 domain, and issuing the commands needed to set the HSM master key through the **ep11TKEd**.

For details, see "Binding and associating an EP11 adapter AP queue using the chzcrypt command" on page 42.

Crypto Express adapter in accelerator mode

To use a Crypto Express adapter in accelerator mode, you must *bind* an AP queue to the secure-execution guest.

About this task

You can choose between these methods to bind an AP queue from a Crypto Express adapter that is configured in accelerator mode:

- “Binding an accelerator AP queue using the chzcrypt command” on page 38
- “Binding an accelerator AP queue using the pvapconfig command” on page 39

Binding an accelerator AP queue using the chzcrypt command

To use a Crypto Express adapter in accelerator mode, you must *bind* an AP queue to the secure-execution guest. You can use the **chzcrypt** command with the `--se-bind` option to bind an accelerator AP queue to a secure-execution guest.

Alternatively you can use the **pvapconfig** command to bind AP queues.

About this task

This example uses the **chzcrypt** and **lszcrypt** commands to bind an AP queue. For an alternative, see “pvapconfig - Implement an AP queue configuration ” on page 63.

Procedure

1. Optionally, on the secure-execution guest, list the available AP queues.
   Use the **lszcrypt** command with the `-V` option to see AP queues listed under SESTAT.
   For example:

```
[secguest]: lszcrypt -V
CARD.DOMAIN  TYPE   MODE        STATUS   REQ... PENDING HWTYPE QDEPTH FUNCTIONS   DRIVER    SESTAT
-------------------------------------------------------------------------------------------------
0f           CEX8A  Accelerator online        0       0     14     08 -MC-A-NF-   cex4card  -
0f.0014      CEX8A  Accelerator online        0       0     14     08 -MC-A-NF-   cex4queue unbound
```

   AP queues that are available for binding are marked unbound.
   SESTAT can show the following states:

   - `usable` - the AP queue can be used for cryptographic requests.
   - bound - the AP queue is bound but not associated.
   - unbound - the AP queue is unbound and must be bound to this secure-execution guest to use it.
   - `illicit` - the AP queue is not available for this secure-execution guest.

2. To bind an AP queue to the guest, issue a command of the following form:

```
[secguest]: chzcrypt --se-bind <aa.dddd>
```

   where *<aa>* is the adapter ID of the cryptographic device and *<dddd>* is the domain.
   For example, to bind the unbound AP queue 0f.0014 to the secure-execution guest, issue:

```
[secguest]: chzcrypt --se-bind 0f.0014
```

3. Optionally confirm that the AP queue is now bound and usable.
   Use the **lszcrypt** command again to check that the status of the AP queue is now `usable`.
   For example:

```
[secguest]: lszcrypt -V
CARD.DOMAIN  TYPE   MODE        STATUS   REQ... PENDING HWTYPE QDEPTH FUNCTIONS   DRIVER    SESTAT
-------------------------------------------------------------------------------------------------
0f           CEX8A  Accelerator online        0       0     14     08 -MC-A-N-F-  cex4card  -
0f.0014      CEX8A  Accelerator online        0       0     14     08 -MC-A-N-F-  cex4queue usable
```

Results

After successfully binding an accelerator AP queue, you can use it to send requests and receive replies for clear key cryptography. The AP queue is now exclusively available to the secure guest. Other operating systems, including that of the KVM host, cannot access the AP queue. However, the KVM host can, when needed, reset the cryptographic resource. As a result, the AP queue is unbound in the secure guest, which leads to failures of further cryptographic requests from the secure guest.

What to do next

You can unbind the AP queue from the guest by using the **chzcrypt** command with the `--se-unbind` option.

For example, to unbind AP queue 0f.0014, issue:

```
[secguest]: chzcrypt --se-unbind 0f.0014
```

Binding an accelerator AP queue using the pvapconfig command

You can use the **pvapconfig** command to implement AP queue device configurations that are defined in a YAML configuration file to bind an AP queue from a Crypto Express adapter that is configured as an accelerator.

About this task

To use a Crypto Express adapter in accelerator mode, all you need to do is *bind* it to the secure guest.

You use a YAML configuration file with the specifications of your Crypto Express adapters as input for the **pvapconfig** command. The command triggers a search for AP queues that are available to the secure guest and that satisfy the specifications in the YAML configuration file. All AP queues that satisfy the entries in the YAML configuration file are bound to the guest.

Any existing AP queue configurations that do not match any of the entries in the configuration file are reset and unbound.

Procedure

1. Optionally, on the secure-execution guest, list the available AP queues.

   Use the **lszcrypt** command with the `-V` option to see AP queues listed under SESTAT.

   For example:

```
[secguest]: lszcrypt -V
CARD.DOMAIN  TYPE   MODE        STATUS  REQ... PENDING HWTYPE QDEPTH FUNCTIONS   DRIVER    SESTAT
-------------------------------------------------------------------------------------------------
00           CEX8A  Accelerator online       0       0     14     08 -MC-A-NF-   cex4card  -
00.0005      CEX8A  Accelerator online       0       0     14     08 -MC-A-NF-   cex4queue unbound
00.0042      CEX8A  Accelerator online       0       0     14     08 -MC-A-NF-   cex4queue unbound
03           CEX8A  Accelerator online       0       0     13     08 -MC-A-NF-   cex4card -
03.0005      CEX8A  Accelerator online       0       0     13     08 -MC-A-NF-   cex4queue unbound
03.0042      CEX8A  Accelerator online       0       0     13     08 -MC-A-NF-   cex4queue unbound
```

   The output shown is shortened. In the SESTAT column, AP queues that are available for binding are marked unbound.

2. Create a **pvapconfig** configuration file (a `.yaml` file) with an entry that describes the Crypto Express adapters from which you want to use AP queues.

For example, a simple **pvapconfig** configuration file can look like this:

```
# my_acc_apconfig.yaml, a configuration file for an accelerator
- name: my Accelerator
  mode: Accel
  mingen: CEX8
```

where:

- `name` is optional and any name you give the configuration.
- `mode` must be `Accel` for an accelerator.
- `mingen` must be CEX8.

For details about the syntax of the `.yaml` file, see "pvapconfig configuration file" on page 63.

3. Run the **pvapconfig** command with your `.yaml` configuration file as input.
   The command now attempts to establish the configuration in the configuration file.
   For example:

```
[secguest]# pvapconfig my_acc_apconfig.yaml
```

   For details about the **pvapconfig** command, see "pvapconfig - Implement an AP queue configuration " on page 63.
4. Optionally confirm that the AP queues are now bound and usable.
   Use the **lszcrypt** command again to check that the status of the available AP queues is now `usable`.
   For example:

```
[secguest]: lszcrypt -V
CARD.DOMAIN  TYPE   MODE        STATUS  REQ... PENDING HWTYPE QDEPTH FUNCTIONS   DRIVER   SESTAT
-----------------------------------------------------------------------------------------------
00           CEX8A  Accelerator online       0       0     14     08 -MC-A-NF-   cex4card  -
00.0005      CEX8A  Accelerator online       0       0     14     08 -MC-A-NF-   cex4queue usable
00.0042      CEX8A  Accelerator online       0       0     14     08 -MC-A-NF-   cex4queue usable
03           CEX8A  Accelerator online       0       0     13     08 -MC-A-NF-   cex4card -
03.0005      CEX8A  Accelerator online       0       0     13     08 -MC-A-NF-   cex4queue usable
03.0042      CEX8A  Accelerator online       0       0     13     08 -MC-A-NF-   cex4queue usable
```

   AP queues that are available for binding are marked `usable`.

What to do next

To unbind AP queues, remove them from the configuration file and run **pvapconfig** again. If the **pvapconfig** command finds AP queues that are bound, but no longer exist in the configuration file, it attempts to unbind them.

Crypto Express adapter coprocessor in EP11 mode

To use a Crypto Express adapter in EP11 coprocessor mode, you need to consider security requirements and the method you want to use for setup.

You can choose between these methods to bind and associate an AP queue that is based on a Crypto Express adapter in Enterprise PKCS #11 coprocessor mode to a secure-execution guest:

- "Binding and associating an EP11 adapter AP queue using the chzcrypt command" on page 42
- "Binding and associating an EP11 adapter AP queue using the pvapconfig command" on page 45

Secure usage requirements for Crypto Express adapters in EP11 mode

You must be able to trust the TKE administrators for the domains and the adapter when you use Crypto Express® adapters in Enterprise PKCS #11 coprocessor mode.

Secure usage of AP queues in Enterprise PKCS #11 coprocessor mode requires careful assignment of master keys to adapter domains and association of association secrets with certain master keys. Trust

in TKE administrators for Crypto Express® adapter domains is paramount. If you cannot trust your TKE domain administrator, you cannot use AP queues securely. This is a general requirement and not specific to IBM Secure Execution for Linux.

Requirement 1: TKE domain administrators of your adapter domains

The TKE domain administrators must provide the administrators of your secure-execution guest with necessary information, such as:

- Adapters (SNs) and domains that are configured for your use, potentially communicating installed certificates in the domains.
- The master key verification patterns of the HSM master keys (aka EP11 wrapping keys) that are installed in your adapter domains.
- Timely communication of any changes to HSM master keys in the adapter domains, such as master key rolls.
- Timely communication of zeroization of adapter domains.

Further, TKE domain administrators must ensure that master keys are configured uniquely in each domain in the same Crypto Express adapter. That is, no master key must be configured in a domain that was, is, or will be configured in another domain in the same adapter. This precludes any domain from using a master key assigned to another domain in the same adapter. Consequently, domains in the same adapter allocated to the same guest must not share master keys. For redundancy, you can configure the same master keys in domains that are contained in separate adapters.

Requirement 2: Protection against adapter zeroization

A TKE adapter administrator holds the power to zeroize a whole Crypto Express adapter and thus zeroize all domains in that adapter. Unless the TKE adapter administrator reliably announces each adapter zeroization, this action can result in a loss of control over domains previously owned by trusted domain administrators.

To maintain control and trust, secure-execution guest administrator applications must verify master key verification patterns for every secure key generated by adapter domains against those communicated by the trusted TKE domain administrator.

Note: The openCryptoki EP11 token can be configured to generate only secure keys with an expected master key verification pattern. For secure keys generated with the **zkey** command to be used with **dm-crypt**, the **zkey list** command displays the HSM master key verification pattern of the generated keys.

