

# Crypto Express for Cloud Workloads

Lydia Parziale

Marco Egli

Harald Freudenberger

Savitri Hunasheekatti

Sandor Irmes



 **Security**

**IBM Z**





IBM Redbooks

**Crypto Express for Cloud Workloads**

September 2024

**Note:** Before using this information and the product it supports, read the information in “Notices” on page v.

**First Edition (September 2024)**

This edition applies to the Crypto Express 8S (CEX8S) coprocessor, IBM z16, IBM LinuxOne 4(GA1.5).

**© Copyright International Business Machines Corporation 2024. All rights reserved.**

Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

# Contents

<b>Notices</b> .....	v
Trademarks .....	vi
<b>Preface</b> .....	vii
Authors .....	vii
Now you can become a published author, too! .....	viii
Comments welcome .....	viii
Stay connected to IBM Redbooks .....	ix
<b>Chapter 1. Introduction</b> .....	1
1.1 Cryptographic components .....	2
1.1.1 CPACF .....	2
1.1.2 Crypto Express cards .....	2
1.1.3 Trusted Key Entry workstation .....	3
1.2 Cryptographic terms .....	4
1.2.1 Clear key versus secure key versus protected key .....	4
1.3 Cryptographic concepts .....	6
<b>Chapter 2. Overview of our environment</b> .....	11
2.1 Lab environment .....	12
2.2 Crypto Express configuration .....	13
2.2.1 Card configuration .....	14
2.2.2 LPAR configuration .....	18
2.2.3 Dynamic Partition Manager .....	20
2.3 Master key setup .....	24
2.3.1 Trusted Key Entry .....	24
2.3.2 Checking the master key setup .....	24
2.3.3 Master key setup with TKE .....	28
2.3.4 Control domains .....	29
<b>Chapter 3. Configure LINUX guests to use CEX adapters</b> .....	31
3.1 Configuring a KVM host to provide CEX functionality .....	32
3.1.1 Setting up the KVM host machine to use CEX functions .....	32
3.1.2 What you should know about VFIO .....	32
3.1.3 Checking kernel modules .....	35
3.1.4 Configuring VFIO AP queues .....	37
3.1.5 Configuring the mediated device - KVM guest level .....	44
3.1.6 Managing VFIO AP mediated devices with libvirt .....	47
3.2 Configuring z/VM guests to use CEX adapters .....	53
3.2.1 Setting up the z/VM host machine to use CEX functions .....	53
3.2.2 Assigning crypto resources on z/VM systems .....	55
3.2.3 IBM Z operational keys: Clear, protected, or secure .....	59
3.3 Setup and configure Linux guests to use crypto resources .....	60
3.3.1 Dynamic assignment of crypto resources to z/VM guests .....	60
3.3.2 Crypto resource assignment to z/VM guests for dedicated use .....	62
3.3.3 Removing dedicated crypto resources from z/VM guests .....	64
3.3.4 Persistence across z/VM host or guest reboots .....	66
<b>Chapter 4. Using a CEX resource within a containerized environment</b> .....	81

4.1 CEX resource deployment in a Docker environment . . . . .	82
4.1.1 Installation and simple usage examples for Podman . . . . .	82
4.1.2 Simple deployment of CEX resources . . . . .	84
4.1.3 A more sophisticated CEX deployment. . . . .	86
4.2 CEX deployment configuration in Kubernetes and Red Hat OpenShift Container Platform . . . . .	89
4.2.1 Kubernetes on a Red Hat OpenShift cluster. . . . .	90
4.2.2 CEX resources in Kubernetes orchestrated containers . . . . .	91
<b>Chapter 5. Guest and workload considerations for using an HSM in the cloud . . . .</b>	<b>107</b>
5.1 Determining the right HSM . . . . .	108
5.2 openCryptoki . . . . .	110
5.2.1 Slots and tokens . . . . .	110
5.2.2 Installation of openCryptoki. . . . .	111
5.2.3 Configuration of openCryptoki . . . . .	111
5.2.4 Managing tokens. . . . .	113
5.2.5 Generating and listing keys. . . . .	114
5.2.6 Token specifications . . . . .	115
5.3 dm-crypt . . . . .	116
5.3.1 Installation and configuration overview. . . . .	117
5.4 Crypto Express support for Secure Execution . . . . .	117
5.4.1 Terms and concepts related to secure execution with CEX support. . . . .	118
5.4.2 Secret preparation for SE guests with CEX support . . . . .	119
5.4.3 KVM host setup for SE guests with CEX support . . . . .	120
5.4.4 KVM guest setup for SE guests with CEX support . . . . .	120
5.4.5 Security Details . . . . .	124
5.4.6 Redundancy . . . . .	124
5.4.7 Protecting AP association secrets. . . . .	124
5.4.8 Important considerations for the secure use of Crypto Express adapters in EP11 mode. . . . .	126
<b>Related publications . . . . .</b>	<b>129</b>
IBM Redbooks . . . . .	129
Other publications . . . . .	129
Online resources . . . . .	129
Help from IBM . . . . .	129

# Notices

This information was developed for products and services offered in the US. This material might be available from IBM in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

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 grant you any license to these patents. You can send license inquiries, in writing, to:

*IBM Director of Licensing, IBM Corporation, North Castle Drive, MD-NC119, Armonk, NY 10504-1785, US*

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 jurisdictions 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 websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you provide in any way it believes appropriate without incurring any obligation to you.

The performance data and client examples cited are presented for illustrative purposes only. Actual performance results may vary depending on specific configurations and operating conditions.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to actual people or business enterprises is entirely coincidental.


## COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

# Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation, 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 <https://www.ibm.com/legal/copytrade.shtml>

The following terms are trademarks or registered trademarks of International Business Machines Corporation, and might also be trademarks or registered trademarks in other countries.

AIX®	PIN®	z/OS®
IBM®	POWER®	z/VM®
IBM Z®	Redbooks®	z16™
IBM z16™	Redbooks (logo)  ®	

The following terms are trademarks of other companies:

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.

Java, and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Red Hat, OpenShift are trademarks or registered trademarks of Red Hat, Inc. or its subsidiaries in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.



# Preface

Highly sensitive workloads on Linux on IBM® Z and LinuxONE can use the premium protection of Crypto Express 8S (CEX8s) adapters in CCA or EP11 mode. Workloads can use CEX8S adapters as directly attached Hardware Security Modules (HSMs) at various levels of virtualization: in an LPAR, an IBM z/VM® or KVM guest, or in a Kubernetes container on Red Hat OpenShift.

The CEX8S hardware security module offers both classical and quantum-safe cryptographic technology to help address use cases that require information confidentiality, integrity and non-repudiation.

With IBM z16™ and LinuxONE 4 (GA 1.5) it is possible to securely attach a domain of a CEX8S adapter to a secure execution guest, allowing a tenant to run sensitive workloads with HSM access in a cloud environment, even if the tenant does not trust all levels of the cloud administration.

This IBM Redbooks® publication also explains how to connect a Trusted Key Entry system to IBM Z® or LinuxONE hardware to configure Crypto Express adapters. In particular, we address running a secure execution guest that uses a Crypto Express adapter.

Additionally, this publication will provide a high level end-to-end overview of how to set up cryptographic resources on all required levels, including hardware, hypervisor, cluster, and operating system or container such that it can run a crypto workload in the cloud and is intended for IT Architects, IT Specialists and system administrators

## Authors

This book was produced by a team of specialists from around the world working at IBM Redbooks, Poughkeepsie Center.

**Lydia Parziale** is a Project Leader for the IBM Redbooks team in Poughkeepsie, New York, with domestic and international experience in technology management including software development, project leadership, and strategic planning. Her areas of expertise include business development and database management technologies. Lydia is a PMI certified PMP and an IBM Certified IT Specialist with an MBA in Technology Management and has been employed by IBM for over 30 years in various technology areas.

**Marco Egli** is a Mainframe Engineer at Swiss Re, based in Switzerland. He has worked for more than 15 years in the mainframe area and has been responsible for software and hardware installations as well as the overall architecture of the mainframe environment. His focus has shifted over the past years from exploiting new technologies (such as zCX, Open Data Analytics and others) towards security and cryptography, in general, and especially on IBM Z.

**Harald Freudenberg** is a Diplom-Informatiker (computer scientist) in Germany. He has about 30 years of experience in different IT areas. He has been working at IBM since 2000 and has implemented network applications and worked as a developer for within the Linux kernel with embedded controllers. The last 8 years he has been working in the Linux on IBM Z Crypto Team and is responsible for the Crypto Card device driver and other crypto-related applications within the Linux kernel.

In 2021, he began to write a first implementation of the CEX-Device-Plug-in for Kubernetes, which makes crypto resources available to containers running in the cloud on the IBM Z platform.

**Savitri Hunasheekatti** is a Software Architect for IBM Hyper Protect at IBM India Systems Development Lab, Bangalore. She received her Bachelor of Engineering degree at BEC Engineering College, Bagalkote. She has over 20 years of experience in software development and joined IBM in 2003. In her recent roles, she has worked on various development projects across IBM POWER® and IBM Z. She has worked on projects related to AIX on POWER kernel development and LinuxOne BareMetal on IBM Z. She is a Plateau holder. She spends time submitting ideas, participating in hackathons, and reading about the latest technologies.

**Irnes Sandor** is a senior IT architect in Hungary who provides Linux on IBM Z and IBM LinuxONE consulting services at EMEA IBM Z Lab Services. He has more than 30 years of experience in IBM POWER and mainframe server technology, and several years of experience in Linux on IBM Z and open source. His fields of specialization encompass hybrid cloud options, infrastructure, and platform services, along with networking and Linux-related capabilities. Sandor is an IBM-certified IT Architect and has worked for IBM for over 16 years in various technology areas.

Thanks to the following people for their contributions to this project:

Robert Haimowitz  
**IBM Redbooks, Poughkeepsie Center**

Reinhard Bündgen, Eric Rossman, Marc van der Meer  
**IBM**

## Now you can become a published author, too!

Here's an opportunity to spotlight your skills, grow your career, and become a published author—all at the same time! Join an IBM Redbooks residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

Find out more about the residency program, browse the residency index, and apply online at:

[ibm.com/redbooks/residencies.html](http://ibm.com/redbooks/residencies.html)

## Comments welcome

Your comments are important to us!

We want our books to be as helpful as possible. Send us your comments about this book or other IBM Redbooks publications in one of the following ways:

- ▶ Use the online **Contact us** review Redbooks form found at:

[ibm.com/redbooks](http://ibm.com/redbooks)

- ▶ Send your comments in an email to:

[redbooks@us.ibm.com](mailto:redbooks@us.ibm.com)

- ▶ Mail your comments to:

IBM Corporation, IBM Redbooks  
Dept. HYTD Mail Station P099  
2455 South Road  
Poughkeepsie, NY 12601-5400

## Stay connected to IBM Redbooks

- ▶ Find us on LinkedIn:

<https://www.linkedin.com/groups/2130806>

- ▶ Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:

<https://www.redbooks.ibm.com/subscribe>

- ▶ Stay current on recent Redbooks publications with RSS Feeds:

<https://www.redbooks.ibm.com/rss.html>





# Introduction

This chapter provides a brief introduction into the cryptographic components used in this book. Terminology that we use are explained, as well as the base concepts of cryptographic configuration.

This chapter introduces:

- ▶ “Cryptographic components” on page 2
- ▶ “Cryptographic terms” on page 4
- ▶ “Cryptographic concepts” on page 6

## 1.1 Cryptographic components

The IBM Z platform offers cryptographic engines that provide high-speed cryptographic operations. In this section, we discuss some of those components that make up the cryptographic engines.

### 1.1.1 CPACF

CP Assist for Cryptographic Functions (CPACF) is a set of instructions that is available on every processor unit that accelerates encryption. CPACF is designed to facilitate the privacy of cryptographic key material when used for data encryption through a key wrapping implementation. It ensures that key material is not visible to applications or operating systems during encryption operations.

### 1.1.2 Crypto Express cards

All details within this section are focused on Crypto Express 8S (CEX8S) features.

The CEX8S is built from the 4770 Hardware Security Module (HSM) that provides a Quantum-Safe Root of Trust, and APIs that are used to modernize existing applications as well as build new ones, leveraging quantum safe cryptographic algorithms. The CEX8S has three configurable modes:

- ▶ Accelerator
- ▶ Common Cryptographic Architecture (CCA)
- ▶ EP11

EP11 mode enhancements include quantum-safe algorithms in hybrid cryptography for secure channel negotiations between the CEX8S and the CPACF and Trust Key Entry (TKE).

#### **Modes of operation**

In this section, we outline three modes of operation in CEX8S that are designed to combine secrecy and authentication.

#### ***Accelerator (CEX8A)***

The accelerator (CEX8A) provides acceleration of public key and private key cryptographic operations that are used with Secure Socket Layer and Transport Layer Security (SSL/TLS) processing.

#### ***Common Cryptographic Architecture Coprocessor***

The Common Cryptographic Architecture (CCA) Coprocessor (CEX8C) provides clear key and secure key operations such as RSA, ECC, DES, 3DES, AES, hashes, MACs, financial services like PIN verification, key storage, and random numbers.

#### ***Public Key Cryptography Standards #11 Coprocessor***

The Public Key Cryptography Standards (PKCS) #11 (EP11) Coprocessor (CEX8P) implements an industry-standardized set of services that adheres to the PKCS #11 specification 2.20 and more recent amendments. This mode introduced the PKCS #11 secure key function. In Enterprise PKCS #11 (EP11), keys can be generated and securely wrapped under the EP11 Master Key. A TKE workstation is always required to support the administration of the CEX8S when it is configured in EP11 mode.

IBM Hyper Protect Virtual Servers employ an EP11 over gRPC (GREP11) container that enables API calls to cryptographic functions on the HSM from other applications or microservices.

## Special features for CEX8S

CEX8S has a number of new functions and features which we discuss in this section.

### *Orderable Features*

The CEX8S (2 HSMs) feature contains two PCIe adapters that can each be configured as a coprocessor supporting secure key transactions or as an accelerator for Secure Sockets Layer (SSL) operations.

The CEX8S (1 HSM) feature contains one PCIe adapter that can be configured as a coprocessor supporting secure key transactions or as an accelerator for Secure Sockets Layer (SSL) operations.

### *Extended support for Quantum Algorithms*

While CEX7S already supports a base set of quantum algorithms (Round 2), CEX8S picks up the results from further NIST certification to support the following enhancements:

- ▶ Embedded support for CRYSTALS-Dilithium Round 3 and CRYSTALS-Kyber Round 2 quantum-safe algorithms
- ▶ Embedded support for CRYSTALS-Kyber Round 2 quantum-safe algorithms:
- ▶ Different strength levels
- ▶ CCA
- ▶ EP11

Those algorithms are available in both coprocessor modes (CCA and EP11).

To exploit these enhanced quantum algorithms within the Linux operating system<sup>1</sup> with openCryptoki, you need at least openCryptoki version 3.20 together with the EP11 host support program 4.0 (for coprocessor in EP11 mode) or CCA library version 8.0 (for coprocessor in CCA mode).