Restriction: Policies for association keys

The owner of a secure-execution guest must refrain from associating an association secret with different master key verification patterns, unless these patterns are sequentially related due to master key roll operations. That is, if two domains serve distinct purposes, they should not share the same association pattern.

Restriction: Never reuse the same association secret for two AP queues of the same adapter

Adhere to the restriction imposed by current IBM Z and IBM LinuxONE firmware, preventing the use of the same association secret with two AP queues of the same adapter. This restriction aims to prevent unexpected side effects that are associated with resetting an EP11 AP queue.

Planning for redundancy

It is considered secure to use the same master key and association secret on two domains located on different adapters. Therefore, for redundancy or backup purposes, place the redundant or backup domain

on a separate adapter from the primary domain. This practice not only enhances security but also ensures hardware redundancy for the redundant or backup domains.

Moving master keys

When your TKE domain administrator needs to "move" a master key from one domain to another on the same adapter, ensure that none of your secure execution guests have any AP queues bound to either domain during the migration process.

Binding and associating an EP11 adapter AP queue using the chzcrypt command

To use a Crypto Express adapter in Enterprise PKCS #11 coprocessor mode, you must first bind an AP queue to the guest, verify the master key verification pattern of the AP queue, and then *associate* the AP queue with an association secret. You can use the **chzcrypt** command to bind an AP queue from a Crypto Express adapter configured as an EP11 coprocessor to a secure-execution guest, and associate the AP queue with a secret.

Alternatively you can use the **pvapconfig** command to bind and associate AP queues.

Before you begin

To use a Crypto Express adapter in Enterprise PKCS #11 coprocessor mode, you need an HSM with your HSM master key. Setting an HSM master key is a sensitive task that is performed by an HSM domain administrator, who can be yourself or a trusted agent. However, it is not necessary to set the HSM master key before the guests starts. It is possible to set it using a secure guest that runs an **ep11TKEd**.

About this task

This example uses the **chzcrypt** and **lszcrypt** commands to bind and associate an AP queue. For an alternative, see “pvapconfig - Implement an AP queue configuration ” on page 63.

Procedure

1. Optionally, on the secure-execution guest, list the available AP queues.
   Use the **lszcrypt** command with the -V option to see AP queues listed under **SESTAT**.
   For example:

```
[secguest]: lszcrypt -V
CARD.DOMAIN  TYPE   MODE        STATUS  REQ... PENDING HWTYPE QDEPTH FUNCTIONS   DRIVER     SESTAT
-------------------------------------------------------------------------------------------------
28           CEX8P  EP11-Coproc online       0       0     14     08 -----XN-F-  cex4card   -
28.0014      CEX8P  EP11-Coproc online       0       0     14     08 -----XN-F-  cex4queue  unbound
```

   AP queues that are available for binding are marked unbound.
2. To bind an AP queue to the guest, issue a command of the following form:

```
[secguest]: chzcrypt --se-bind <aa.dddd>
```

   where *<aa>* is the adapter ID of the cryptographic device and *<dddd>* is the domain.
   For example, to bind the unbound AP queue 28.0014 to the secure-execution guest, issue:

```
[secguest]: chzcrypt --se-bind 28.0014
```

3. Optionally confirm that the AP queue is now bound and usable.
   Use the **lszcrypt** command again to check that the status of the AP queue is now bound.
   For example:

```
[secguest]# lszcrypt -V
CARD.DOMAIN  TYPE   MODE        STATUS  REQ... PENDING HWTYPE QDEPTH FUNCTIONS   DRIVER     SESTAT
-------------------------------------------------------------------------------------------------
28           CEX8A  Accelerator online       0       0     14     08 -----XN-F-  cex4card   -
28.0014      CEX8A  Accelerator online       0       0     14     08 -----XN-F-  cex4queue  bound
```

4. Create an add-secret request.

   On a trusted Linux instance, issue a **pvsecret create** command to create the request for adding a secret. The request is written to a file that you specify.Specify a command of the following form:

   ```
   [trusted]# pvsecret create -k <host_key_doc> --hdr <secure_exe_header> \
   -o <request_file> \
   --crt <CA_certificate> --crt <IBM_signing_certificate> association <string>
   ```

   The command uses the following parameters:

   - The `-k` parameter specifies the host key certificate, which assures that the request can only be processed on the intended IBM Z or IBM LinuxONE 4 system.
   - The `--hdr` parameter specifies the IBM Secure Execution header of the secure image, or the IBM Secure Execution image itself.
   - The `-o` parameter specifies the output file for the request. Use the `association` parameter for a phrase that names the secret. The command creates these output files:
     - The request.
     - A .yaml file with a secret ID.
   - Two `--crt` parameters specify the IBM Z signing key and the CA certificate to establish the chain of trust.

   For example, to use a host-key document `z16.crt`, a guest header `se.hdr`, a CA certificate `DigiCert.crt`, and an IBM signing key `ibm-sign.crt`, issue the following command on a trusted system:

   ```
   # pvsecret create -k z16.crt --se-hdr se.hdr -o addSecretReq \
   --crt DigiCertCA.crt --crt ibm-sign.crt association "my secret"
   ```

   The command creates:

   - An add-secret request and writes it to `addSecretReq`.
   - An identifier for the request, consisting of a hash of the given string "my secret", and writes it to `addSecretReq`.

   For details about the **pvsecret create** command, see “pvsecret create” on page 71.

5. Optional: Verify the add-secret request. If you did not create the request yourself, you might want to verify it.

   Assuming that you want to accept only signed add-secret requests, you must verify that the add-secret request is signed by a valid origin, represented by the origin's certificate. For example, to verify that the request `addSecretReq` was signed with the private key that corresponds to the certificate `myCert.pem`, issue:

   ```
   [secguest]# pvsecret verify --user-cert myCert.pem -o user_data_out addSecretReq
   ```

   Add the secret to the ultravisor only if the verification is successful.

6. Add the secret to the ultravisor.

   Assume that you created an add-secret request with the **pvsecret create** command, and the request was saved to `addSecretReq`

   On the guest, issue the following command:

   ```
   [secguest]# pvsecret add addSecretReq
   ```

7. List the secrets in the ultravisor to find the index of the secret you want.

   Use **pvsecret list** to list the secrets.

   For example:

   ```
   [secguest]# pvsecret list
   Total number of secrets: 1

   0 Ap-Association:
     5468697320697320612076657279207365637265742073656372657420696 42e
   ```

   The secret that was added has index 0. You can add more secrets to associate up to 12 AP queues with the guest.
8. Verify the master key verification patterns (MKVP) of the AP queue.

   Check whether the APQN is configured with the expected MKVP. Check the MKVP by reading the `mkvps` sysfs attribute of a domain:

   ```
   # cat /sys/bus/ap/devices/<aa.dddd>/mkvps
   ```
9. Use the **chzcrypt --se-associate** command to associate one AP queue with one secret.

   Issue a command of the following form:

   ```
   # chzcrypt --se-associate <secret_index> <aa.dddd>
   ```

   For example, to associate the secret with index 0 and AP queue 28.0014:

   ```
   [secguest]: chzcrypt --se-associate 0 28.0014
   ```

   The **chzcrypt --se-associate** command might take a small amount of time to complete.
10. Optionally confirm that the AP queue is now usable with the **lszcrypt** command.
    For example:

```
[secguest]: lszcrypt -V
CARD.DOMAIN  TYPE   MODE        STATUS  REQ... PENDING HWTYPE QDEPTH FUNCTIONS   DRIVER     SESTAT
-------------------------------------------------------------------------------------------------
28           CEX8A  Accelerator online       0       0     14     08 -----XN-F-  cex4card   -
28.0014      CEX8A  Accelerator online       0       0     14     08 -----XN-F-  cex4queue  usable
```

Results

After successfully binding and associating an AP queue, you can use it to send requests and receive replies for secure key cryptography. The AP queue is now exclusively available to the secure guest. Other operating systems, including that of the KVM host, cannot access the AP queue. However, the KVM host can, when needed, reset the cryptographic resource. As a result, the AP queue is unbound in the secure guest, which leads to failures of further cryptographic requests from the secure guest.

What to do next

You can disassociate and unbind an AP queue from a guest by using the **chzcrypt** command with the `--se-unbind` option.

For example:

```
[secguest]: chzcrypt --se-unbind 0 28.0014
```

The command results in the AP queue being unbound, which you can see by displaying it's **SESTAT** attribute with the **lszcrypt** command:

```
[secguest]: lszcrypt -V
CARD.DOMAIN  TYPE   MODE        STATUS  REQ... PENDING HWTYPE QDEPTH FUNCTIONS   DRIVER     SESTAT
-------------------------------------------------------------------------------------------------
28           CEX8A  Accelerator online       0       0     14     08 -----XN-F-  cex4card   -
28.0014      CEX8A  Accelerator online       0       0     14     08 -----XN-F-  cex4queue  unbound
```

Binding and associating an EP11 adapter AP queue using the pvapconfig command

You can use the **pvapconfig** command to implement AP queue device configurations that are defined in a configuration YAML file to bind and associate an AP queue from a Crypto Express adapter configured as an Enterprise PKCS#11 coprocessor.

Use the pvapconfig command to automate the mapping of AP queues to association secrets. The command is also useful if many AP queues need to be configured.

About this task

IBM Secure Execution for Linux uses a special secret to associate a secure guest to an AP queue. The untrusted provider of the host environment configures the AP queue for the KVM guest, but cannot use it once it is associated with the secure guest.

For details of how to enhance the security of an add-secret request, see “Preventing the misuse of add-secret requests” on page 22.

For Crypto Express8S adapters in Enterprise PKCS #11 coprocessor mode, you need to *bind* and *associate* an AP queue with a secret.

The procedure that follows presents a simple example of binding an EP11 AP queue and associating it with a specific HSM master key, thus making it usable for the secure guest.

Note: The simple procedure does not ensure that the origin of the add-secret request is the current owner of the guest.

Procedure

1. On a trusted Linux instance, issue a **pvsecret create** command to create the request for adding a secret. The request is written to a file that you specify.

   Specify a command of the following form:

   ```
   [trusted]# pvsecret create -k <host_key_doc> --hdr <secure_exe_header> \
   -o <request_file> \
   --crt <CA_certificate> --crt <IBM_signing_certificate> association <string>
   ```

   The command uses the following parameters:

   - The `-k` parameter specifies the host key certificate, which assures that the request can only be processed on the intended IBM Z or IBM LinuxONE 4 system.
   - The `--hdr` parameter specifies the IBM Secure Execution header of the secure image, or the IBM Secure Execution image itself.
   - The `-o` parameter specifies the output file for the request. Use the `association` parameter for a phrase that names the secret. The command creates these output files:
     - The request.
     - A .yaml file with a secret ID.
   - Two `--crt` parameters specify the IBM Z signing key and the CA certificate to establish the chain of trust.

   For example, to use a host-key document `z16.crt`, a guest header `se.hdr`, a CA certificate `DigiCert.crt`, and an IBM signing key `ibm-sign.crt`, issue the following command on a trusted system:

```
# pvsecret create -k z16.crt --se-hdr se.hdr -o my_addsecreq \
--crt DigiCertCA.crt --crt ibm-sign.crt association "my secret"
```

The command creates:

- an add-secret request and writes it to `my_addsecreq`
- an identifier for the request, consisting of a hash of the given string "my secret", and writes it to `my_secret.yaml`

For details about the **pvsecret create** command, see “pvsecret create” on page 71.

2. Create a **pvapconfig** configuration file (in `.yaml` format) with an entry that describes your AP queue.

   You need the secret ID from the `.yaml` file created with the **pvsecret create** command in the previous step, the master key verification pattern of the AP queue from the HSM domain administrator, and optionally the serial number of the Crypto Express adapter.

   For example, a simple **pvapconfig** configuration file file can look like this:

```
- mode: EP11
  mkvp: 0xdb3c3b3c3f097dd55ec7eb0e7fdbcb93
  serialnr: 93AADFK719460083
  secretid: 0x5468697320697320612076657279207365637265742073656372657420696642e
```

   For details about the syntax of the **pvapconfig** configuration file, see “pvapconfig configuration file” on page 63.

3. Transfer the request and configuration file to the secure guest.

   You can use secure copy, or a similar program.

   For example:

```
[trusted]# scp my_addsecreq seguest:
[trusted]# scp my_apconfig.yaml seguest:
```

4. On the KVM guest running in secure-execution mode, add the secret to the ultravisor.

   Issue a command of the following form:

```
[se_guest]# pvsecret add <request_file>
```

   For example, to use the add-secret request that was created in the previous steps, issue:

```
[se_guest]# pvsecret add my_addsecreq
```

5. Optionally list the secrets.

   The **pvsecret list** command now displays the secret index and the secret ID.

   For example:

```
[se_guest]# pvsecret list
Total number of secrets: 1

0 Ap-Association:

5468697320697320612076657279207365637265742073656372657420696642e
```

6. After all association secrets are added, you can prevent anyone from adding more secrets. To do this issue the **pvsecret lock** command.

   After this command, the ultravisor rejects any further add-secret requests for your secure guest.

```
[se_guest]# pvsecret lock
```

7. Run the **pvapconfig** command with your `.yaml` configuration file as input.

The command now checks for AP queues that are available to the secure guest and that satisfy the specifications in the configuration file. The master key verification pattern assures that your guest is not presented with a tampered AP queue.

For example:

```
[se_guest]# pvapconfig my_apconfig.yaml
```

For details about the **pvapconfig** command, see “pvapconfig - Implement an AP queue configuration ” on page 63.

Results

The ultravisor exclusively binds the AP queue to the secure guest. Other operating systems, including that of the KVM host, cannot access the AP queue or observe exchanges between the secure guest and the AP queue.

The ultravisor uses the secret to associate the secure guest with the AP queue. Only the secure guest that has created the secret can access the AP queue. Malicious peers cannot exploit a stolen secure key.

What to do next

You can automate steps 5 - 7 of the previous section by running the **pvapconfig** command with a policy that describes which association secrets shall be used with which APQN configurations. See “Preventing the misuse of add-secret requests” on page 22.

Chapter 10. Cloud provider tasks

As a cloud provider, your tasks comprise setting up the KVM host and running the workload provided to you by a customer.

Before you start

You require a z15 or later mainframe with the IBM Secure Execution technology enabled and the IBM provided key bundles applied.

For information about enabling IBM Secure Execution, see *IBM Dynamic Partition Manager (DPM) Guide*, SB10-7176-02.

For how to install a key bundle, see "Importing key bundles" on page 49.

Tip: For setups with many or large guests, a large value for `vmalloc`, for example `vmalloc=1T`, might help to ensure that enough virtual contiguous host memory is available for guest addressing. Increasing the value does not cause more memory to be used.

Providing cloud customers with the machine serial number

Cloud customers need the serial number of the machine that will host their workload to request host key documents from Resource Link.

Procedure

- On the HMC Systems Details Dialog, go to the Product Information tab.

  Find the Machine Serial in the table. The Machine Serial consists of the Country of Origin Code and the serial number. The five right-most digits constitute the serial number needed for Resource Link.
- Alternatively, read the five right-most digits of the Sequence Code field from `/proc/sysinfo`, for example:

```
# cat /proc/sysinfo
Manufacturer:          IBM
Type:                  3931
LIC Identifier:        401e26ff62dc9b82
Model:                 701              A01
Sequence Code:         0000000000012345
...
```

  Here 12345 is the serial number.

Importing key bundles

After ordering and installing the IBM Secure Execution for Linux feature you can import key bundles, if none are installed.

About this task

You need to import keys only once, because keys are reimported with any initial microcode load (IML). To import a key bundle, follow these steps:

Procedure

1. Log in to the SE or HMC using an ID with sufficient permissions.
2. Open the **System Details** task to the Instance Information tab
3. Click the Manage button next to **Secure Execution for Linux**

4. On the **Manage Secure Execution Keys** panel, if **Host key** shows Not Installed, click the **Update** button. See Figure 9 on page 50.

   Do not change the bundle file names.

   You can import one key bundle at a time, from either an FTP server or a removable-media device.

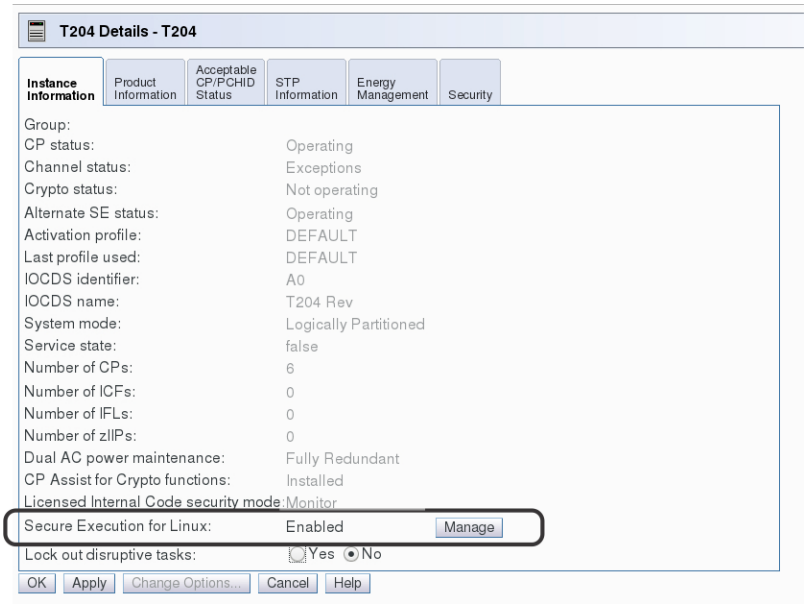


Figure 9. HMC showing how to import a key bundle

Enabling the KVM host for IBM Secure Execution

A cloud provider sets up a KVM host in an LPAR for IBM Secure Execution.

About this task

The KVM host must opt in to IBM Secure Execution, that is, to use the Ultravisor. Use the `prot_virt` kernel parameter to opt in for IBM Secure Execution on the host.

Procedure

Modify a Linux instance to be able to act as an IBM Secure Execution host.

1. Modify the boot configuration.

   For example, if you use **`zipl`**, add the parameter `prot_virt` to the parameters in the `zipl.conf` file and save.

   For example:

```
# vi zipl.conf
...
parameters=" ...prot_virt=1"
...
```

   Run **`zipl`**. For more information about **`zipl`**, see the *Device Drivers, Features, and Commands* or the man page.

2. From the HMC, IPL the device, which then boots the secure KVM host.

   The KVM host then donates some memory to the ultravisor. The ultravisor uses the memory to store the security context for memory in the LPAR. Because of this memory donation, the KVM host sees slightly less memory than what is available in the LPAR. The resulting setup is shown in Figure 10 on page 51.

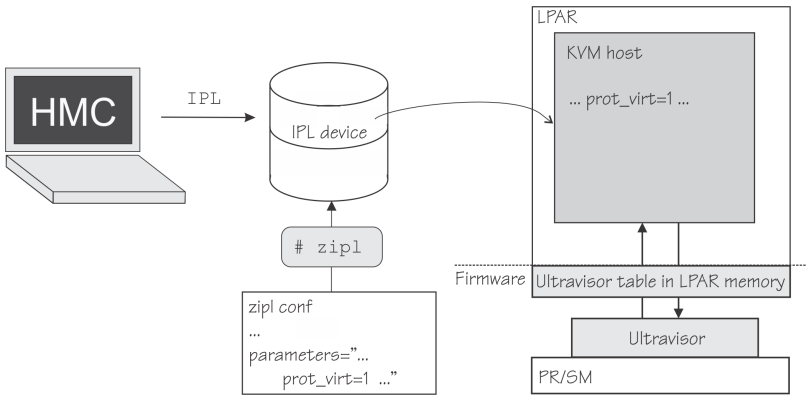


Figure 10. A KVM host is set up to run in IBM Secure Execution mode

3. Verify that the opt-in was successful.

   Check the output of the **dmesg** command. The command must show that memory was reserved for the ultravisor.

   For example:

```
[ 1.010810] Reserving 322MB as ultravisor base storage
```

   The exact amount varies with the size of the LPAR.

   Tip: In a trusted environment, if your distribution supports it, you can read sysfs attributes as indicators of IBM Secure Execution mode:

   - `sys/firmware/uv/prot_virt_host`. The KVM host runs in IBM Secure Execution mode if the value is 1.
   - `sys/firmware/uv/prot_virt_guest`. The KVM guest runs in IBM Secure Execution mode if the value is 1.