## 1.1.3 Trusted Key Entry workstation

The TKE workstation is an optional feature that offers key management functions. The TKE provides a secure, remote, and flexible method of providing Master Key Part Entry and to remotely manage PCIe cryptographic coprocessors. The cryptographic functions on the TKE are run by one PCIe cryptographic coprocessor. The TKE workstation communicates with the IBM Z or LinuxONE system through a TCP/IP connection. TKE securely manages multiple cryptographic modules that run in CCA or EP11 and use compliant-level hardware-based key management techniques from a single point of control.

Connection to the TKE requires an active component on the receiving side, either a TKE daemon under Linux or the CSFTTCP task in IBM z/OS®.

With the installation of the CCA support program library (csulcca<sup>2</sup>) the daemon will automatically be installed and started same applies for EP1 where the program library is ep11-host<sup>3</sup>.

---

<sup>1</sup> [CEX8S Linux on IBM Z](#)

<sup>2</sup> [Initial system set-up tips](#)

<sup>3</sup> [Installing the host part of the EP11 library](#)

## 1.2 Cryptographic terms

The following are some of the cryptographic terms used in this IBM Redbooks publication.

**CIPHER key** A type of data-encrypting key. CIPHER keys can be limited to more specific uses than DATA keys, preventing their use outside of data set encryption. CIPHER keys require CEX6S or higher and ICSF FMID HCR77C1 or higher. The key must allow encryption and decryption, any cipher mode, and export to CPACF protected key format.

**Key-encrypting key** A key that encrypts or wraps other keys.

**Master key** A special key-encrypting key (KEK) that is in a tamper-responding, Crypto Express adapter only and sits at the top level of a KEK hierarchy which is only available in an HSM. In EP11 mode, *wrapping key* is often used as a synonym for master key.

### CPACF wrapping

**key** A special key-encrypting key that is generated at logical partition (LPAR) activation and is in the Hardware System Area (HSA), which is inaccessible to applications and the operating system. Each Guest (LPAR, KVM guests, [...]) gets its own wrapping key.

**Secure key** A data-encrypting key that is encrypted by a master key or key-encrypting key and never appears in clear text that is outside of a secure environment, such as a tamper-responding HSM, or IBM Z firmware. Secure keys can be stored in an ICSF key data set or returned to the ICSF caller (only in IBM z/OS).

**Clear key** A data-encrypting key that is not encrypted by any other key. The key material is in clear text. Clear keys can be stored in an ICSF key data set or returned to the ICSF caller at key creation (only in z/OS). Most open source libraries only deal with clear keys like the popular library openssl.

**Protected key** A data-encrypting key that is encrypted by the CPACF wrapping key and used within the Z platform (HSA). Although protected keys are cached in ICSF, they are not persistently stored in an ICSF key data set. Protected keys can be returned to authorized ICSF callers (only in z/OS). Protected keys are only usable in the context of the HSA area specifically used by the CPACF wrapping key. This makes the key ephemeral and unusable to other guests.

**Operational key** A key that is not a master key, such as a data-encrypting key, which can be clear, secure, or protected.

### Hardware Security

**Module** IBM Crypto Express adapters are tamper-responding Hardware Security Modules (HSMs) that support cryptographic operations managing, processing and storing cryptographic keys within a FIPS 140-2 Level 3 validated environment.

### 1.2.1 Clear key versus secure key versus protected key

A clear key has not been encrypted under another key and has no additional protection within the cryptographic environment. For clear keys, the security of the keys is provided by operational procedures. Crypto operations may be performed in CPACF or on a Crypto Express adapter.



A secure key is protected by another key that is called a master key. IBM secure key hardware (the Crypto Express adapter) provides a tamper-sensing and tamper-responding environment that, when attacked, zeroizes the hardware and prevents the key values from being compromised. The secure key hardware requires that a master key is loaded. That master key is stored inside the secure hardware and used to protect operational keys.

The clear value of a secure key is generated inside the hardware (through a random number generator function), and encrypted under the master key. When a secure key must leave the secure hardware boundary (to be stored in a data set), that key is encrypted under the master key.

So, the encrypted value is stored, and not the clear value of the key. Some time later, when data must be recovered (decrypted), the secure key value is loaded back into the secure hardware. It is then decrypted from under the master key. The original key value is then used, inside the secure hardware, to decrypt the data.

The following are the secure key functions:

- ▶ FIPS compliance

The United States government has introduced a set of standards that define cryptographic algorithms and procedures to be used by government agencies and companies that work for the US government. These standards are part of FIPS and include, for example, AES and DES encryption algorithms.

The Crypto Express adapters for IBM Z are designed and certified to work in a standard-compliant mode, freeing application programmers from dealing with the intricacies of these standards. In secure key mode, you do not have to configure the adapter specifically for FIPS compliance.

- ▶ RSA public and private key processing

Running in clear key mode, the Crypto Express adapter supports RSA encryption and decryption with key lengths of up to 2048 bits. In secure mode, this function is extended to key lengths of up to 4096 bits.

- ▶ DES and triple DES

DES and triple DES are common shared key encryption algorithms that are used in many applications. Examples include smart cards, SSL communication, and disk encryption. The Crypto Express adapters in coprocessor mode provide an implementation of these algorithms, just as CPACF does. The difference lays in the asynchronous processing that is performed with the Crypto Express adapter.

Although using CPACF blocks your physical unit (PU) from any other work until the cryptographic operation is completed, Crypto Express works asynchronously. Requests are queued and pushed to the adapter where they are processed while the PU is free to run other code. Depending on your setup, the z90crypt driver then either polls the adapter for completed requests or is notified by the adapter itself, and the processed data is handed back to your application.

- ▶ MAC processing

In cryptography, a distinction exists between message confidentiality, which is provided by encryption, and message integrity, which can be provided either by signing the message or adding a message authentication code (MAC) to it.

Signatures use asymmetric keys to provide an advanced level of integrity and non-repudiability (ensuring the sender cannot deny sending a message), but MACs use symmetric keys and provide only basic integrity verification.

In secure key mode, the Crypto Express adapter provides MAC functions, which enable you to offload both MAC creation and MAC verification to the coprocessor. The IBM CCA RPM package contains sample source code in `/opt/IBM/4764/samples/mac.c` for computing a MAC by using the Crypto Express adapter.

From the perspective of the actual cryptographic operation, no difference exists between clear key and secure key operations. A piece of original plaintext, encrypted by using the same algorithm and the same key, can produce the same ciphertext whether the key was clear or secure.

The difference between clear key and secure key is simply in the management of the keys. With clear key, the key is protected only by file system permissions. A sufficiently privileged user can locate where the key is stored and read it. A secure key is encrypted by using a different key (the master key) and is never visible in the clear outside the secure cryptographic hardware.

IBM Z hardware adds support for protected keys. Protected keys blend the security of the Crypto Express hardware and the performance characteristics of the CPACF. An enhancement to CPACF facilitates the continued privacy of cryptographic key material when used for data encryption. CPACF, by using key wrapping, ensures that key material is not visible to applications or operating systems during encryption operations. Keys that are protected under the DES or AES master key are stored in a VSAM data set that is called the cryptographic key data set (CKDS). Only protected keys that are created from secure keys should be used when using pervasive encryption.

Integrated Cryptographic Service Facility (ICSF) can reencipher the secure key to decrypt it from under the original master key and reencrypt it under the new master key, all within the secure hardware and before it is stored back into a new CKDS, which is now associated with the new master key value.

IBM hardware that supports secure key operation provides protection for secure keys by employing tamper-sensitive storage that zeros the memory of the device if it is attacked. This protects the keys even when they are being used inside the hardware. The hardware can even support the changing of the master key by decrypting the secure key and re-encrypting it by using the new master key within the secure cryptographic hardware.

## 1.3 Cryptographic concepts

An LPAR, regardless if running z/OS, KVM or z/VM, has a 1:n connection to a Hardware Security Module (HSM) of a Crypto Express card. This connection is known as a *domain*. Assigning more than one domain to an LPAR makes sense when there are plans to run special loads that are able to exploit multiple domains (for example, kvm host, multi-tenant opencryptoki applications, docker).

Crypto Cards are subdivided into independent (virtual) crypto units, or domains. A CEX8S provides up to 85 domains (0-84).

- ▶ A maximum of 30 CEX8S (2 HSMs) can be installed. Resulting in effectively available HSMs of  $2 \times 30 \times 85 = 5100$
- ▶ A maximum of 16 CEX8S (1 HSM) can be installed. Resulting in effectively available HSMs of  $16 \times 85 = 1360$ .

Each domain of a crypto card is an independent HSM with its own set of Master Keys, defined roles, and control point settings.

Crypto cards are attached to an extra adjunct processor (AP) bus. With AP instructions, requests can be queued into an IBM Z firmware *queue*. Each millicode queue corresponds to one crypto unit and is called an *APQN*. The system can access each APQN, as shown in Figure 1-1.

		Card →				
		0	1	2	...	30
Domain ↓	0	(0,0)	(1,0)	(2,0)	...	(30,0)
	1	(0,1)	(1,1)	(2,1)	...	(30,1)
	2	(0,2)	(1,2)	(2,2)	...	(30,2)
	...	...	...	...	...	...
	84	(0,84)	(1,84)	(2,84)	...	(30,84)

Figure 1-1 Cards and Domains

An example of an LPAR that may be used to run a KVM or z/VM host is shown in Figure 1-2 on page 8. Four cards and four domains are assigned.

		Card →									
		0	1	2	3	4	5	6	...	15	
Domain ↓	0	(0,0)	(1,0)	(2,0)	(3,0)	(4,0)	(5,0)	(6,0)	...	(15,0)	
	1	(0,1)	(1,1)	(2,1)	(3,1)	(4,1)	(5,1)	(6,1)	...	(15,1)	
	2	(0,2)	(1,2)	(2,2)	(3,2)	(4,2)	(5,2)	(6,2)	...	(15,2)	
	3	(0,3)	(1,3)	(2,3)	(3,3)	(4,3)	(5,3)	(6,3)	...	(15,3)	
	4	(0,4)	(1,4)	(2,4)	(3,4)	(4,4)	(5,4)	(6,4)	...	(15,4)	
	5	(0,5)	(1,5)	(2,5)	(3,5)	(4,5)	(5,5)	(6,5)	...	(15,5)	
	6	(0,6)	(1,6)	(2,6)	(3,6)	(4,6)	(5,6)	(6,6)	...	(15,6)	
	...	...	...	...	...	...	...	...	...	...	
	84	(0,84)	(1,84)	(2,84)	(3,84)	(4,84)	(5,84)	(6,84)	...	(15,84)	

Figure 1-2 Card and Domain - sample

An example design point could be to have pairs of APQNs available for KVM guests:

guest 1: APQN (0,2) and (2,2)

guest 2: APQN (4,2) and (6,2)

guest 3: APQN (0,3) and (2,3)

The assignment of a crypto domain to an LPAR is defined in the Load Profile in the Crypto tab. An LPAR can either be assigned as “Control” or “Control and Usage”. Besides that there are two states of a card - “Candidate” and “Candidate and Online”. Changing the state of a crypto card can be done inside the operating system.

Changes of card or domain assignments requires edition the LPAR activation profile and requires a deactivation/activation of the LPAR to reflect the change.

The combination of usage domains and cards forms an APQN set and must be unique. There must not be an overlap of APQN sets. The Service Element (SE) checks if there is another LPAR with an overlapping APQN set and will refuse the activation in case of conflicting assignments.

Understanding the following key terms is crucial:

- Control** Only administrative commands can be sent to the card like setting master keys. Operational usage of the card to data de- and encryption is not possible.  
This option does allow a single LPAR to manage all crypto cards in the same CEC.
- Control and Usage** Administrative commands can be sent to the card and the card is operationally usable for data decryption and encryption.
- Candidate** The set of cryptographic coprocessors that the logical partition may access.

**Candidate and  
Online**

The set of cryptographic coprocessors that will be brought online when the logical partition is activated.

A detailed description of how to setup and configure crypto cards can be found in 2.2, “Crypto Express configuration” on page 13.





# Overview of our environment

In this chapter, we describe the following:

- ▶ 2.1, “Lab environment” on page 12
- ▶ 2.2, “Crypto Express configuration” on page 13
- ▶ 2.3, “Master key setup” on page 24

## 2.1 Lab environment

The lab environment used in this book consists of four Kernel-based Virtual Machines (KVMs), one IBM z/VM logical partition (LPAR) and four Crypto Express 8S (CEX8S) cards (with dual hardware security modules (HSMs)) hosted on an IBM Z16 A01 machine. Figure 2-1 provides a high-level abstraction of the environment. Color coding is used to emphasize the relationship between the guests and the card tied to the domains.

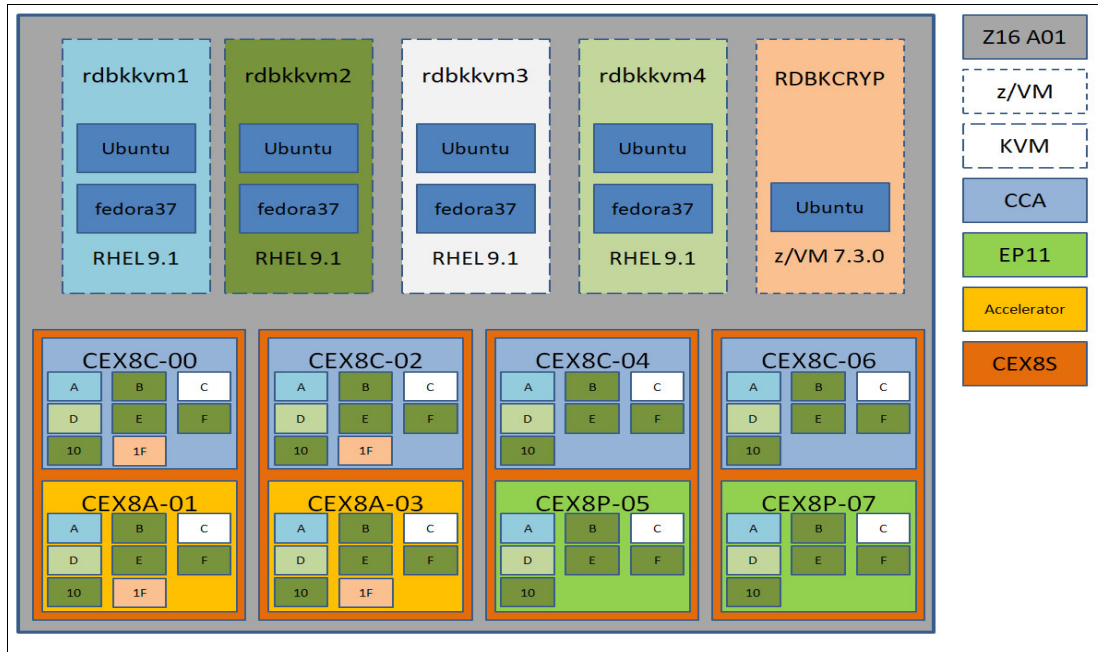


Figure 2-1 Architectural overview

The HSMs on the CEX8S cards each have a CCA processor configured and two cards. One HSM is configured as an accelerator, whereas the other two are configured for EP11.

Figure 2-2 outlines the matrix of card assignment.