   Note that this does not constitute full proof that the host or guest is secure.

Starting the secure virtual server

On the KVM host, create a domain configuration-XML for the virtual machines that are to run in IBM Secure Execution mode.

Before you begin

You need a bootable disk image that is encrypted with the public host key of the mainframe on which you want to run it. See “Preparing the boot image” on page 16.

Procedure

1. Place the bootable disk image on the KVM host file system in `/var/lib/libvirt/images`
   For example, assuming that the image is called `secguest1.img`:

```
# ls /var/lib/libvirt/images
...
secguest1.img
...
```

2. Modify the domain configuration-XML you received from your customer.

   Add the launchSecurity element with type s390-pv to set defaults that simplify configuring the virtual server for IBM Secure Execution for Linux.

   a) Optional: Confirm that this setting is available in your environment

Look for the following line in the output of the **virsh domcapabilities** command:

```
<s390-pv supported="yes">
```

b) Add the launchSecurity element.
For example:

```
<domain type="kvm">
    ...
    <launchSecurity type="s390-pv"/>
    ...
</domain>
```

For example, this setting makes the required bounce buffer for virtio devices the default and you do not have to specify it explicitly for each device. This setting also leads to warning messages if the CPU model of the virtual server does not include all features that are required by IBM Secure Execution for Linux.

Manual domain-XML configuration

If the output of the **virsh domcapabilities** command shows that you do not have support for the launchSecurity element, you must configure the domain XML manually:

a. Ensure that the XML has `iommu="on"` set to allow the use of bounce buffers on every element that represents a virtio device, for example, the `<disk>`, `<serial>`, and `<interface>` elements.

b. Do not define a memory balloon device for secure guests. Use the following definition in the guest XML:

```
<memballoon model='none'/>
```

For example, the following domain configuration-XML, called `secguest1.xml`, configures a virtual machine called secguest1 that allows bounce buffers:

```
<domain type="kvm">
   <name>secguest1</name>
   ...
   <devices>
      <disk type="file" device="disk">
         <driver name="qemu" type="raw" cache="none" io="native" iommu="on"/>
         <source file="/var/lib/libvirt/images/secguest1.img"/>
         <target dev="vda" bus="virtio"/>
         <address type="ccw" cssid="0xfe" ssid="0x0" devno="0x1108"/>
         <boot order="1"/>
      </disk>
      ...
      <memballoon model='none'/>
   </devices>
```

For details about the domain configuration-XML and how to configure virtual servers, see *KVM Virtual Server Management*, SC34-2752.

Tip: Use **virt-manager** to work with the XML.

3. Optional: Configure for direct kernel boot.

If you received the secure boot image as a separate bootable kernel image file, modify the domain XML for a direct kernel boot.

For example, this domain XML configures a guest that is booted from a kernel image:

```
<os>
     ...
     <kernel>/var/lib/images/secure_img</kernel>
</os>
```

The `<kernel>` entry must contain the fully qualified path and file name of the secure boot image file.

4. On the KVM host console, define the virtual machine with the **virsh define** command.
For example, to define secguest1 defined by the secguest1.xml:

```
# virsh define secguest1.xml
```

5. From the KVM host console, verify that the guest can be started with the **`virsh start`** command. For example, to start secguest1:

```
# virsh start secguest1
```

Results

The KVM guest defined by `secguest1.img` starts running in IBM Secure Execution mode. For information about troubleshooting, see “Starting virtual server fails” on page 57.

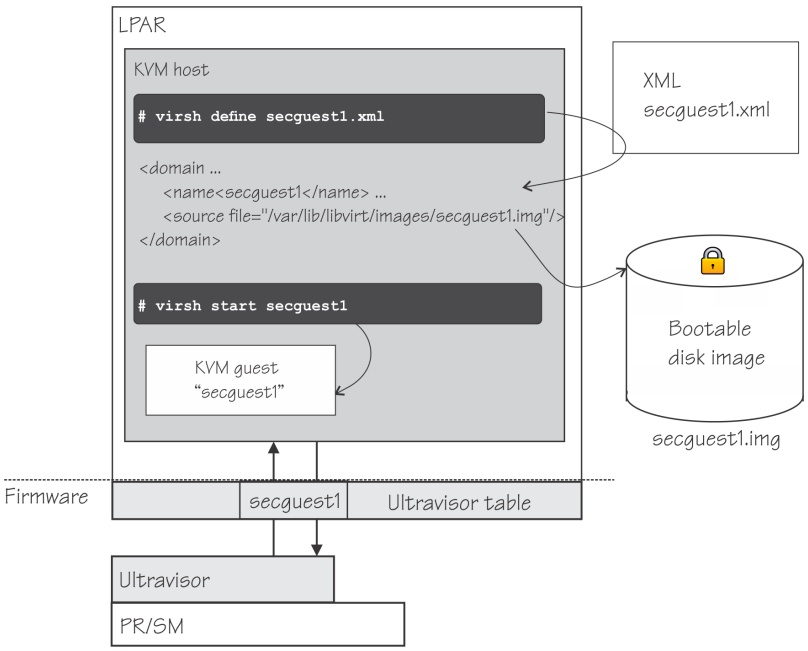


Figure 11. A KVM guest is created from a bootable image to run in IBM Secure Execution mode

Chapter 11. Troubleshooting

Methods of problem determination include examining logs, configuring the virtual server, and using debugfs.

Procedure

Actions that you can take to help with troubleshooting include the following:

- Check that the Linux distribution you are using supports IBM Secure Execution.

  The guest console output reports errors due to a missing facility if the distribution is not current enough.
- Configure the virtual machine to preserve debug data.

  Use the `on_crash` element in the domain configuration XML, for example:

  ```
  <on_crash>preserve</on_crash>
  ```
- On the host, verify that the opt-in to IBM Secure Execution was successful.

  Examine the dmesg for output similar to this example:

  ```
  "[    0.828273] prot_virt: Reserving 322MB as ultravisor base storage"
  ```

  The message shows that the Ultravisor successfully reserved memory and started.
- Examine the output of the QEMU process. The output is saved in `/var/log/libvirt/qemu/<domain>.log`. Problems might be mentioned here, for example, if a virtio device was configured without bounce buffers.
- Check the libvirt log for messages. By default, libvirt log messages are stored in the system journal. You can specify a different location in the libvirt configuration file at `/etc/libvirt/libvirtd.conf`. For more information, see libvirt.org/logging.html.

  For how to set the logging level, see *KVM Virtual Server Management*, SC34-2752.
- Use debugfs, if supported.

  1. Mount debugfs with

     ```
     mount -t debugfs none /sys/kernel/debug/
     ```
  2. Find information relevant to IBM Secure Execution under

     ```
     /sys/kernel/debug/s390dbf/kvm-uv/*
     ```
- If possible, include the output from around the time of failure of the following files with your problem report:
  - `/sys/kernel/debug/s390dbf/kvm-trace/sprintf`
  - `/sys/kernel/debug/s390dbf/kvm-uv/sprintf`

Starting guests fail with error: "Protected boot has failed: 0xa02"

Unable to start IBM Secure Execution guests on a KVM host with numerous, or large guests.

Symptoms

On a KVM host with many guests defined, not all guests can start.

Causes
The kernel has run out of virtual address space to map the areas needed for the guest metadata. The size of the metadata depends on the logical size of the guest, and needs to be allocated fully in the virtual address space of the kernel.

Resolving the problem
Use the vmalloc parameter to add more virtual contiguous host memory for the addressing.

User response: Add vmalloc with a large value to the kernel command line of the KVM host, for example:

```
vmalloc=1T
```

Note: Increasing the `vmalloc` value does not cause more memory to be used.

Attaching a disk with virsh attach-disk causes guest to crash

When attempting to add a disk to a running guest using virsh attach-disk, the virtual machine reports successful attachment.

Symptoms
A **`virsh attach-disk`** command results in a success message and a subsequent crash of a guest in secure execution mode.

Causes
Guests in secure execution mode require `iommu="on"` to be set for all virtio devices to allow the use of bounce buffers. Adding a device without allowing bounce buffers destabilizes the guest.

Resolving the problem
All virtio devices must be configured with the specification `iommu="on"` to allow the use of bounce buffers.

User response: To attach a disk to a guest running in secure execution mode, use the **`virsh attach-device`** command with a device configuration XML. For details about the command, see *KVM Virtual Server Management*, SC34-2752. For an example, see “Example: Adding a disk to a guest running in secure execution mode” on page 56

Example: Adding a disk to a guest running in secure execution mode

Procedure
1. Configure the disk in a separate device configuration XML file. This configuration must include the `iommu="on"` setting.

   Specify `iommu="on"` in the XML file as an attribute of the driver element within the disk configuration.

   For example, in this `MyDisk.xml` file, iommu is set to "on" for a disk:

   ```
   <disk type="block" device="disk">
       <driver name="qemu" type="raw" cache="none" io="native"
        iothread="2" iommu="on"/>
   ...
   ```

2. Specify this device configuration XML file as an argument for the **`virsh attach-device`** command when you hotplug the disk.
   For example, this command picks up the `MyDisk.xml` configuration file for the disk:

   ```
   # virsh attach-device --config Guest1 ~/MyDisk.xml
   ```

   Tip: Also use the **`virsh attach-device`** command to attach interfaces.

Starting virtual server fails

Unable to start virtual server.

Symptoms
The virtual server cannot start.

Causes
The virtual server might not have enough memory to unlock volumes during the boot process.

Resolving the problem
Increase the memory allocated for the virtual server.

User response: Try these options:

- Increase the memory by using the `<memory>` element in the domain-configuration XML of the virtual server.
- Change the LUKS2 key-derivation method from the default Argon2 to PBKDF2 by using the **`cryptsetup luksConvertKey`** command.

Host key document verification fails

If verification of the host key document fails, read what the problem might be and what you can do.

About this task

Commands that verify the host key document might fail due to:

- An expired document, for example the **`genprotimg`** command issues the error:

  ```
  Failed to verify host-key document: '<host_key_document.crt>': validity
  period is not valid
  ```

- A invalid combination of host key document and certificates, for example the **`genprotimg`** command issues the error:

  ```
  Failed to verify host-key document: '<path_to_certificate>': no IBM Z signing
  key that issued this host-key document found
  ```

If the host key document has expired, you need to obtain a new document.

Procedure

Depending on your hardware, select one of these links to obtain a new host key document.

- On IBM z16, go to this link:
  https://www.ibm.com/support/resourcelink/api/content/public/secure-execution-gen2.html

  Follow the link given for the host key document.
- On IBM z15, go to this link:
  https://www.ibm.com/support/resourcelink/api/content/public/secure-execution-gen1.html

  Follow the link given for the host key document.

Appendix A. Commands for IBM Secure Execution

You can use specific commands to secure a KVM guest, extract a header, and attest a KVM guest running in IBM Secure Execution mode.

The following commands exist for working with KVM guests running in IBM Secure Execution mode.

genprotimg - Generate an IBM Secure Execution image

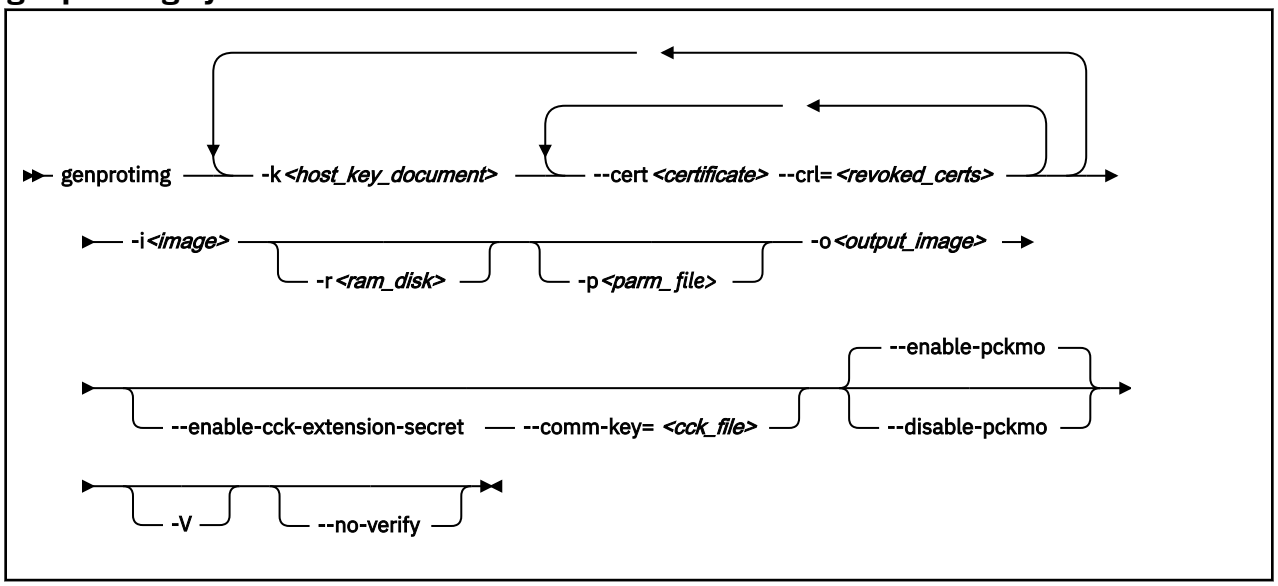
The **genprotimg** command builds an encrypted boot record from a given kernel, initial RAM disk, parameters, and public host-key document.

Command availability

If your distribution does not contain the **genprotimg** command, you can either copy the kernel and initial RAM file to an environment that includes **genprotimg** and build the secure image there, or build the command yourself from the source on GitHub:

```
https://github.com/ibm-s390-tools/s390-tools/tree/master/genprotimg
```

genprotimg syntax



Parameters

-k *<host_key_document>* or --host-key-document=*<host_key_document>*
Specifies the host key document. The document must match the host system for which the image is prepared. Specify multiple host key documents to enable the image to run on more than one host. The document is a plain text file with a name of the form: HKD-*<type>*-*<serial>*.crt

--cert <certificate>
specifies the certificate that is used to establish a chain of trust for the verification of the host key documents. Specify this option twice to specify the IBM Z signing-key certificate (also called the host-key-signing-key certificate) and the intermediate CA certificate (signed by the root CA).

Ignored when `--no-verify` is specified.

--crl=*<revoked_certs>*
Optional: specifies a list of revoked certificates.

-i *<image>* or --image=*<image>*
Specifies the Linux kernel image.

Note: The **genprotimg** command cannot use an ELF file as a Linux kernel image.

-r *<ramdisk>* or --ramdisk=*<ramdisk>*
Specifies a RAM file system.

-p *<parm_file>*or --parmfile=*<parm_file>*
Provides a file with kernel parameters.

-o or --output
Specifies the target image name.

--enable-cck-extension-secret --comm-key=*<cck_file>*
Requires that the extension secret that is used for add-secret requests is based on the customer communication key (CCK).

--disable-pckmo

Disables the Permit CPACF Key Management Operations (PCKMO) support.

The PCKMO options configure key management operations on the virtual server. If enabled, keys can be created that use the DEA, TDEA, AES, or ECC algorithms.

--enable-pckmo

Enables the PCKMO support. This option is the default.

Interface change:

For genprotimg versions with the `--enable-pckmo` option, PCKMO key operations are enabled by default. To confirm that `--enable-pckmo` is available on your distribution, issue:

```
# genprotimg -h
```

If the `--enable-pckmo` option is listed, no further action is needed to enable PCKMO operations. To return to the previous behavior, specify `--disable-pckmo`.

If no `--enable-pckmo` option is listed, and you want PCKMO operations, try:

```
# genprotimg ... --x-pcf '0xe0'
```

-V or --verbose
Prints more runtime information.

--no-verify
Specifies that the host key document is not verified.

Warning: The **genprotimg** as of s390-tools 2.17.0 automatically verifies the host key document. If you need to use the manual procedure (see Appendix D, "Verifying the host key document," on page 79) for verification, use the `--no-verify` option. Working with an unverified key makes your image vulnerable to man-in-the-middle attacks. Whoever gave you the host key document might be able to decrypt your image.

-v or --version
Displays the version information for the command.

-h or --help
Displays out a short help text, then exits. To view the man page, enter **man genprotimg**.

--help-experimental
Displays experimental usage information, then exits.

--help-all
Displays all help text, then exits.

Example: Using genprotimg to generate an IBM Secure Execution image

Assume that you have an Ubuntu guest that you would like to convert into an IBM Secure Execution guest. You have the following information ready:

- The guest has the following `zipl.conf`:

```
[ubuntu]
target=/boot
image=/boot/vmlinuz
ramdisk=/boot/initrd.img
parameters=root=UUID=694fd9a4-4180-4c47-92e0-7aa4fe06d370 crashkernel=196M
```

- A host key document called `HKD-8651-00020089A8.crt`,
- The intermediate CA certificate, here DigiCert, in `DigiCertCA.crt`
- The IBM Z signing-key certificate in `SigningKey.crt`

1. Verify the host key document, see Appendix D, “Verifying the host key document,” on page 79.
2. Create a parameter file called `parmfile`. Copy the content of the parameter that specifies the root device.
3. Specify bounce buffers with a `swiotlb` parameter with a value of `262144`.

   The result is a parameter file with the following content:

   ```
   root=UUID=694fd9a4-4180-4c47-92e0-7aa4fe06d370 crashkernel=196M swiotlb=262144
   ```

4. Generate an IBM Secure Execution image in `/boot/secure-linux` with the command:

   ```
   # genprotimg -i /boot/vmlinuz -r /boot/initrd.img -p parmfile
   -k HKD-8651-00020089A8.crt --cert SigningKey.crt --cert DigiCertCA.crt -o /boot/secure-linux
   ```

pvapconfig - Implement an AP queue configuration

Use the **pvapconfig** command on a KVM guest that is running in IBM Secure Execution mode to implement AP queue device configurations that are defined in a YAML file.

Prerequisites

The **pvapconfig** command requires privileges for the following tasks:

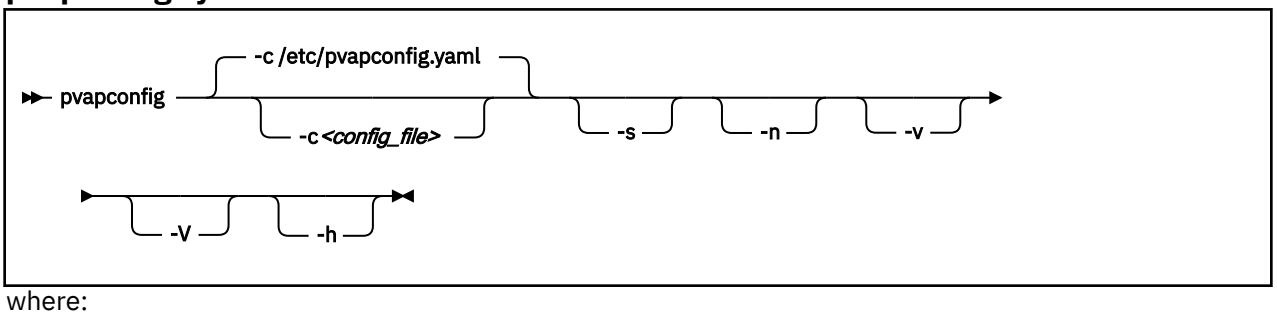
- Read and write sysfs entries.
- Open `/dev/uv` and run **ioctl** calls on this device.

Locking

To prevent multiple instances of **pvapconfig**, the command creates a lock file, `/var/lock/pvapconfig.lock`. A second instance of **pvapconfig** detects this lock file and terminates with an error message.

If for any reason this file still exists as a leftover from a previous **pvapconfig** crash, for example, you must remove it. The lock file contains the process ID of the **pvapconfig** process that created it.

pvapconfig syntax



where:

-c or --config *<config_file>*
specifies the YAML configuration file that specifies the AP queues you want to work with. The default file is `/etc/pvapconfig.yaml`.

-n or --dry-run
processes the configuration, the available AP queues, and secrets. Then simulates the bind, unbind or associate action on the chosen AP queue. Use this option together with the verbose option to see which actions **pvapconfig** would perform if ran without `-n`.

-s or --strict
requires all AP-queue configuration entries to be valid for **pvapconfig** to terminate successfully. Without this option, one valid AP configuration entry is enough for **pvapconfig** to terminate successfully.

-v or --verbose
prints detailed information about the processing.