	10	A	B	C	D	E	F	1F
CEX8C-00	KVM2	KVM1	KVM2	KVM3	KVM4	KVM2	KVM2	zVM
CEX8A-01	KVM2	KVM1	KVM2	KVM3	KVM4	KVM2	KVM2	zVM
CEX8C-02	KVM2	KVM1	KVM2	KVM3	KVM4	KVM2	KVM2	zVM
CEX8A-03	KVM2	KVM1	KVM2	KVM3	KVM4	KVM2	KVM2	zVM
CEX8C-04	KVM2	KVM1	KVM2	KVM3	KVM4	KVM2	KVM2	
CEX8P-05	KVM2	KVM1	KVM2	KVM3	KVM4	KVM2	KVM2	
CEX8C-06	KVM2	KVM1	KVM2	KVM3	KVM4	KVM2	KVM2	
CEX8P-07	KVM2	KVM1	KVM2	KVM3	KVM4	KVM2	KVM2	

Figure 2-2 Card assignment - matrix



**Note:** Depending on where the crypto card domain is shown, it will be either in hexadecimal or decimal only. Hardware Management Console (HMC)/Support Element (SE) shows the values in decimal and, as an example, 1szcrypt in hexadecimal.

Table 2-1 shows a mapping for the names used at different places in the subsequent chapters.

Table 2-1 System names mapping

Short	System	LPAR-Name
KVM1	rdbkkvm1	PAVO45
KVM2	rdbkkvm2	PAVO46
KVM3	rdbkkvm3	PAVO47
KVM4	rdbkkvm4	PAVO48
z/VM1	rdbkcryp	VELA28

Figure 2-3 displays the configuration of the cards as retrieved by the Cryptographic Configuration Task on the HMC.

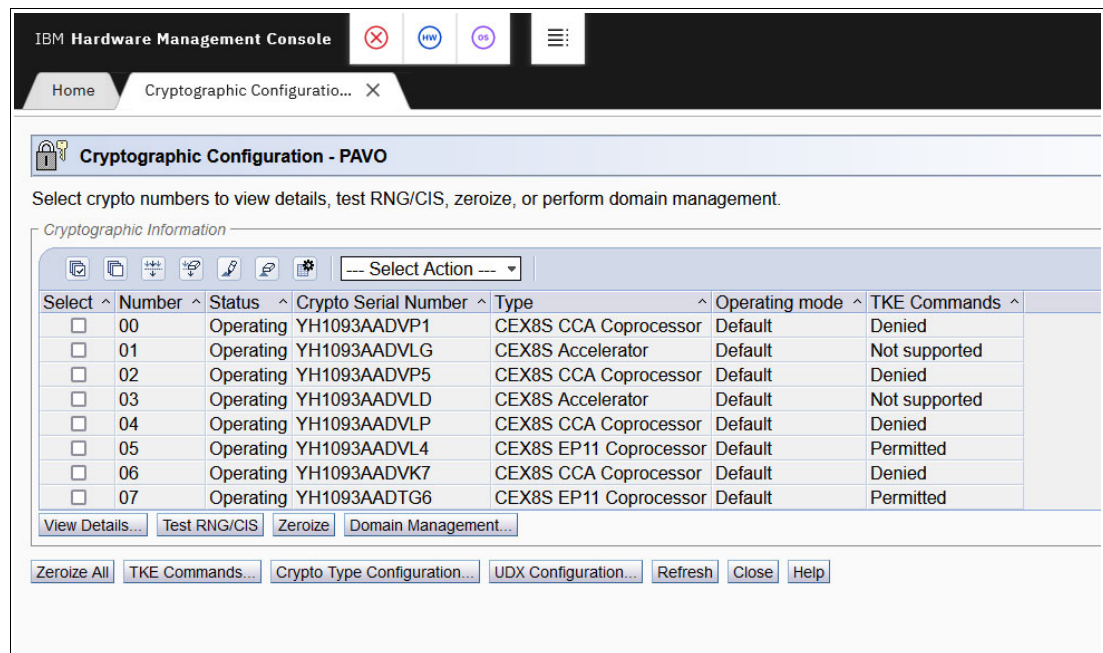


Figure 2-3 Cryptographic Configuration - HMC

## 2.2 Crypto Express configuration

In this section, we describe the steps to achieve our lab environment's configuration.

The steps outlined for the configuration of a Crypto Express card assumes that the Hardware Management Console HMC/SE is up and running and the authenticated user has the appropriate role assigned to perform the required activities.

## 2.2.1 Card configuration

When logged in to the HMC, select the System where the Configuration should be done, as shown in Figure 2-4. Select the **Cryptographic Configuration**, as highlighted.

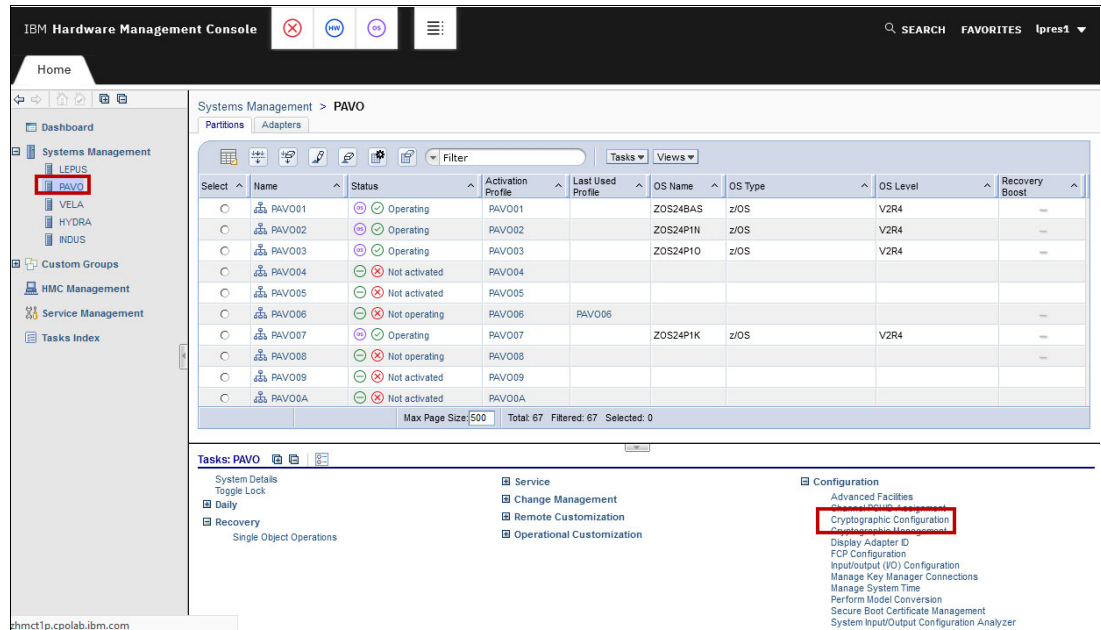


Figure 2-4 HMC entry screen with Cryptographic Configuration

This opens a new tab listing all available crypto cards, as shown in Figure 2-5. This tab lists the currently configured cards with the associated type of operation, as listed in the column named **Type**.

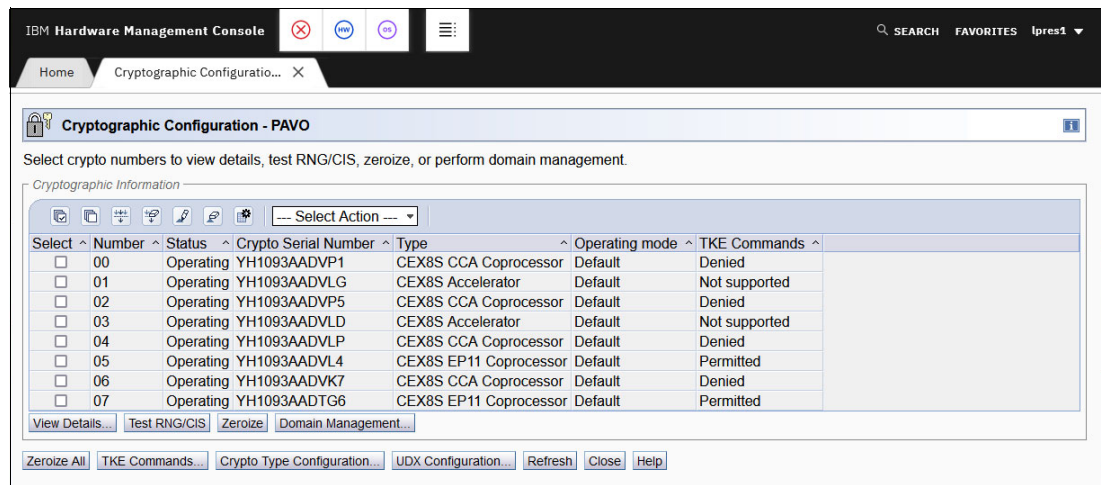


Figure 2-5 Cryptographic Configuration - Overview

If the type of a card is to be changed, click on **Crypto Type Configuration...**, as shown in Figure 2-6 on page 15.

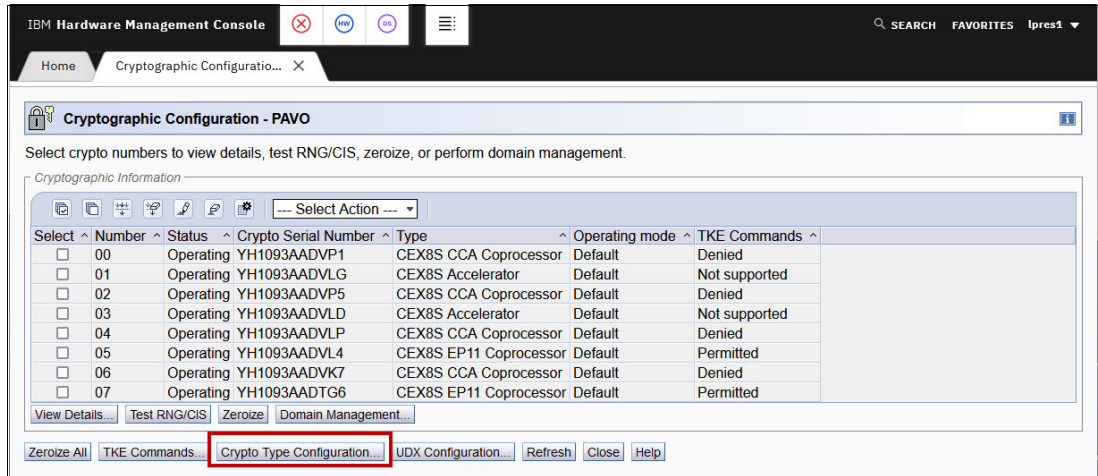


Figure 2-6 Cryptographic Configuration - Crypto Type Configuration

Once this activity is complete, a **Crypto Type Configuration - PAVO** tab appears. To change the type from a CEX8S EP11 Coprocessor to a CCA Coprocessor, select the card by clicking the checkbox in column **Select**. Next, choose the type and then click **Apply** at the bottom, as shown in Figure 2-7. A new window appears that asks for confirmation of the activity and provides notification that the card will be zeroized.

**Caution:** The cryptographic keys will be zeroized and the usage domains will operate in the default compliance mode when the cryptos are configured to an online state and the activity is confirmed.

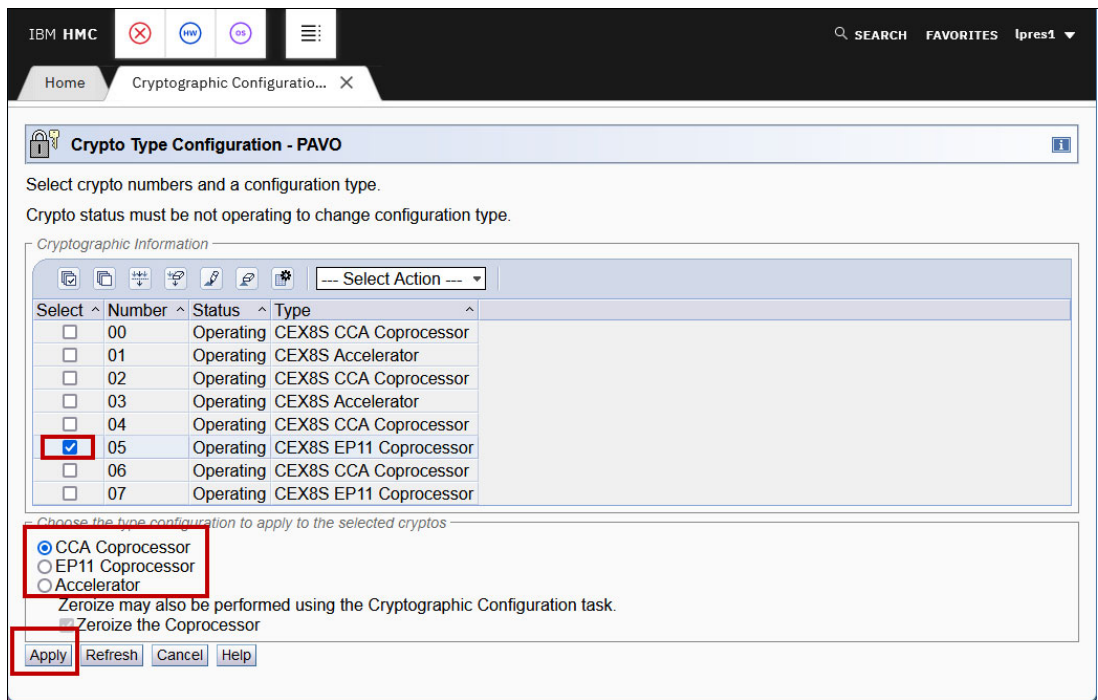


Figure 2-7 Cryptographic Configuration - Crypto Type Change

If the activity is successful, the overview will be updated to reflect the change type, as initially shown in Figure 2-5 on page 14. For this book's activity, the action was not executed as the cards were already in the desired state, hence no update is required.

Further details of a card can be displayed by checking the checkbox in column **Select** and then clicking **View Details....** The following figures show the three different types of cards. Figure 2-8 shows an Accelerator.

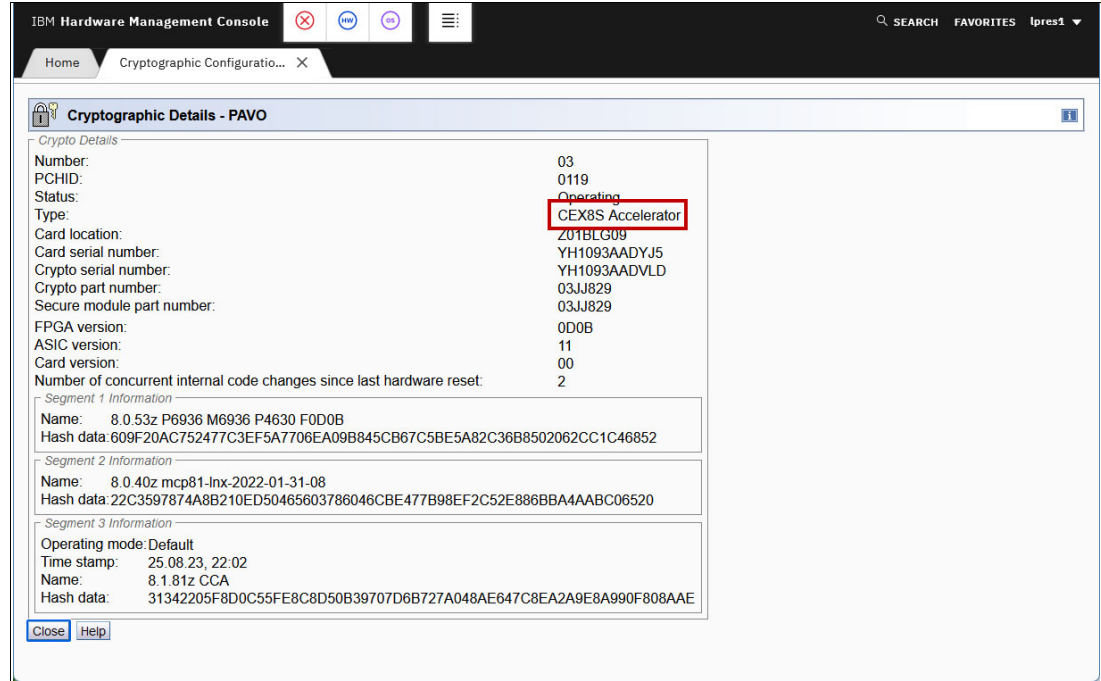


Figure 2-8 Cryptographic Configuration - Accelerator details

Figure 2-9 shows a CCA Coprocessor.