-V or --version
prints version information and exits.

pvapconfig configuration file

You can use a configuration file to automate which association secrets shall be used with which APQN configuration.

Create a `.yaml` file with an entry that describes your AP queues.

The configuration file contains AP queue configurations in a YAML format. Each configuration must contain the mode specification and the parameters for that mode:

AP queue mode, either EP11 or Accel
Required.

- Specify EP11 for an AP queue based on a Crypto Express adapter in Enterprise PKCS #11 coprocessor mode.
- Specify Accel for an AP queue based on a Crypto Express adapter in accelerator mode.

For EP11:

mkvp
Required. The master key verification pattern (MKVP) of the AP queue as a hex string optionally prefixed with 0x. The hex string value can hold either 16 bytes (32 hex numbers) or 32 bytes (64 hex numbers) but only the leftmost 16 bytes hold MKVP information. The rest is ignored.

secretid
Required. Find the 32-byte ID of the secret in the .yaml file that you generated with the **pvsecret create** command. The secret ID is a hex string optionally prefixed with 0x. This is an SHA-256 hash of the string that was specified with the association subcommand when the add-secret request was created.

serialnr
Optional. The serial number of the Crypto Express adapter as a case-sensitive ASCII string.

mingen
Optional. The only valid value is CEX8.

For Accel:

mingen
Optional. The only valid value is CEX8.

mkvp, serialnr, secretid
Ignored for Accel mode.

Common optional parameters

name
A name of your choice as an ASCII string. Must fit on one line. This is the string that was specified with the association subcommand when the add-secret request was created. If both a `secretid` and a `name` is given, the command checks that `secretid = sha256(name)`.

description
A one-line description as an ASCII string.

Example configuration files

The following file defines an AP queue based on a Crypto Express adapter in Enterprise PKCS #11 coprocessor mode with a secret ID created by the **pvsecret create** command, a verification pattern, and serial number provided by the HSM domain administrator:

```
# A configuration file for an EP11
- mode: EP11
  mkvp: 0xdb3c3b3c3f097dd55ec7eb0e7fdbcb93
  serialnr: 93AADFK719460083
  secretid: 0x5468697320697320612076657279207365637265742073656372657420736563726574206964e
```

The following file defines one accelerator.

```
# A configuration file for an accelerator
- name: my Accelerator
  mode: Accel
  mingen: CEX8
```

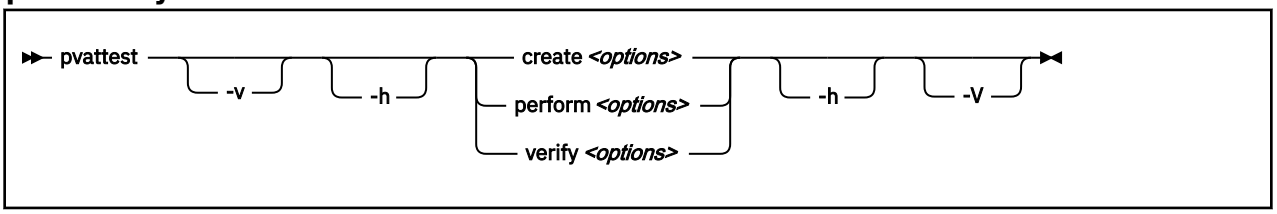
The following file defines a pair of virtual adapters in Enterprise PKCS #11 coprocessor mode with the same HSM master key and the same secret ID, but on different Crypto Express adapters.

```
# A configuration file for a backup pair
- name: my EP11 APQN 1
  mode: EP11
  mkvp: 0xdb3c3b3c3f097dd55ec7eb0e7fdbcb93
  serialnr: 93AADFK719460083
  secretid: 0x546869732069732061207665727920736563726574207365637265742069642e
- name: my EP11 APQN 2
  mode: EP11
  mkvp: 0xdb3c3b3c3f097dd55ec7eb0e7fdbcb93
  serialnr: 93AADHZU42082261
  secretid: 0x546869732069732061207665727920736563726574207365637265742069642e
```

pvattest - Create, perform, and verify attestation requests

Use the **pvattest** command to create an attestation request, perform an attestation measurement, and verify the result.

pvattest syntax



Where:

create *<options>*
On a trusted Linux instance, creates an attestation request, see "pvattest create" on page 66 for details.

perform *<options>*
On a KVM guest running in secure execution mode, performs an attestation measurement, see "pvattest perform" on page 68 for details.

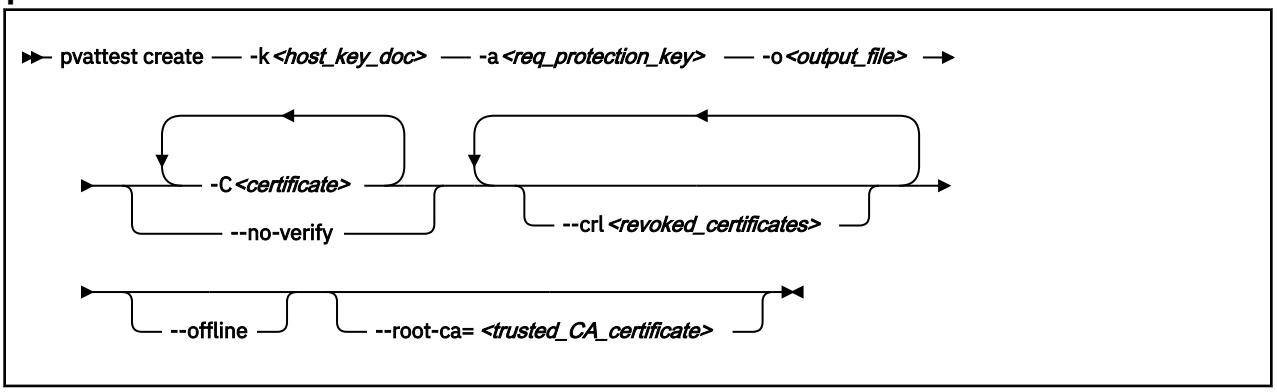
verify *<options>*
On a trusted Linux instance, compares calculated and measured attestation results, see "pvattest verify" on page 68 for details.

-h or --help
Optional: displays short information about command usage. Specify after the main command for general help and after a sub-command for help specific to that command.

-V or --verbose
Optional: displays verbose messages.

-v or --version
Optional: displays version information.

pvattest create



where:

-k
specifies the host key document.

-a *<req_protection_key>* or --arpk=*<req_protection_key>*
generates a random GCM-AES256 key that protects the attestation request. Take care not to inadvertently publish this key, as the attestation could then be tampered with.

-o *<output_file>* or --output-file=*<output_file>*
specifies the file that contains the created request.

-C *<certificate>* or --cert=*<certificate>*
specifies the certificate that is used to establish a chain of trust for the verification of the host key documents. Specify this option twice to specify the IBM Z signing-key certificate and the intermediate CA certificate (signed by the root CA).

Ignored when `--no-verify` is specified.

--crl=*<revoked_certificates>*
Optional: specifies a list of revoked certificates.

--no-verify
Creates the request without verifying the host key document.

Warning: Working with an unverified host key document makes your KVM guest vulnerable to man-in-the-middle attacks.

--offline
Optional: does not download certificate-revocation lists. Every certificate requires a list of revoked certificates. If you specify `--offline`, specify one `--crl` for every `-C`.

--root-ca=*<trusted_CA_certificate>*
Optional: specifies a trusted root CA to use instead of one of the root CAs that are installed on the system.

-h or --help
Optional: shows the help.

-V or --verbose
Optional: provides more detailed output.

Examples: These examples illustrate common uses for **`pvattest create`**.

- A typical attestation request requires the following input:
  - A host key document in `hkd.crt`
  - A CA certificate, here from DigiCert, in `DigiCertCA.crt`
  - The IBM Z signing-key certificate in `SigningKey.crt`

  To create the attestation request, issue:

```
# pvattest create -k hkd.crt -C DigiCertCA.crt -C SigningKey.crt -a arp.key -o arcb.bin
```

  This example generates the following output:

  - A request protection key in `arp.key`
  - An attestation request in `arcb.bin`
- To create an attestation request without downloading revoked certificate lists, but instead use local lists specified with `--crl`, issue:

```
# pvattest create -k hkd.crt -C DigiCertCA.crt -C IbmSigningKey.crt --offline \
--crl DigiCertCA.crl --crl IbmSigningKey.crl --crl rootCA.crl -a arp.key -o arcb.bin
```

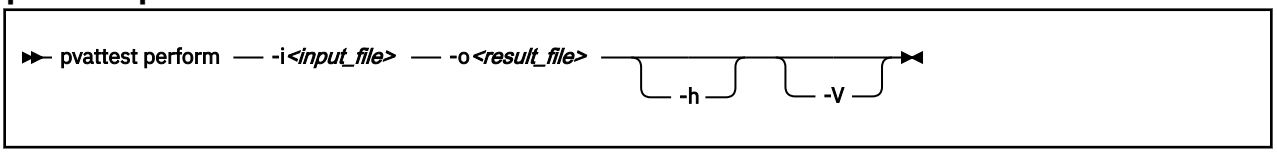
  The example generates the same output as the previous one.
- To create an attestation request on a test system, without verifying the host key document.

  Warning: Use only for testing or when the host key document is already verified.

```
# pvattest create -k hkd.crt --no-verify --arpk arp.key -o arcb.bin
```

pvattest perform



Where:

-i *<input_file>* or --input=*<input_file>*
specifies the attestation request created with the **pvattest create** command.

-o *<result_file>* or --output=*<result_file>*
specifies the file to which the result of the attestation measurement is written.

-h or --help
Optional: shows the help.

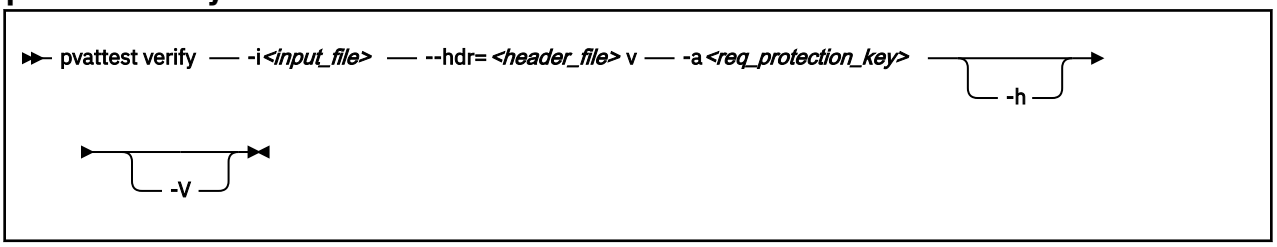
-V or --verbose
Optional: provides more detailed output.