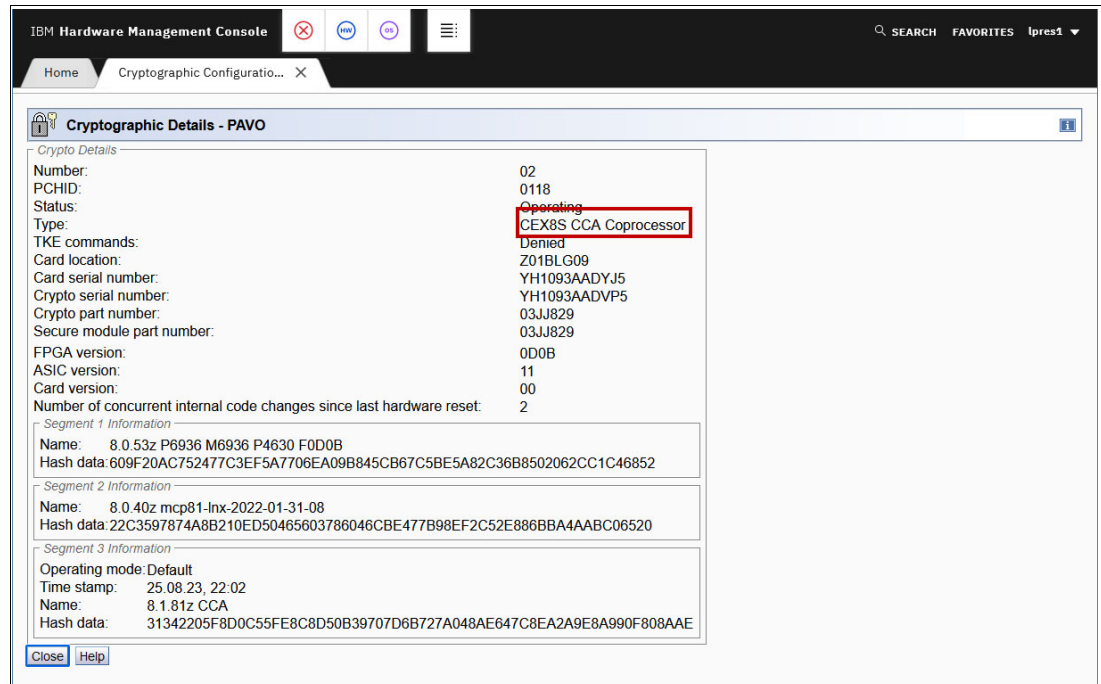


Figure 2-9 Cryptographic Configuration - CCA details

Figure 2-10 shows an EP11 Coprocessor.

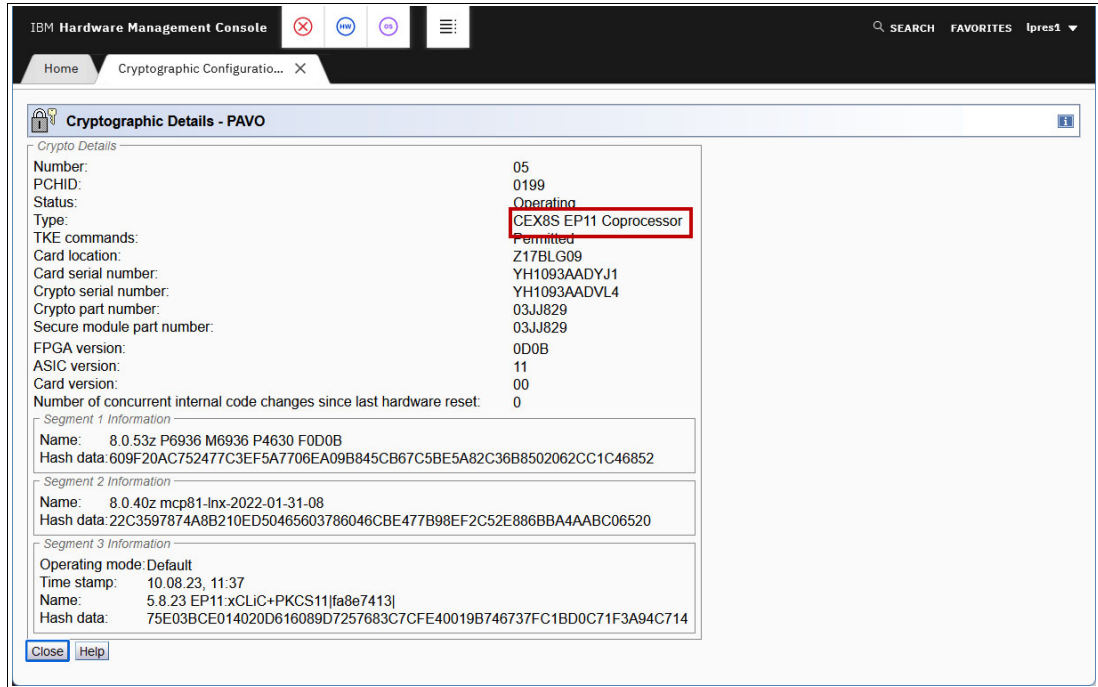


Figure 2-10 Cryptographic Configuration - EP11 details

All three card types (Accelerator, CCA and EP11) support the domain management option. When a card is selected and **Domain Management...** is clicked, a window, shown in Figure 2-11, appears. Select the domains that will be zeroized on the card (Domain 43 in our example) and confirm by clicking **Zeroize** at the bottom.

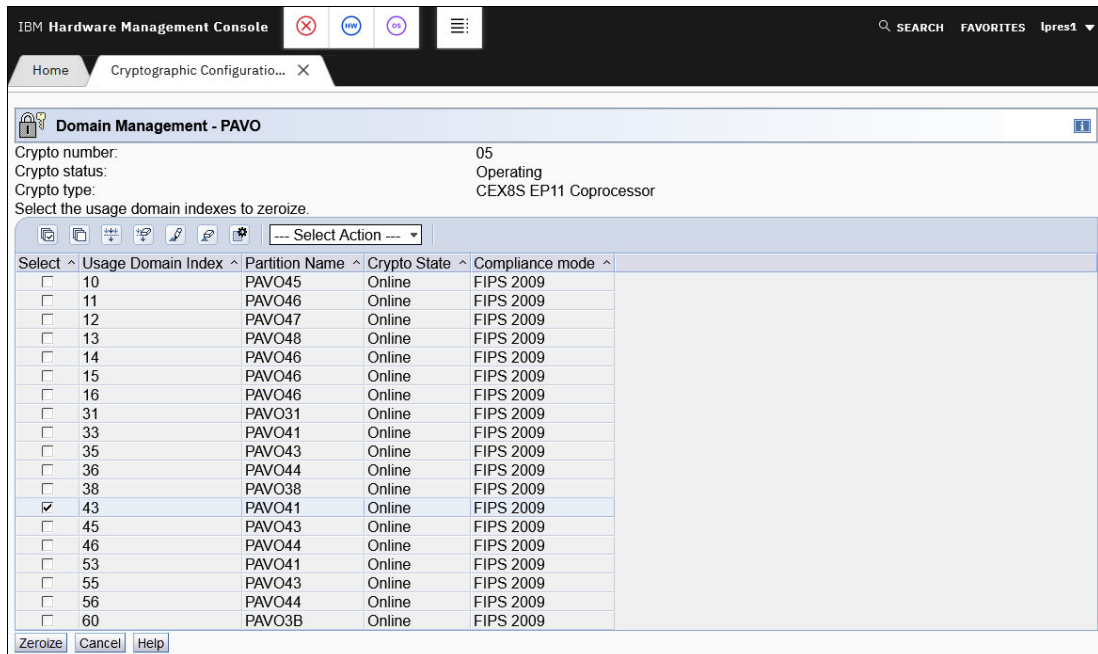


Figure 2-11 Cryptographic Configuration - Domain Management

## 2.2.2 LPAR configuration

2.2.1, “Card configuration” on page 14 outlined how the cards can be configured and introduced crypto domain assignments. The assignment of the domain is configured in the Load-Profile of the LPAR.

When logged into the HMC, select the system where the configuration should be done, as shown in Figure 2-12. Select the **Customize/Delete Activation Profiles** task as highlighted, from the **Operational Customization** task group.

The screenshot displays the IBM Hardware Management Console interface for system PAVO. The left-hand navigation pane shows the 'Systems Management' tree with 'PAVO' selected. The main area features a table of LPAR profiles with columns for Name, Status, Activation Profile, Last Used Profile, OS Name, OS Type, OS Level, and Recovery Boost. Below the table, a 'Tasks: PAVO' section is visible, containing several task groups. The 'Operational Customization' group has the 'Customize/Delete Activation Profiles' task highlighted with a red box.

Select	Name	Status	Activation Profile	Last Used Profile	OS Name	OS Type	OS Level	Recovery Boost
<input type="checkbox"/>	PAVO01	Operating	PAVO01		ZOS24BAS	z/OS	V2R4	--
<input type="checkbox"/>	PAVO02	Operating	PAVO02		ZOS24PIN	z/OS	V2R4	--
<input type="checkbox"/>	PAVO03	Operating	PAVO03		ZOS24PIO	z/OS	V2R4	--
<input type="checkbox"/>	PAVO04	Not activated	PAVO04					--
<input type="checkbox"/>	PAVO05	Not activated	PAVO05					--
<input type="checkbox"/>	PAVO06	Not operating	PAVO06	PAVO06				--
<input type="checkbox"/>	PAVO07	Operating	PAVO07		ZOS24PIK	z/OS	V2R4	--
<input type="checkbox"/>	PAVO08	Not operating	PAVO08					--
<input type="checkbox"/>	PAVO09	Not activated	PAVO09					--
<input type="checkbox"/>	PAVO0A	Not activated	PAVO0A					--
<input type="checkbox"/>	PAVO0B	Not activated	PAVO0B					--
<input type="checkbox"/>	PAVO0C	Not activated	PAVO0C					--
<input type="checkbox"/>	PAVO0D	Not activated	PAVO0D					--

Figure 2-12 LPAR Configuration - Activation Profiles

Once this activity is complete, a new tab, **Customize/Delete Activating Profiles: PAVO** appears. Select an LPAR Profile (RDBKVM1 in our example) and click on **Customize profile** to work with the profile, as shown in Figure 2-13 on page 19.

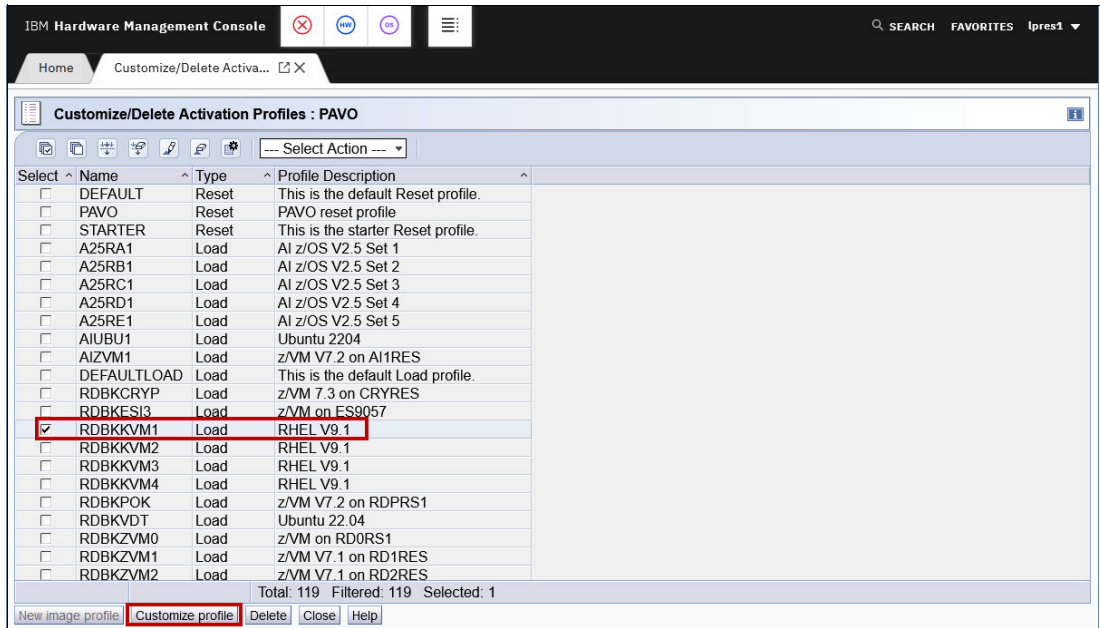


Figure 2-13 LPAR configuration - Customize profile

After clicking, customize the profile in the **General** section of the profile, as shown in Figure 2-14.

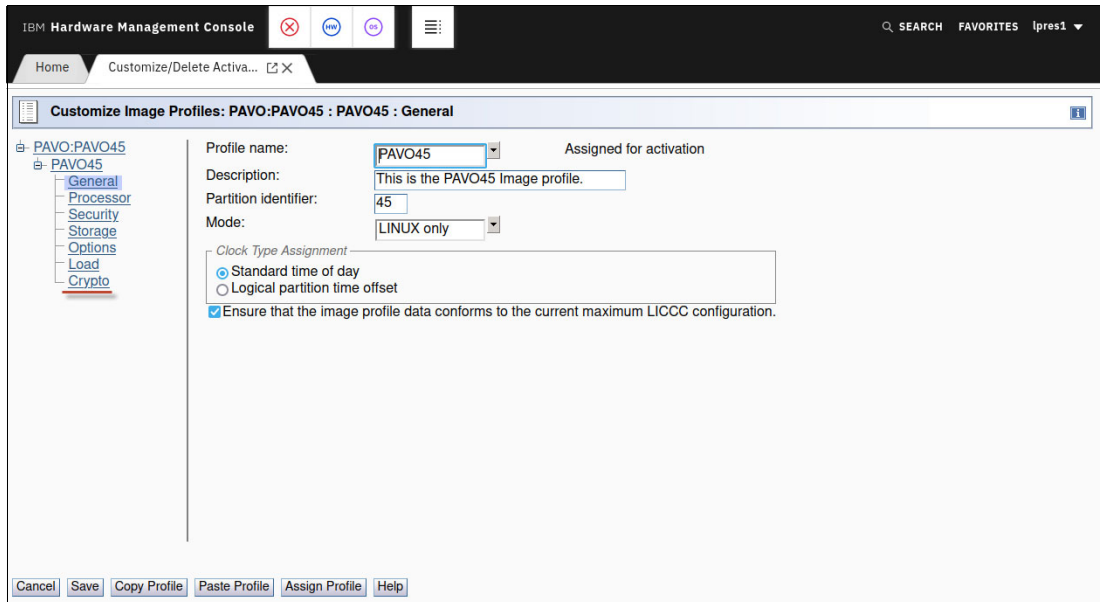


Figure 2-14 LPAR configuration - Customizes Profiles General

To work with the Crypto settings, select **Crypto** in the navigation tree on the left-hand side, as shown in Figure 2-15 on page 20.

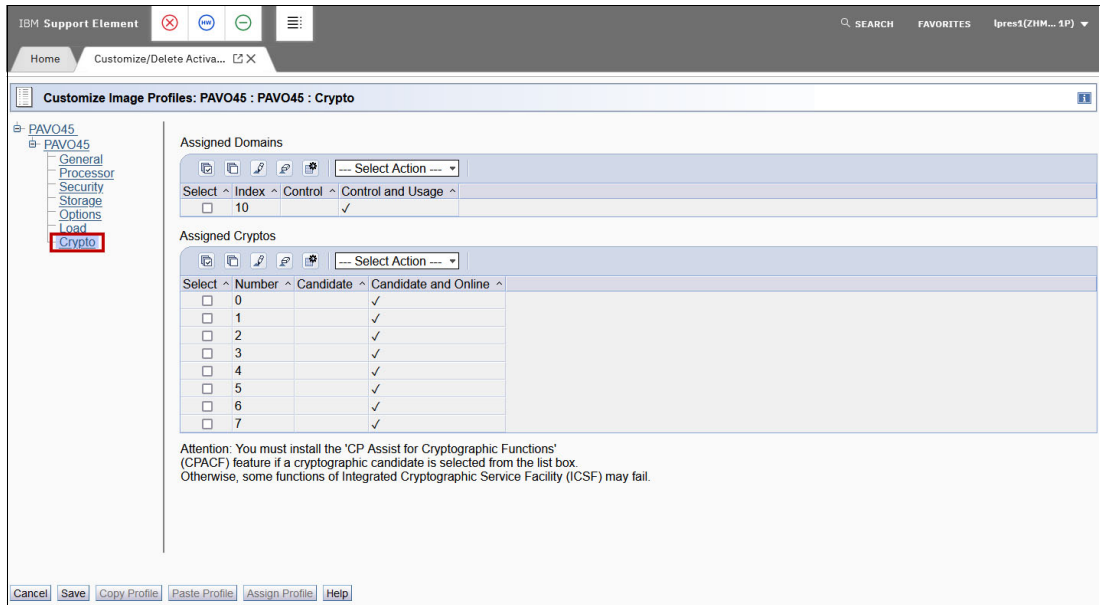


Figure 2-15 LPAR configuration - Customizes Profiles Crypto

When navigating to the **Crypto** category, PAV045 or RDBKKVM1 has the Index 10 assigned, which is in decimal format and would show as an “A” in hex-encoded outputs such as lszycrypt. The **Candidate** and **Online** checkmarks show that the LPAR in question has all eight cards available for use.

**Important:** To reflect changed Activation Profiles, the LPAR must be deactivated/activated.

## 2.2.3 Dynamic Partition Manager

The IBM Dynamic Partition Manager (DPM) is another operating mode for IBM Z machines. The following section outlines how to perform the same updates as described in 2.2.1, “Card configuration” on page 14 within DPM mode. For more information, see [Dynamic Partition Manager \(DPM\)](#).

**Note:** The preceding figures illustrate a different system than the one depicted in the previous chapter due to the unavailability of a DPM during the writing of this book. Consequently, the screenshots serve merely as visual aids for reference and illustrative purposes only.

As shown in Figure 2-16 on page 21, the system mode shows up as **Dynamic Partition Manger**.



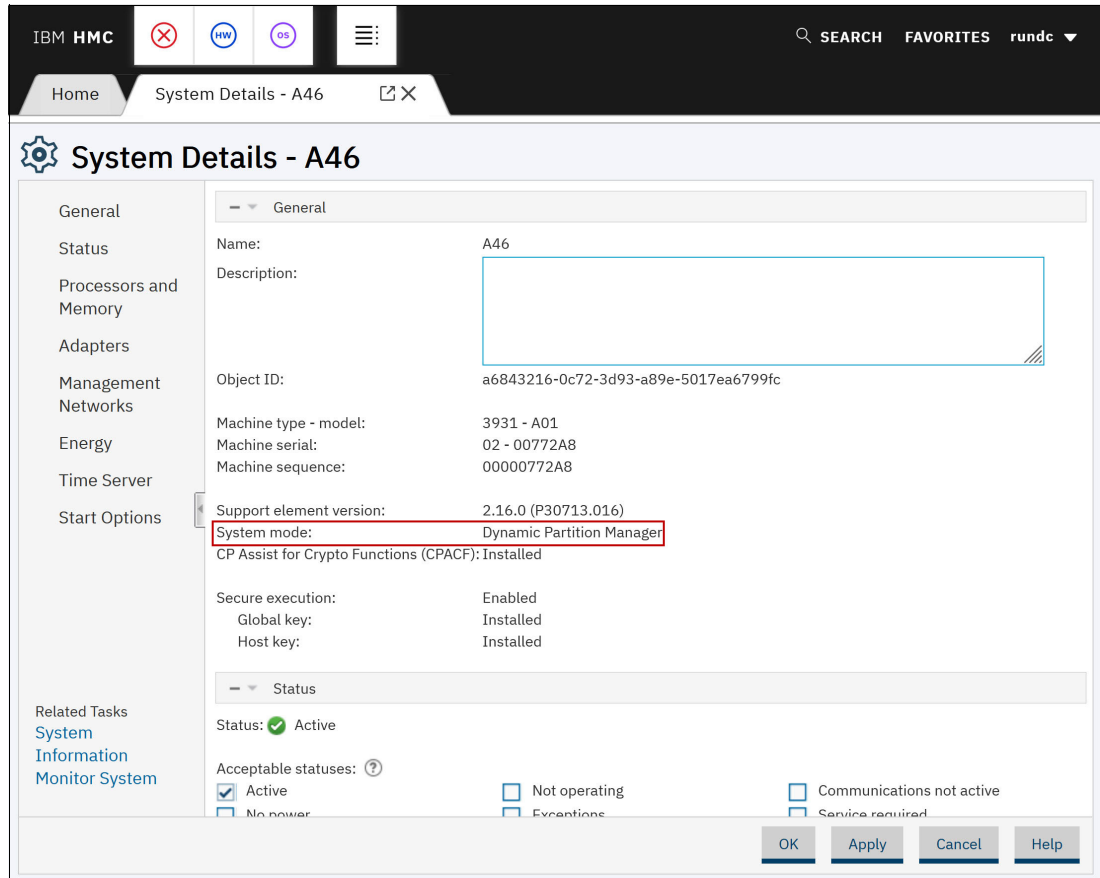


Figure 2-16 System Details

When working with partition details in DPM mode, all details are shown in different sections, as seen in Figure 2-17 on page 22.

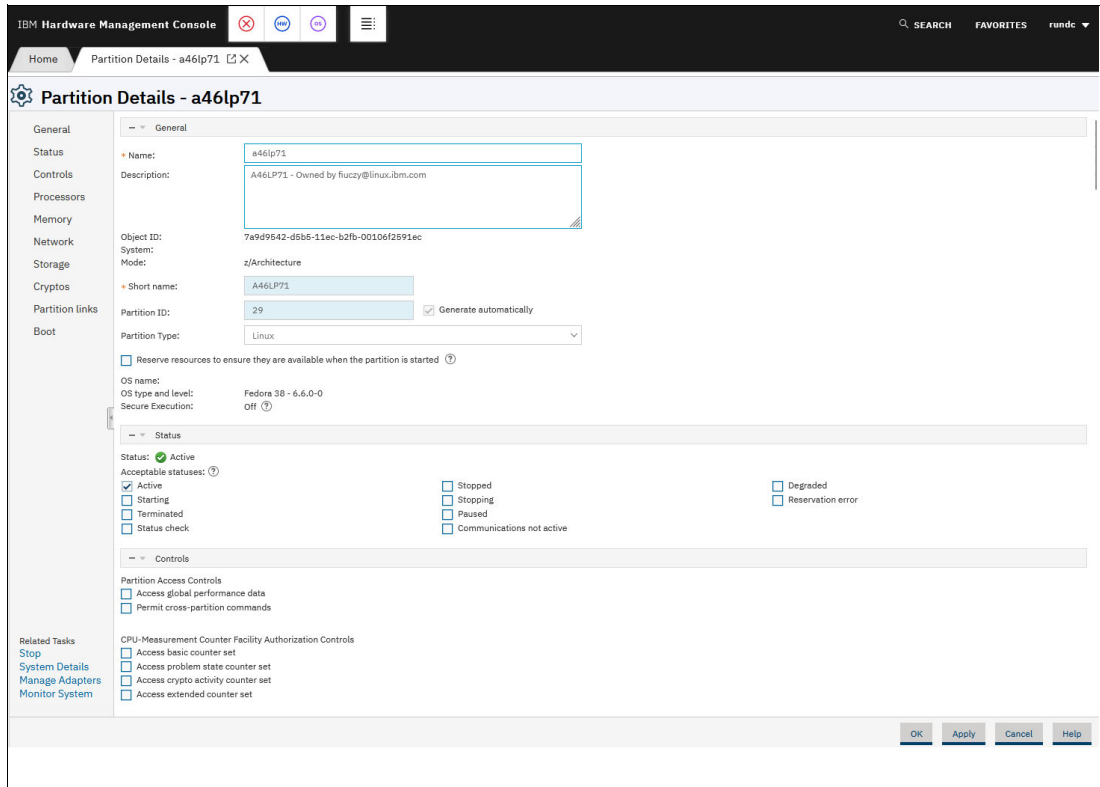


Figure 2-17 Partition Details - General

Figure 2-18 on page 23 shows the available cards in **Crypto Adapters**, which are two CCA coprocessors. The following section shows the **Available Domains** and the associated usage of the domain, either **Usage** or **Control & Usage**. In DPM mode, both modes are combined within one column and are differentiated by the icon shown, which is different than non-DPM mode.

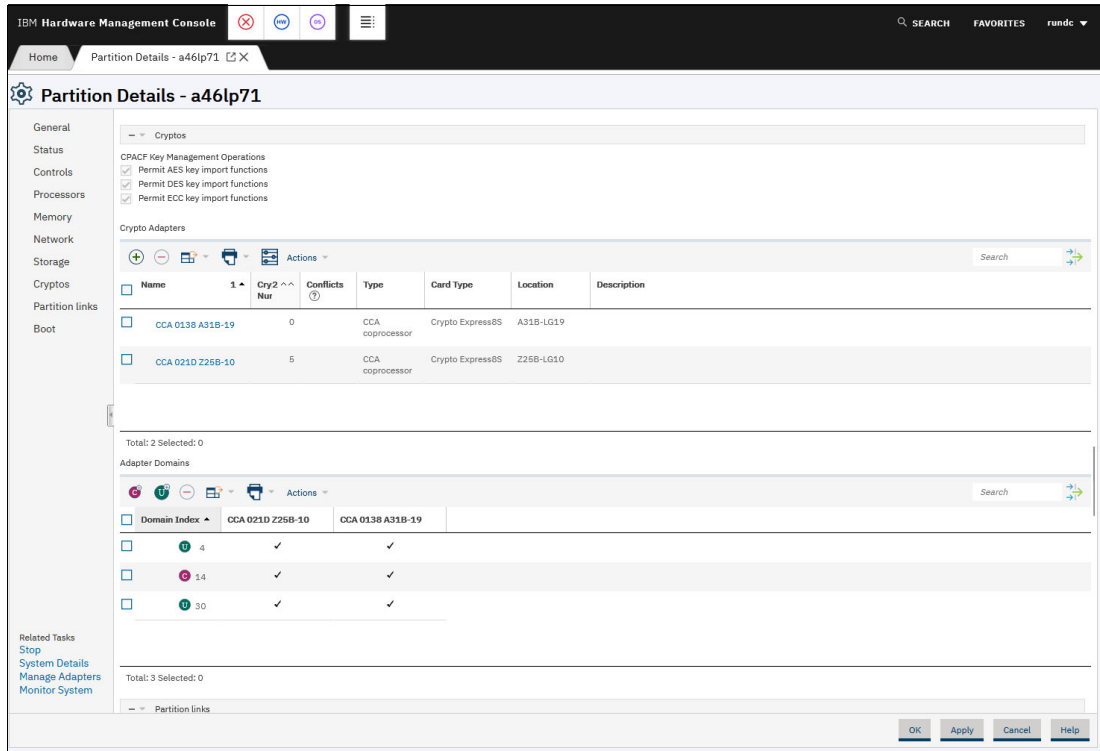


Figure 2-18 Partition Details - Cryptos

As an enhancement in DPM mode, the utilization and usage domain allocation is shown. This reflects the amount of domains configured for the card, which is limited to 85 domains. Clicking on + in **Crypto Adapters** will trigger a pop-up. Figure 2-19 illustrates the contents of this pop-up, demonstrating how to incorporate extra adapters. In this sample, an additional CEX8C and CEX8P card can be configured.

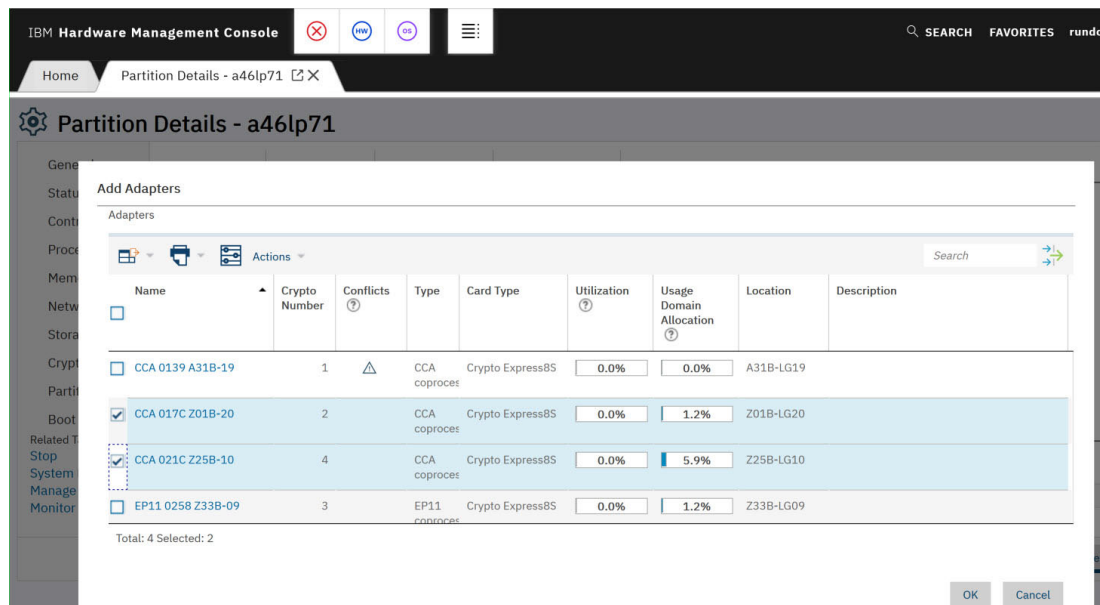


Figure 2-19 Partition Details - Crypto Adapters

To configure additional domains or change a current operating mode, click on either the **C** or **U** icon in the **Adapter Domain** section. Clicking either or will trigger a pop-up, as shown in Figure 2-20. That pop-up enables you to designate extra or alternative domains with the option to conceal utilization domains from different segments, thus forestalling conceivable replication of assignments right now.