Example:

- To perform an attestation with a request `attreq.bin` and receive the output in `attresp.bin`, issue:

```
# pvattest perform -i attreq.bin -o attresp.bin
```

pvattest verify



Where:

-i *<input_file>* or --input=*<input_file>*
specifies the attestation request created with the **pvattest create** command.

--hdr=*<header_file>*
specifies the header of the KVM guest to be attested .

-a or --arpk=*<req_protection_key>*
specifies the request-protection key that is used to decrypt the request.

-h or --help
Optional: shows the help.

-V or --verbose
Optional: provides more detailed output.

Example:

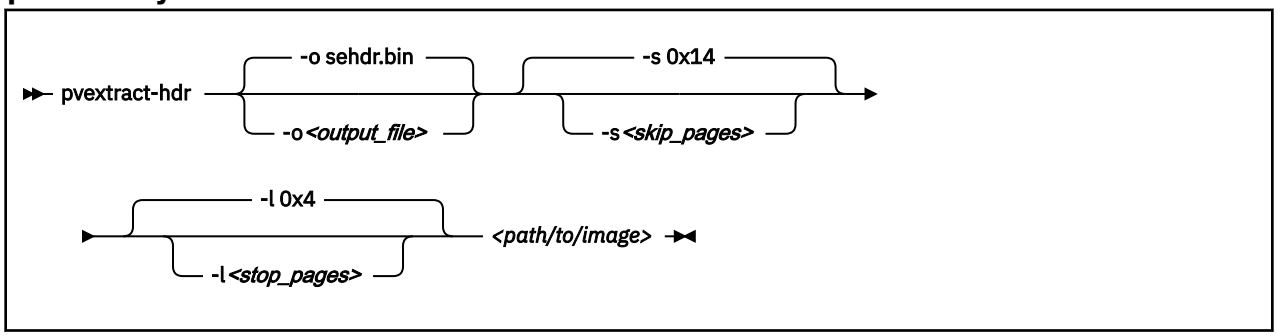
- To verify an attestation with the response from **pvattest perform** in `attresp.bin`, the request protection key generated by **pvattest create** in `arp.key`, and the header in `hdr.bin`, issue:

```
# pvattest verify -i attresp.bin --arpk arp.key --hdr hdr.bin
```

pvextract-hdr - Extract an IBM Secure Execution header

Use the **pvextract-hdr** command to obtain the IBM Secure Execution header and write it into a file.

pvattest syntax



Where:

-o *<output_file>*
specifies the file to which the IBM Secure Execution header is written.

-s *<skip_pages>*
Optional: specifies the number of pages to skip before starting to search for the header. The default is 0x14 pages.

Note: For the `-s` and `-l` options, the defaults are optimized to find headers at an offset of 14 pages and not longer than 2 pages. Accept the defaults unless the command fails and the header is not found.

-l *<stop_pages>*
Optional: specifies the number of pages after which to stop searching for the header. The default is 0x4 pages.

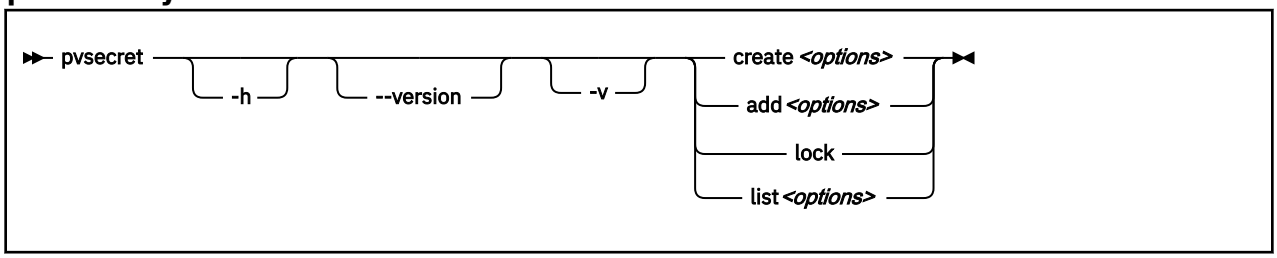
<path/to/image>
the path to the KVM guest image from which you want to extract the header.

-h
Optional: displays short information about command usage.

pvsecret - Create requests, add and list secrets, and lock the store of secrets

Use the **pvsecret** command to create add-secret requests, add secrets to the ultravisor, list secrets, and lock the store of secrets.

pvsecret syntax



Where:

create *<options>*
On a trusted Linux instance, creates an add-secret request, see “pvsecret create” on page 71 for details.

add *<options>*
On a KVM guest running in secure execution mode, adds the secret to the store of secrets, see “pvsecret add” on page 72 for details.

lock
On a KVM guest running in secure execution mode, locks the store of secrets, see “pvsecret lock” on page 73 for details.

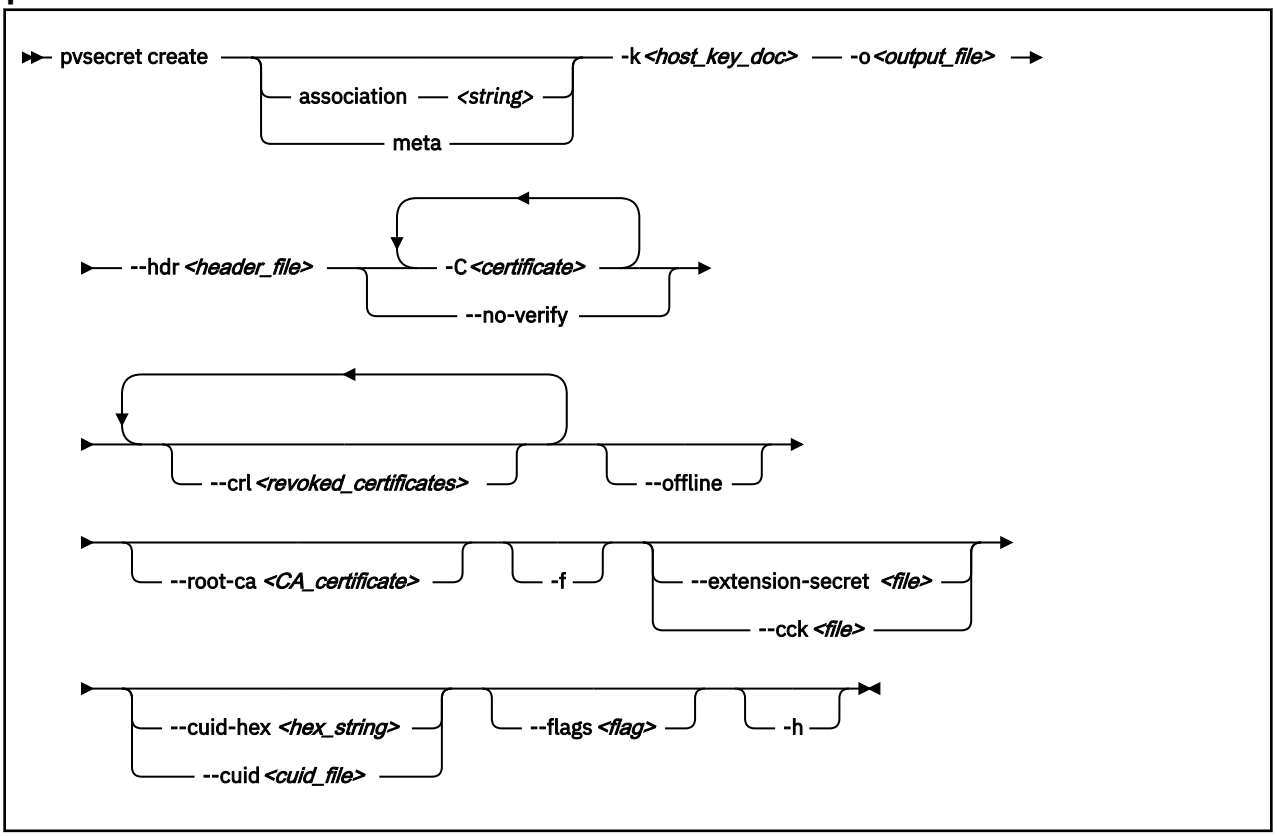
list
On a KVM guest running in secure execution mode, lists the secrets in the store of secrets, see “pvsecret list” on page 72 for details.

-h or --help
Optional: The short form, **-h**, displays short information about command usage. Specify after the main command for general help and after a sub-command for help specific to that command. The long form, **--help** gives more information.

-v or --verbose
Optional: displays verbose messages.

--version
Optional: displays version information.

pvsecret create



where:

association
Use an association secret to connect a trusted I/O device to a guest. For more information about association secrets, see “pvapconfig - Implement an AP queue configuration ” on page 63.

meta
Use a meta secret to carry flags to the ultravisor without having to provide an actual secret value. Meta secrets do not appear in the list of secrets.

-k or --host-key-document *<host_key_doc>*
specifies the host key document.

--no-verify
Creates the request without verifying the host key document.

Warning: Working with an unverified host key document makes your KVM guest vulnerable to man-in-the-middle attacks.

-C *<certificate>* or --cert *<certificate>*
specifies the certificate that is used to establish a chain of trust for the verification of the host key documents. Specify this option twice to specify the IBM Z signing-key certificate and the intermediate CA certificate (signed by the root CA).

Ignored when `--no-verify` is specified.

--crl *<revoked_certificates>*
Optional: specifies a list of revoked certificates.

--offline
Optional: does not download certificate-revocation lists. Every certificate requires a list of revoked certificates. If you specify `--offline`, specify one `--crl` for every `-C`.

--root-ca=*<trusted_CA_certificate>*
Optional: specifies a trusted root CA to use instead of one of the root CAs that are installed on the system.

--hdr *<header_file>*
specifies the header of the KVM guest.

-f or --force
forces the generation of add-secret requests on IBM Secure Execution guests.

-o *<output_file>* or --output *<output_file>*
specifies the file that contains the created request.