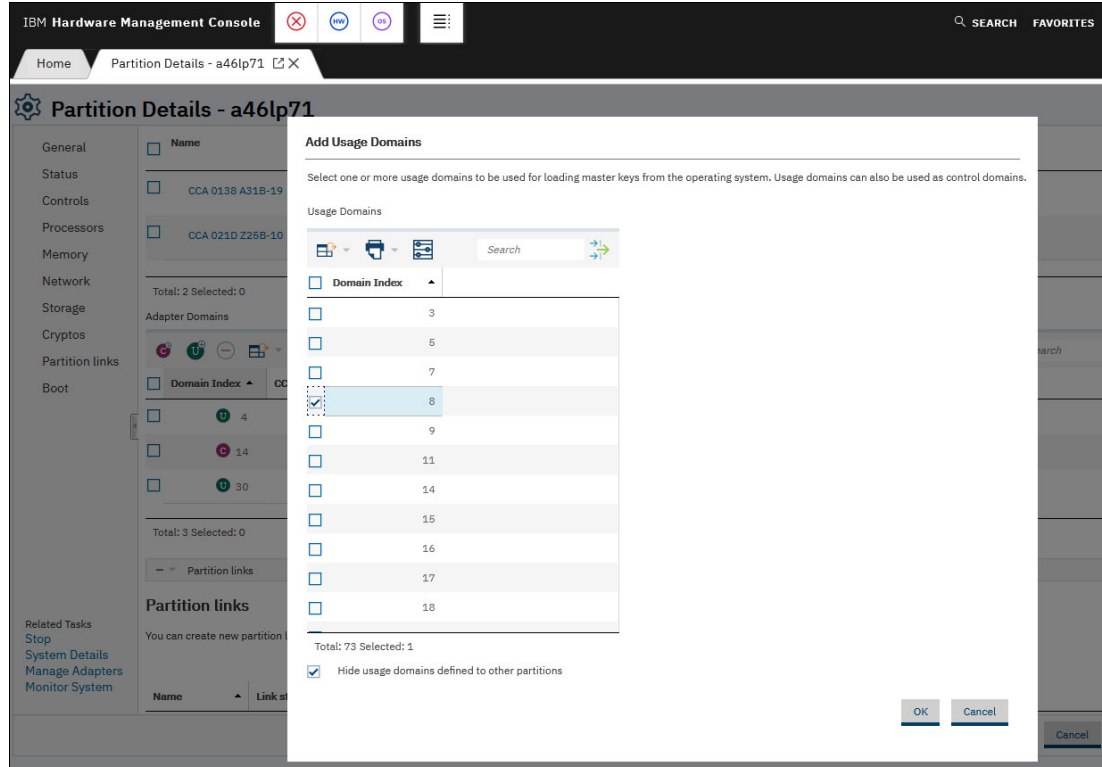


Figure 2-20 Partition Details - Crypto Domains

## 2.3 Master key setup

There are various ways to load master keys to CEX8S cards. This section offers guidance about master key setup and handling.

### 2.3.1 Trusted Key Entry

Dedicated examples of how to generate and activate master keys through Trusted Key Entry (TKE) are not part of this IBM Redbooks publication. Further information on setting up and configuring a TKE to load master keys can be found at [z/OS Trusted Key Entry Workstation](#) and in the [IBM MediaCenter](#).

### 2.3.2 Checking the master key setup

There is one important term related to master keys: Master Key Verification Pattern (MKVP). A Master Key Verification Pattern is a verification pattern that is generated for each master key stored in the master-key registers (new, current, and old). Key verification patterns confirm that the key sent by one party is the same key received by another. The MKVP is usually a raw byte sequence shown as a hexadecimal string of 16 to 64 characters. There are several ways to check the master key setup on crypto resources.

## CCA crypto adapters

With the CCA host library installed, access the command-line interface through the `ivp.e` tool. Known as the Installation Verification Program (IVP), this utility serves as the goto option for gathering details regarding the cryptographic resources accessible on the system through the CCA library.

Figure 2-21 shows a snippet from an `ivp.e` terminal output. For all four master keys used by CCA, the old, current, and new register state is shown. Configured master keys (MKs) should show the state **Valid** for all **Cur MK REG** fields.

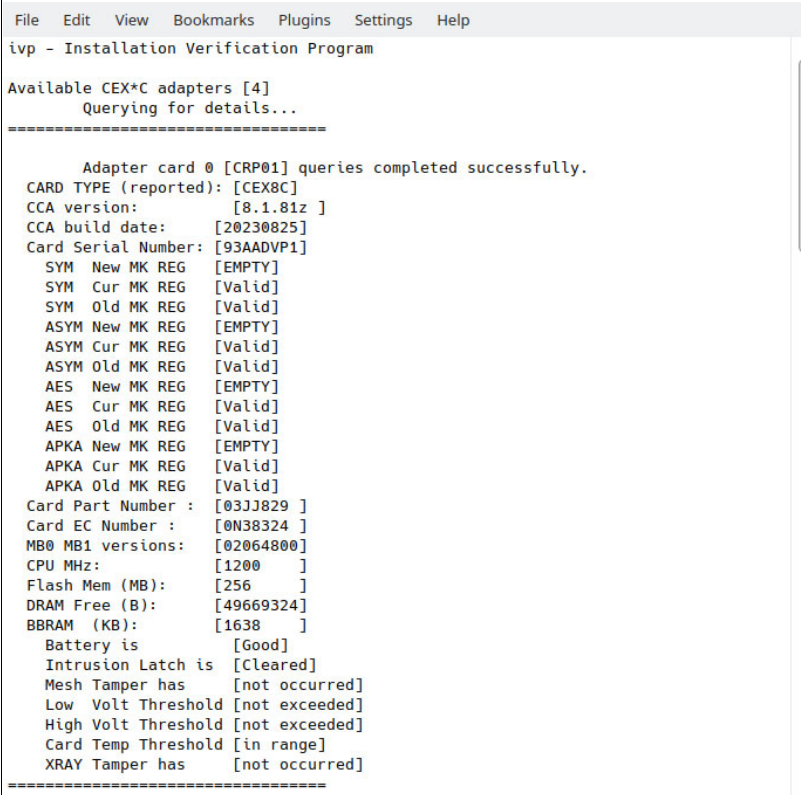
HSMs in CCA mode know four different master keys used for different purposes:

- ▶ AES Master Key
- ▶ SYM - DES Master Key
- ▶ ASYM - old RSA Master Key
- ▶ APKA - ECC, new RSA, and QSA Master Key

Unfortunately, `ivp.e` does not show the MKVPs of each of the current master keys. There is the CCA tool `panel.exe` for this job, which can handle the following arguments:

```
panel.exe --adapter=<adapter> --mk-query --mktype=[ASYM|SYM|AES|APKA]
--mkregister=[NEW|CURRENT|OLD]
```

An example command and output is shown in Figure 2-21.



```
File Edit View Bookmarks Plugins Settings Help
ivp - Installation Verification Program

Available CEX*C adapters [4]
  Querying for details...
=====

  Adapter card 0 [CRP01] queries completed successfully.
CARD TYPE (reported): [CEX8C]
CCA version:          [8.1.81z ]
CCA build date:       [20230825]
Card Serial Number:  [93AADVP1]
SYM New MK REG       [EMPTY]
SYM Cur MK REG       [Valid]
SYM Old MK REG       [Valid]
ASYM New MK REG      [EMPTY]
ASYM Cur MK REG      [Valid]
ASYM Old MK REG      [Valid]
AES New MK REG       [EMPTY]
AES Cur MK REG       [Valid]
AES Old MK REG       [Valid]
APKA New MK REG      [EMPTY]
APKA Cur MK REG      [Valid]
APKA Old MK REG      [Valid]
Card Part Number :   [03JJ829 ]
Card EC Number :     [0N38324 ]
MB0 MB1 versions:   [02064800]
CPU MHz:             [1200 ]
Flash Mem (MB):      [256 ]
DRAM Free (B):       [49669324]
BBRAM (KB):          [1638 ]
Battery is           [Good]
Intrusion Latch is   [Cleared]
Mesh Tamper has      [not occurred]
Low Volt Threshold   [not exceeded]
High Volt Threshold  [not exceeded]
Card Temp Threshold  [in range]
XRAY Tamper has      [not occurred]
=====
```

Figure 2-21 Example `ivp.e` output

Figure 2-22 shows an example query for the MKVP for the AES and the APKA current master key verification patterns.

```

File Edit View Bookmarks Plugins Settings Help
[root@rdbkvm3 ~]# panel.exe --adapter=0 --mk-query --mktype=AES --mkregister=CURRENT
==> Using Card [93AADVP1 (CRP01)]
Preparing to QUERY master key verification pattern

Query of Key Verification Pattern for Master key [AES-MK ] [KEY-KM ] returned:

RND[0000000000000000]
VER[E9A49A58CD039BED]
[root@rdbkvm3 ~]# panel.exe --adapter=0 --mk-query --mktype=APKA --mkregister=CURRENT
==> Using Card [93AADVP1 (CRP01)]
Preparing to QUERY master key verification pattern

Query of Key Verification Pattern for Master key [APKA-MK ] [KEY-KM ] returned:

RND[0000000000000000]
VER[5F2F27AAA2D59B4A]
[root@rdbkvm3 ~]# cat /sys/devices/ap/card00/00.000c/mkvp
AES NEW: empty 0x0000000000000000
AES CUR: valid 0xe9a49a58cd039bed
AES OLD: valid 0x7d10d17bc8a409c4
APKA NEW: empty 0x0000000000000000
APKA CUR: valid 0x5f2f27aaa2d59b4a
APKA OLD: valid 0x82a5e2cd5030d5ec
[root@rdbkvm3 ~]#

```

Figure 2-22 Example panel.exe MK query and sysfs output

The TKE can be used to display the MKVPs of the CCA master keys as well. On the **Crypto Module** dialog, select the **Domains** tab. Next, select the domain you want to query. Finally, the **Keys** tab at the bottom and the state and hash patterns (MKVPs) of all the MKs are shown. Figure 2-23 shows an example of the displayed **Keys** tab with MK information. This dialog view is capable of changing the MK values as well, but that is beyond the scope of this section.

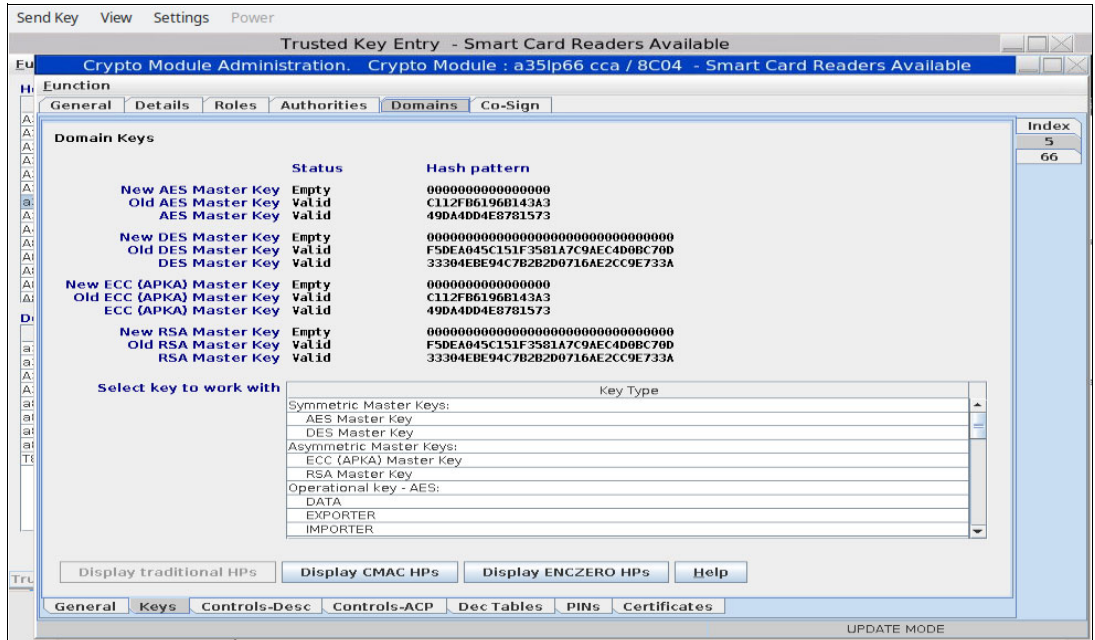


Figure 2-23 Example TKE dialog showing CCA MK information

There is also a way to query some MK information from the Linux command line. For each adapter and domain the following sysfs entry can supply some MK information:

```

/sys/devices/ap/card<aa>/<aa>.<ddd>/mkvp

```

## EP11 crypto adapters

The EP11 library comes with a command-line tool, `ep11info` that can retrieve basic information from crypto adapters in EP11 mode. EP11 uses one master key, often referred to as the *Wrapping Key* (WK) as it is used to wrap the user's working keys by encrypting them with the WK.

The following is the command to query the MKVP of the master key:

```
ep11info -m <adapter> --dominfo
```

The `-m` option specifies the crypto adapter which is in EP11 considers a module. Figure 2-24 shows an example of the command and its results after querying for domain information for the crypto adapters 5 and 7. The output column, named **wrapping key**, displays the MKVP as a 32 byte hexadecimal string.

```

File Edit View Bookmarks Plugins Settings Help
[root@rdbkvm1 ~]# ep11info -m 5,7 --dominfo
-----
module-nr  domain-nr  imprinted  sign. thr.  revoke thr.  compliance  wrapping key
-----
5          10         YES        2           2           00000001    8B991263 E3A8F4E4 BE0D5EC8 F0A4DF9E 25A32EBC 3B9EDA61 C9C062CE 00000000
7          10         YES        2           2           00000001    8B991263 E3A8F4E4 BE0D5EC8 F0A4DF9E 25A32EBC 3B9EDA61 C9C062CE 00000000
-----
[root@rdbkvm1 ~]# cat /sys/devices/ap/card85/05.000a/mkvp
WK CUR: valid 0x8b991263e3a8f4e4be0d5ec8f0a4df9e25a32ebc3b9eda61c9c062ce00000000
WK NEW: empty -
[root@rdbkvm1 ~]# cat /sys/devices/ap/card87/07.000a/mkvp
WK CUR: valid 0x8b991263e3a8f4e4be0d5ec8f0a4df9e25a32ebc3b9eda61c9c062ce00000000
WK NEW: empty -
[root@rdbkvm1 ~]#

```

Figure 2-24 Example `ep11info` query and `sysfs` output

With a working connection to a TKE workstation, the TKE's **Crypto Module** dialog can be used to display the verification hash value of the MK. Select the **Domains** tab at the top of the dialog. Next, select the domain number on the right border and finally, click on the **Domain Keys** tab. The center of the dialog shows the state of the current and new MK and the leftmost 16 bytes of the MKVP. Figure 2-25 shows an example of this TKE dialog.

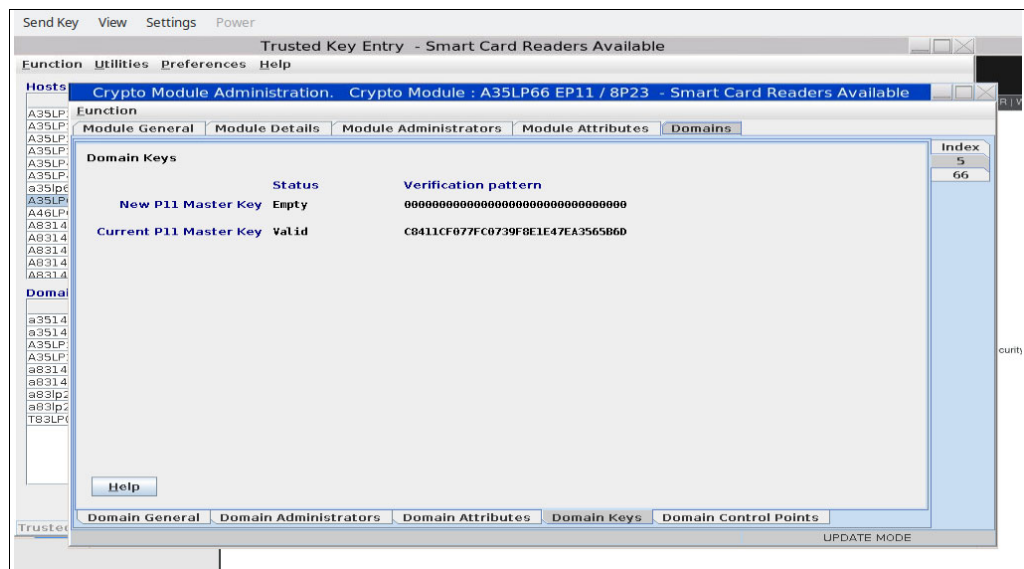


Figure 2-25 Example TKE dialog showing EP11 MK information

There is also a way to query some MK information from an EP11 crypto resource on the Linux command line. For each adapter and domain the following `sysfs` entry can supply some MK information as well:

```
/sys/devices/ap/card<aa>/<aa>.<ddd>/mkvp
```

This command supplies the state and MKVP of the current and new EP11 master key.

### 2.3.3 Master key setup with TKE

Setup of the master keys involves interaction between some partners and usually must comply with some company rules for security. There is an administrator of the target system involved and usually at least two different individuals acting on the TKE (this is known as the Four Eyes Principle).

The most comprehensive documentation for TKE handling is the IBM publication [Cryptographic Services ICSF Trusted Key Entry Workstation User's Guide, SC14-7511-10](#).

To define your TKE host and the control domains for an LPAR, see “Setting a master key on the Crypto Express EP11 coprocessor” on page 19 of [Linux on Z and LinuxONE Exploiting Enterprise PKCS #11 using openCryptoki 3.15, SC34-2713](#).

Additionally, see [Using the Linux on Z EP11 enablement](#).

There is a step-by-step tutorial for setting up the Master Key on an EP11 Coprocessor HSM included at [Setting a master key on the Crypto Express EP11 coprocessor](#).

The designers and developers of the Trusted Key Entry workstation have prepared a series of videos guiding you through all kinds of activities. For more information, see [z/OS Other resources](#).

To use the TKE workstation for master key setup, there needs to be an interconnection established between TKE and the Linux instance. The EP11 library comes with a systemd service, EP11TKEd, which listens to incoming network connections from the TKE. However, by default this service is disabled. The Linux administrator can start it by using the following command:

```
systemctl start EP11TKEd
```

and enabling it permanently with the following command:

```
systemctl enable EP11TKEd
```

The CCA library offers the systemd service named CSUTKEcat, which is also disabled by default and needs to be started in a similar way.

Both of these services are listening on dedicated TCP ports, as shown in Table 2-2.

Table 2-2 TKE daemon default ports

Service	Default TCP port
CSUTKEcat	50003
EP11TKEd	50004
EP11TKEd with TLS enabled	50104

By default, both TKE daemons EP11TKEd and CSUTKEcat enforce a TLS connection if OpenSSL is installed in 1.1 or higher. For a TLS connection to be established between TKE and daemon the TKE needs to import the daemon's certificate. This certificate and the respective private key are generated during package installation. For the exact location of these files and the procedure on how to import into the TKE follow the references provided below.



If there is a firewall service running on the Linux system, it may block incoming connections. A Linux administrator needs to exclude these ports from the firewall barrier. For example, the CCA TKE daemon firewall pass is enabled with:

```
firewall-cmd --zone=public --add-port=50003/tcp --permanent firewall-cmd --reload
```

More information about the EP11 TKE daemon can be found on the installed system in `/usr/share/doc/ep11/README_TKED.txt`.

The installation instructions for the CCA library are available in [Secure Key Solution with the Common Cryptographic Architecture Application Programmer's Guide Version 8.0, SC33-8294-11](#). It includes a section about the TKE daemon and how to use TLS for the network connections.

## 2.3.4 Control domains

When editing an LPAR profile at the **Crypto** tab the **Domains** table allows you to add domains in two flavors:

- ▶ Control

A control domain is restricted to only permit administrative commands, real crypto load using the HSMs accessible via this domain is not allowed. This means that one can for example set the master key(s) but not generate and use a working key for encryption or decryption load.

During activation of the LPAR profile, control only domains are not checked for unique assignment to only one active LPAR. It may happen and is supported to have more than one active LPAR sharing a control domain on the same system.

- ▶ Control and usage

A control and usage domain allows you to run administrative commands and real crypto load on the HSMs accessible using this domain. It represents a full-fledged crypto resource with no restrictions.

During activation of the LPAR profile, the intersection of assigned usage domains and assigned crypto cards is calculated. There must not be any other active LPAR having usage access to any of the HSMs in this set. If there is any overlap, the activation of the LPAR profile will fail.

The assignment of control only domains opens up the possibility to maintain master key settings for crypto resources used by other LPARs. The TKE software and the TKE daemons are prepared for this setup and support this indirect way of maintenance.





## Configure LINUX guests to use CEX adapters

This chapter helps you to take advantage of the encryption capabilities of IBM Z or IBM LinuxONE servers in KVM environments, especially the encryption features associated with CEX adapters, and how to define encryption functions on the host machine. It then provides help for setting up encryption on guest servers.

The chapter describes and discusses the following topics:

- ▶ Set up and configure a KVM host.
- ▶ Configure KVM guests to use CEX functionality:
  - Using default Linux commands, scripts
  - An alternative way that uses the `libvirt` commands<sup>1</sup>

---

<sup>1</sup> The examples, diagrams, descriptions and commands used in this chapter can be found at [Linux on IBM Systems](#).

## 3.1 Configuring a KVM host to provide CEX functionality

Once the KVM environment is installed and the necessary settings for using virtualization have been made, the server is ready to install guest servers on it. These servers will of course only be able to use the tools that the host machine offers them.

In order to make the various encryption (CEX) tools and their functions available to the guest servers, after the basic KVM configuration has been done, additional configuration steps need to be done on the host server.

The following subsections describe these steps, which extend KVM's virtualization capabilities towards encryption.

### 3.1.1 Setting up the KVM host machine to use CEX functions

In Chapter 2, “Overview of our environment” on page 11, we have shown how to configure CEX devices and the crypto domains on the HMC/DPM consoles of IBM Z/LinuxONE servers for each LPAR. On those interfaces, we not only had to make the device available but also specify which domains would be available in the LPAR to which we were attaching the CEX adapter(s).

In a KVM environment, each guest server must have its own dedicated crypto domain. On KVM, it is NOT possible to configure domains in a shared way.

Therefore, when performing the HMC/DPM configuration, care must be taken to configure as many crypto domains for the KVM partition as you want to use on the guest servers later.

In addition to these settings, in KVM environments we will need to use the Virtual Function I/O (VFIO) framework and the VFIO mediated device framework to pass host devices and their attributes to KVM guests.

For general information about VFIO and VFIO mediated devices, see `Documentation/vfio.txt` and `Documentation/vfio-mediated-device.txt` in the Linux kernel source. You can also find this information by searching for “vfio” at [The Linux Kernel](#).

### 3.1.2 What you should know about VFIO

Depending on the device type, Linux handles devices with specific device drivers. Figure 3-1 on page 33 shows the devices associated with their “native” device drivers (the PCI function with the PCI device driver, the DASD with the DASD device driver, and the crypto adapter with the zcrypt device driver).

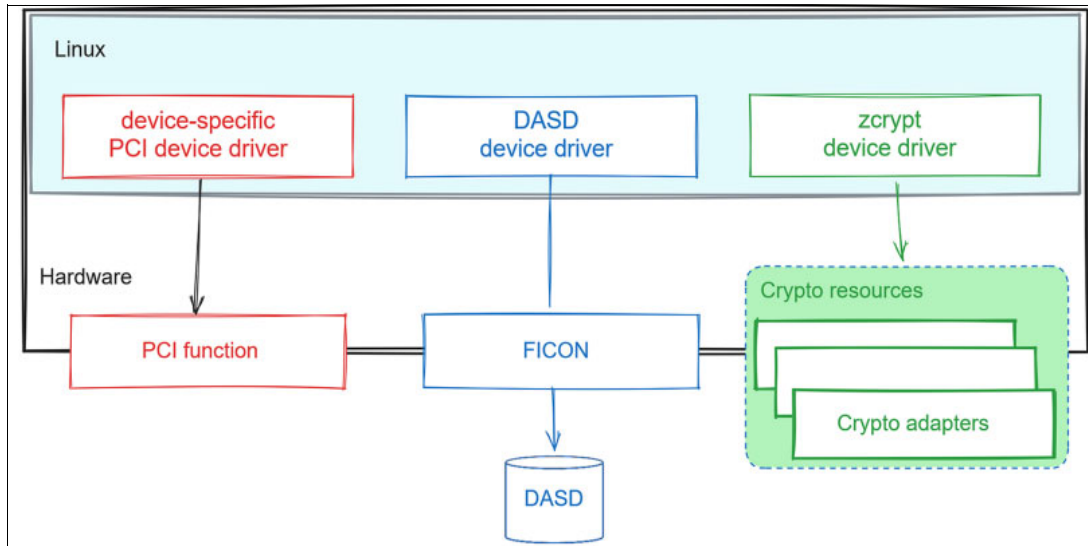


Figure 3-1 Device drivers on Linux

For Linux virtual servers running on the KVM post, QEMU provides virtual VFIO gateway devices that preserve the attributes of the host device.

Therefore, Linux on KVM accesses a pass-through device with the same device driver as the host would use to access the corresponding host resource. For example, Linux in LPAR mode uses the DASD device driver to access DASD disks. Correspondingly, Linux on KVM will use the zcrypt device driver to access the VFIO pass-through crypto resource.

Figure 3-2 shows virtual PCI functions as associated with a PCI device driver, a virtual DASD with the DASD device driver, and a virtual crypto adapter with the zcrypt device driver.

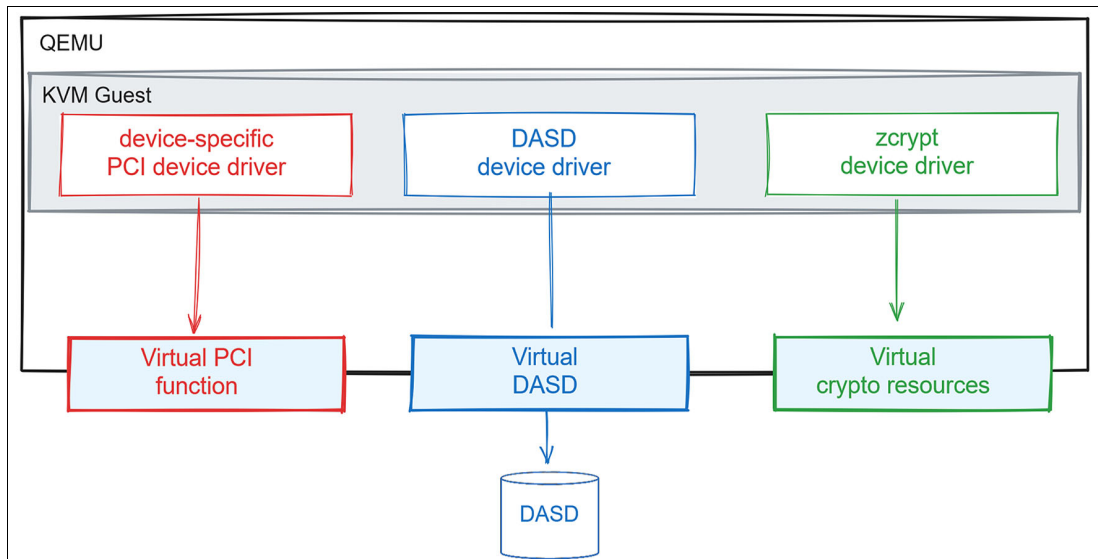


Figure 3-2 Device drivers for VFIO pass-through devices

To avoid contention, the KVM host must relinquish direct control of the host resource supporting the VFIO pass-through device.

For these host resources, the VFIO framework (Figure 3-3) replaces the KVM host's default device drivers with device-specific VFIO device drivers.

These replacement device drivers reserve host resources for guest use and provide access to these resources on behalf of the guest.