--extension-secret *<ext_file>*
specifies the file that contains the extension secret. For more information about extension secrets, see “Preventing the misuse of add-secret requests” on page 22.

--cck *<cck_file>*
specifies the customer-communication key (CCK) to derive the extension secret.

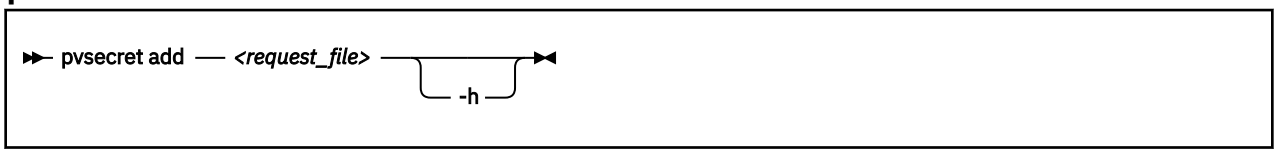
--cuid-hex *<hex_string>*
specifies the hex string to use as the Configuration Unique ID.

--cuid *<cuid_file>*
uses the content of *<file>* as the Configuration Unique ID

--flags *<flag>*
specifies flags for the add-secret request. Possible flag value: `disable-dump`.

-h or --help
Optional: The short form, **-h**, displays short information about the subcommand. The long form, **--help** gives more information.

pvsecret add

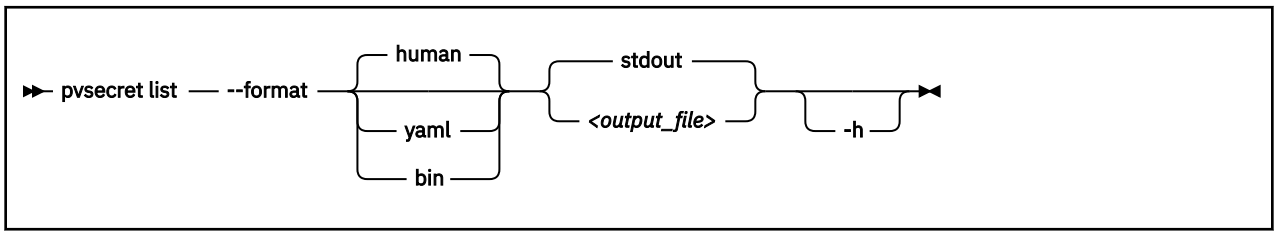


where:

<request_file>
contains the request with the secret to be added to the store. The secret can be a dummy secret.

-h or --help
Optional: The short form, **-h**, displays short information about the subcommand. The long form, **--help** gives more information.

pvsecret list



where:

--format *<format>*
defines the output format of the list. Valid values for *<format>* are:

human
Human-readable format (default).

yaml
YAML format.

bin
Binary format, as given by the ultravisor.

<output file>
Optional: writes the output to a file. Default is standard out.

-h or --help
Optional: The short form, **-h**, displays short information about the subcommand. The long form, **--help** gives more information.

Examples:

- To list the store of secrets, issue:

```
# pvsecret list
Total number of secrets: 2

0 Association:
 a63a6f8b796ec96304ae6b0c635986a3b5fac19b9ce7eac55978453e2f222fd5
1 Association:
 5468697320697320612076657279207365637265742073656372657420696 42e
```

The output shows that there are two secrets of type association with indexes 0 and 1.

- To list the secrets in YAML format, use the `yaml` option:

```
# pvsecret list --format yaml
```

pvsecret lock

where

-h or --help
Optional: The short form, **-h**, displays short information about the subcommand. The long form, **--help** gives more information.

Locks the store of secrets so that no more secrets can be added.

Example: On the guest for which you want to lock the store of secrets and prevent any further secrets from being added, issue:

```
# pvsecret lock
```

pvsecret

Appendix B. Boot configurations

By default, **zipl** processes the default configuration in the default configuration file `/etc/zipl.conf`.

A generic zipl configuration file

```
# vi zipl.conf
...
[secure]
target=/boot
image=/boot/secure-linux
...
```

Red Hat Enterprise Linux BLS configuration

You can use the `zipl.conf` configuration file or BLS snippets to configure the booting of a Red Hat Enterprise Linux guest. A sample BLS snippet could look as follows:

```
title Red Hat Enterprise Linux (secured)
version 5.5.0-10-bls-test
linux /boot/secure-linux
```

Assuming that this zipl configuration-file and the BLS snippet are both at their default locations, the following command processes the BLS snippet:

```
# zipl -V
```

SUSE Linux Enterprise Server GRUB2 configuration

This example assumes that the `/boot` filesystem resides on a BRTFS volume.

1. Append the following text to `/etc/grub.d/40_custom`:

   ```
   menuentry "Secure execution image" {
       linux ${btrfs_subvol}/boot/secure-linux
   }
   ```

2. Run the following command:

   ```
   # grub2-mkconfig -o /boot/grub2/grub.cfg
   ```

For more information on GRUB2, see the SUSE Linux *Administration Guide*, available at:

```
https://documentation.suse.com/sles
```

Ubuntu Server zipl configuration

The following sample zipl.conf file shows a setup for an Ubuntu Linux instance:

```
[ubuntu]
target=/boot
image=/boot/secure-linux
```

Assuming that this zipl configuration file is at its default location, the following command processes the Ubuntu definition:

```
# zipl -V
```

Appendix C. Obtaining a host key document from Resource Link

You can download a host key document from Resource Link, if the IBM Secure Execution feature is enabled on your IBM Z or LinuxONE.

Before you begin

If you have never signed in to Resource Link, you need to register before you can access the host key document page.

1. Open the main page, www.ibm.com/servers/resourcelink
2. Click **Sign in**. You are prompted to register.

You will receive an email in about an hour when the registration is complete.

Procedure

1. As a registered user, access the **Host key documents** page:

   ```
   https://www.ibm.com/support/resourcelink/api/content/public/host-key-documents.html
   ```

2. Follow the instructions on the page for requesting a host key document.

Appendix D. Verifying the host key document

To ensure that the host key document is genuine and provided by IBM, you need to verify the document manually.

Before you begin

To verify the host key document you need:

- The host key document that you received from your cloud provider, HKD-*<mmmm-nnnn>*.crt, or downloaded from Resource Link, see Appendix C, "Obtaining a host key document from Resource Link," on page 77.
- The DigiCert CA certificate, DigiCertCA.crt
- The IBM Z signing-key certificate, ibm-z-host-key-signing.crt
- The certificate revocation list (CRL), ibm-z-host-key.crl

You can download the CA certificate, the signing-key certificate, and the CRL from IBM Resource Link:

- For IBM z15:

  https://www.ibm.com/support/resourcelink/api/content/public/secure-execution-gen1.html

- For IBM z16:

  https://www.ibm.com/support/resourcelink/api/content/public/secure-execution-gen2.html

Check the **genprotimg** man for the latest updates to the verification procedure.

About this task

The procedure assumes an Internet connection. The commands download CRLs. For information about working offline, see the OpenSSL documentation.

Tip: Use the sample script available from s390-tools to perform the verification steps:

https://github.com/ibm-s390-tools/s390-tools/tree/master/genprotimg/samples/check_hostkeydoc

Procedure

1. Verify the CA certificate with the following command:

   ```
   # openssl verify -crl_download -crl_check DigicertCA.crt
   ```

2. Verify the IBM Z signing-key certificate with the following command:

   ```
   # openssl verify -crl_download -crl_check -untrusted DigicertCA.crt ibm-z-host-key-signing.crt
   ```

3. Verify the signature of the host key document:
   a) Extract the public signing key into a file.
   In this example the file is called pubkey.pem:

   ```
   # openssl x509 -in ibm-z-host-key-signing.crt -pubkey -noout > pubkey.pem
   ```

   b) Extract the host key signature from the host key document.
   The following command returns the offset value *<n>* of the signature:

   ```
   # openssl asn1parse -in HKD-<mmmm-nnnn>.crt | tail -1 | cut -d : -f 1
   ```

Use the resulting value *<n>* to extract the host key signature into a file called `signature`:

```
# openssl asn1parse -in HKD-<mmmm-nnnn>.crt -out signature -strparse <n> -noout
```

c) Extract the host key document body into a file called `body`:

```
# openssl asn1parse -in HKD-<mmmm-nnnn>.crt -out body -strparse 4 -noout
```

d) Verify the signature using the `signature` and `body` files:

```
# openssl sha512 -verify pubkey.pem -signature signature body
```

4. Verify the host key document issuer.

   Compare the outputs of the following two commands:

```
# openssl x509 -in HKD-<mmmm-nnnn>.crt -issuer -noout
```

```
# openssl x509 -in ibm-z-host-key-signing.crt -subject -noout
```

   The order of the arguments and options might differ, but it is important that all elements are present and their values are the same.

5. Verify that the host key document is still valid by checking the output of the following command:

```
# openssl x509 -in ibm-z-host-key-signing.crt -dates -noout
```

6. Verify that the host key has not been revoked.

   a) Verify the signature of the CRL file. Follow the steps described in “3” on page 79, but replace `HKD-<mmmm-nnnn>.crt` with `ibm-z-host-key.crl`.

   b) Verify the CRL issuer. Follow the steps described in “4” on page 80, but use the following command to find the CRL issuer:

```
# openssl crl -in ibm-z-host-key.crl -issuer -noout
```

   c) Verify that the revocation list is still valid by checking the output of the following command:

```
# openssl crl -in ibm-z-host-key.crl -lastupdate -nextupdate -noout
```

   d) Check whether the serial number of the host key is contained in the CRL.

   To find the serial number of all revoked host keys, use the following command:

```
# openssl crl -in ibm-z-host-key.crl -text -noout | grep "Serial Number"
```

   To obtain the serial number of the host key document, use the following command:

```
# openssl x509  -in HKD-<mmmm-nnnn>.crt  -serial -noout
```

   If the host key serial number is contained in the CRL, do not use this host key document.

Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml

Adobe is either a registered trademark or trademark of Adobe Systems Incorporated in the United States, and/or other countries.

The registered trademark Linux is used pursuant to a sublicense from the Linux Foundation, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis.

Red Hat® is a trademark or registered trademark of Red Hat, Inc. or its subsidiaries in the United States and other countries.

Index

L

M

O

P

Q

R

S

T

U

V

W

SC34-7721-04