Red Hat Enterprise Linux 9.2 as a KVM host on IBM Z supports the following types of pass-through devices:

- ▶ PCIe
- ▶ CCW (DASD)
- ▶ Cryptographic adapter resources (AP queues)

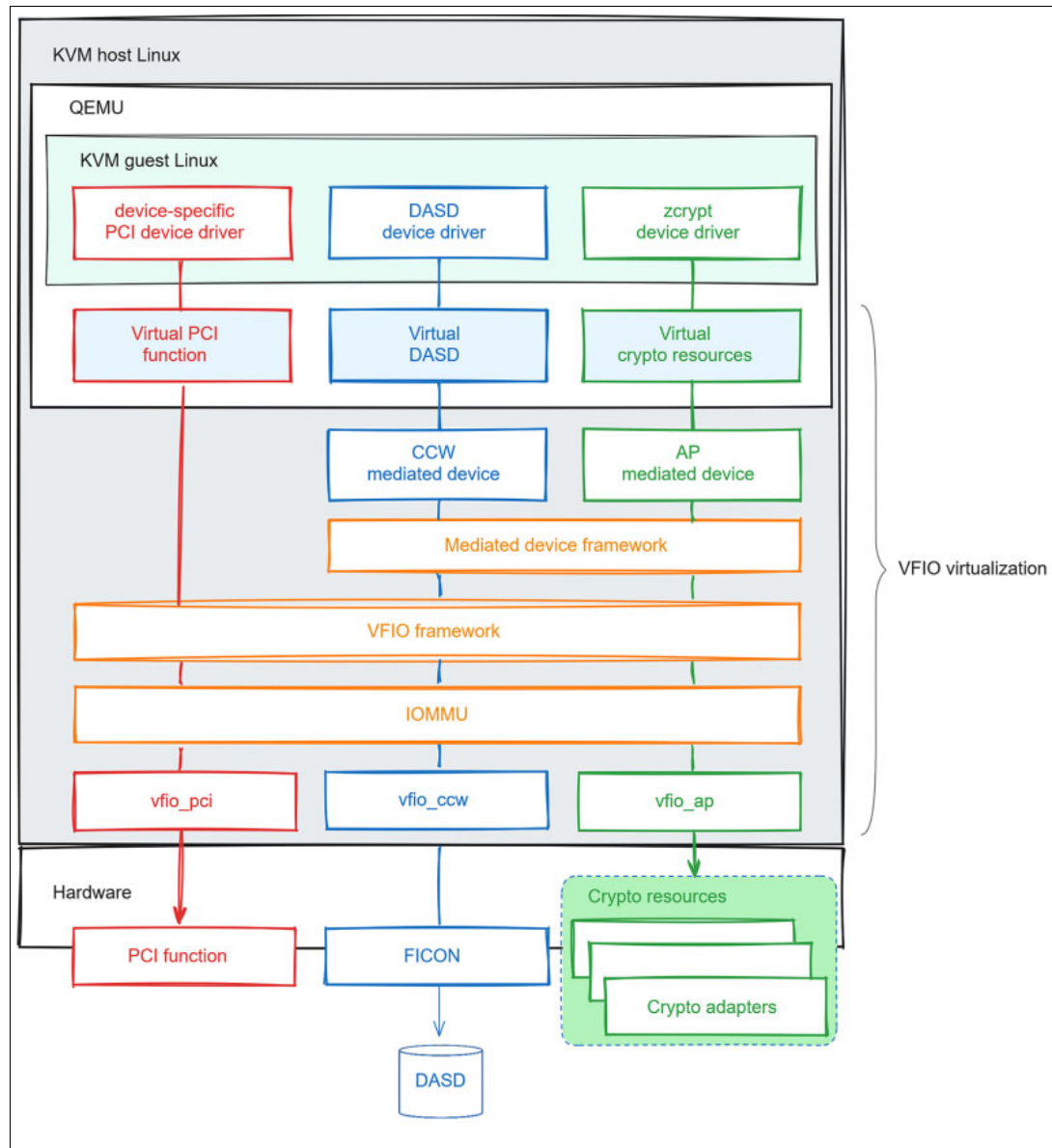


Figure 3-3 VFIO virtualization

The KVM host must define the resources that support the VFIO pass-through device and associate these resources with the appropriate VFIO device driver.

The required configuration steps depend on the type of device. You must create specific VFIO mediated devices for the cryptographic adapter resources to pass through.

The KVM hypervisor will then use the VFIO brokered devices as the source of the passing devices.

### 3.1.3 Checking kernel modules

The first thing to do is to find out if the kernel functions required to use crypto are loaded into the kernel (for example, if the cryptographic device driver is loaded).

This can be queried by using the `lsmod` command and filtering for the string “ap”, as shown in Example 3-1.

*Example 3-1 Check the kernel*

---

```
[root@rdbkvm4 ~]# lsmod | grep ap
macvtap          16384  13
macvlan          28672  1 macvtap
tap              28672  28 macvtap,vhost_net
tape_34xx        24576  0
tape             61440  1 tape_34xx
tape_class       16384  1 tape
```

---

As you can see, by default, the modules that support (allow the use of) VFIO and mediated device functionality are NOT loaded into the kernel. We have to load them.

This could be done dynamically with the `modprobe` command (Example 3-2). The command will load all the kernel modules (which were compiled separately) that are required to support the VFIO functionality.

*Example 3-2 Load modules and check the kernel modules again*

---

```
[root@rdbkvm4 ~]# modprobe vfio_ap

[root@rdbkvm4 ~]# lsmod | grep ap
vfio_ap          28672  0
macvtap          16384  13
macvlan          28672  1 macvtap
tap              28672  28 macvtap,vhost_net
tape_34xx        24576  0
tape             61440  1 tape_34xx
tape_class       16384  1 tape
mdev           28672  2 vfio_ccw,vfio_ap
vfio             49152  4 vfio_ccw,vfio_iommu_type1,mdev,vfio_ap
kvm              471040  21 vfio_ap
```

---

If you want to make the configuration “reboot-proof”, create the file shown in Example 3-3 in the directory, `/etc/modules-load.d/`.

*Example 3-3 Create a crypto.conf*

---

```
root@rdbkvm4 ~]# vi /etc/modules-load.d/crypto.conf
# Make sure the vfio_ap device driver is loaded.
# It's usually compiled as a separate module,
# so it's likely to be loaded separately.
```

```
# The command loads vfio_ap and any additional modules that may be needed.
vfio_ap
```

Next, we check how much and what type of crypto functionality is available to us. You can do this by using the **1scrypt** command, shown in Example 3-4.

*Example 3-4 Check crypto availability on KVM level*

```
[root@rdbkvm4 ~]# 1scrypt
```

CARD.DOM	TYPE	MODE	STATUS	REQUESTS
00	CEX8C	CCA-Coproc	online	3939
00.000d	CEX8C	CCA-Coproc	online	3939
00.0011	CEX8C	CCA-Coproc	online	0
01	CEX8A	Accelerator	online	0
01.000d	CEX8A	Accelerator	online	0
01.0011	CEX8A	Accelerator	online	0
02	CEX8C	CCA-Coproc	online	2208
02.000d	CEX8C	CCA-Coproc	online	2208
02.0011	CEX8C	CCA-Coproc	online	0
03	CEX8A	Accelerator	online	0
03.000d	CEX8A	Accelerator	online	0
03.0011	CEX8A	Accelerator	online	0
04	CEX8C	CCA-Coproc	online	2208
04.000d	CEX8C	CCA-Coproc	online	2208
04.0011	CEX8C	CCA-Coproc	online	0
05	CEX8P	EP11-Coproc	online	193
05.000d	CEX8P	EP11-Coproc	online	193
05.0011	CEX8P	EP11-Coproc	online	0
06	CEX8C	CCA-Coproc	online	2208
06.000d	CEX8C	CCA-Coproc	online	2208
06.0011	CEX8C	CCA-Coproc	online	0
07	CEX8P	EP11-Coproc	online	243
07.000d	CEX8P	EP11-Coproc	online	243
07.0011	CEX8P	EP11-Coproc	online	0

If we add the **-V** (Verbose) option to the previous command we will get much more information (shown in Example 3-5).

*Example 3-5 Check crypto availability on KVM level - Verbose*

```
root@rdbkvm4 ~]# 1scrypt -V
```

CARD.DOM	TYPE	MODE	STATUS	REQUESTS	PENDING	HWTYP	QDEPTH	FUNCTIONS	DRIVER
-									
00	CEX8C	CCA-Coproc	online	3939	0	14	08	S--D--NF-	cex4card
00.000d	CEX8C	CCA-Coproc	online	3939	0	14	08	S--D--NF-	cex4queue
00.0011	CEX8C	CCA-Coproc	online	0	0	14	08	S--D--NF-	cex4queue
01	CEX8A	Accelerator	online	0	0	14	08	-MC-A-NF-	cex4card
01.000d	CEX8A	Accelerator	online	0	0	14	08	-MC-A-NF-	cex4queue
01.0011	CEX8A	Accelerator	online	0	0	14	08	-MC-A-NF-	cex4queue
02	CEX8C	CCA-Coproc	online	2208	0	14	08	S--D--NF-	cex4card
02.000d	CEX8C	CCA-Coproc	online	2208	0	14	08	S--D--NF-	cex4queue
02.0011	CEX8C	CCA-Coproc	online	0	0	14	08	S--D--NF-	cex4queue
03	CEX8A	Accelerator	online	0	0	14	08	-MC-A-NF-	cex4card
03.000d	CEX8A	Accelerator	online	0	0	14	08	-MC-A-NF-	cex4queue



03.0011	CEX8A	Accelerator	online	0	0	14	08	-MC-A-NF-	cex4queue
04	CEX8C	CCA-Coproc	online	2208	0	14	08	S--D--NF-	cex4card
04.000d	CEX8C	CCA-Coproc	online	2208	0	14	08	S--D--NF-	cex4queue
04.0011	CEX8C	CCA-Coproc	online	0	0	14	08	S--D--NF-	cex4queue
05	CEX8P	EP11-Coproc	online	193	0	14	08	-----XNF-	cex4card
05.000d	CEX8P	EP11-Coproc	online	193	0	14	08	-----XNF-	cex4queue
05.0011	CEX8P	EP11-Coproc	online	0	0	14	08	-----XNF-	cex4queue
06	CEX8C	CCA-Coproc	online	2208	0	14	08	S--D--NF-	cex4card
06.000d	CEX8C	CCA-Coproc	online	2208	0	14	08	S--D--NF-	cex4queue
06.0011	CEX8C	CCA-Coproc	online	0	0	14	08	S--D--NF-	cex4queue
07	CEX8P	EP11-Coproc	online	243	0	14	08	-----XNF-	cex4card
07.000d	CEX8P	EP11-Coproc	online	243	0	14	08	-----XNF-	cex4queue
07.0011	CEX8P	EP11-Coproc	online	0	0	14	08	-----XNF-	cex4queue

You can see that the administrator has connected eight CEX adapters to this KVM LPAR.

In this configuration each adapter will be able to use domains 13 and 17 (hex represented as 000d and 11 respectively).

**Important:** These crypto domains, although having the same domain ID, are **different**, as each one belongs to a different CEX adapter.

### 3.1.4 Configuring VFIO AP queues

Having checked the adapters and domains assigned to the KVM host server, and verified that VFIO and mediated device are supported by the kernel, we can move on to the next configuration task.

Before we get to the specific commands and instructions, let's briefly look at what's going on behind the scenes.

The resources of a cryptographic adapter are called AP queues. AP queues are in fact nothing more than the crypto domains designated for use on each CEX adapter.

Therefore, each AP queue is always defined by a pair of adapter and domains, as shown in Table 3-1 on page 38. AP queues correspond to a cell in the table.

In our case, we assigned eight adapters to the KVM, and made domains 13 and 17 (hex representation:000d, 0011) available on each of them.

Table 3-1 on page 38 is often called the matrix of AP queues. The term matrix is widely used for representations of tables of AP queues, for example, in sysfs attributes. The gray table cells shown in Table 3-1 on page 38 indicate the implicitly assigned AP queues, which are 00.000d, 01.000d, ... 08.000d,... and 00.0011,.. 08.0011.

Table 3-1 Matrix of AP queues

DOMAINS \ ADAPTERS	ADAPTERS					
	00	01	02	07	08	
0000	00.0000	01.0000	02.0000	07.0000	08.0000	
0001	00.0001	01.0001	02.0001	07.0001	08.0001	
0002	00.0002	01.0002	02.0002	07.0002	08.0002	
0000d	00.000d	01.000d	02.000d	07.000d	08.000d	
00011	00.0011	01.0011	02.0011	07.0011	08.0011	

You work with matrices of AP queues at different levels:

► LPAR level

This is actually the level of AP queues available to the KVM host machine.

For Linux servers running in LPAR mode, these are the AP queues that are defined by the LPAR AP configuration. On a running server, this matrix can be changed by dynamically modifying the LPAR interface and domain configurations.

► Host level

The AP queues that are controlled by the zcrypt device driver of the host machine, so that they can be accessed by applications running on the host machine.

By default, all queues accessible to the host are controlled by the host's zcrypt device driver.

This default can be modified (dynamically) with the kernel parameters **ap.apmask=** and **ap.aqmask=**. (In the background, the available adapters can be modified with the chzdev command.)

This status is presented by the Example 3-5 on page 36.

When the vfio\_ap device driver is loaded, it takes control of AP queues that are not controlled by the host zcrypt device driver.

These AP queues can then be forwarded to KVM clients via AP-mediated devices.

► Mediated device level

AP Queues assigned to a VFIO AP mediated device.

The KVM endpoint makes AP queues available to servers by attaching a VFIO AP-mediated device to the KVM endpoint.

Brokered devices must not contain queues that are assigned to zcrypt device drivers on the host machine at the host machine level.

For each line to be usable at the brokered device level, it must first be “unlocked” and then mapped to the device. You can dynamically change the adapter and domain assignment through a mediated device's sysfs attributes, see Table 3-2.

Table 3-2 Mediated device sysfs attributes

Attribute	Explanation
assign_adapter	Write an adapter ID to this attribute to assign the adapter to the mediated device. Specify the adapter ID in decimal or hexadecimal notation. For hexadecimal notation, use the prefix “0x”. Example: <b># echo 0x0a &gt; assign_adapter</b>

Attribute	Explanation
assign_control_domain	Write a domain ID to this attribute to assign the domain as a control domain to the mediated device. Assign a control domain for each usage domain that you assign to the mediated device, so that you can manage your domains from the guest that uses the mediated device. Specify the domain ID in decimal or hexadecimal notation. For hexadecimal notation, use the prefix "0x". Example: <b>#echo 0x001b &gt; assign_control_domain</b>
assign_domain	Write a domain ID to this attribute to assign a usage domain to the mediated device. Specify the domain ID in decimal or hexadecimal notation. For hexadecimal notation, use the prefix "0x". Example: <b>#echo 0x001b &gt; assign_domain</b>
control_domains	Read this attribute to list the assigned control domains. Example: <b>#cat control_domains 001b</b>
guest_matrix	Read this attribute to list the subset of assigned AP queues that are eligible for KVM guests.
matrix	Read this attribute to list the assigned AP queues that result from the adapter and domain assignments. Example: <b>#cat matrix 0a.001b</b>
mdev_type	Symbolic link that points to the vfio_ap-passthrough directory.
remove	Write 1 to this attribute to remove the mediated device. Example: <b>#echo 1 &gt; remove</b>
subsystem	Symbolic link that points to the matrix bus
unassign_adapter	Write an adapter ID to this attribute to remove the adapter from the mediated device. Specify the adapter ID in decimal or hexadecimal notation. For hexadecimal notation, use the prefix "0x". Example: <b>#echo 0x0a &gt; unassign_adapter</b>
unassign_control_domain	Write a domain ID to this attribute to remove the domain from the control domains of the mediated device. Specify the domain ID in decimal or hexadecimal notation. For hexadecimal notation, use the prefix "0x". Example: <b>#echo 0x001b &gt; unassign_control_domain</b>
unassign_domain	Write a domain ID to this attribute to remove the domain from the usage domains of the mediated device. Specify the domain ID in decimal or hexadecimal notation. For hexadecimal notation, use the prefix "0x". Example: <b>#echo 0x001b &gt; unassign_domain</b>

Figure 3-4 on page 40 is a schematic illustration of the three matrix layers as a succession of filters, where grey cells indicate assigned AP queues. An AP queue can be passed on to a KVM guest only if it is suitably assigned at each level.

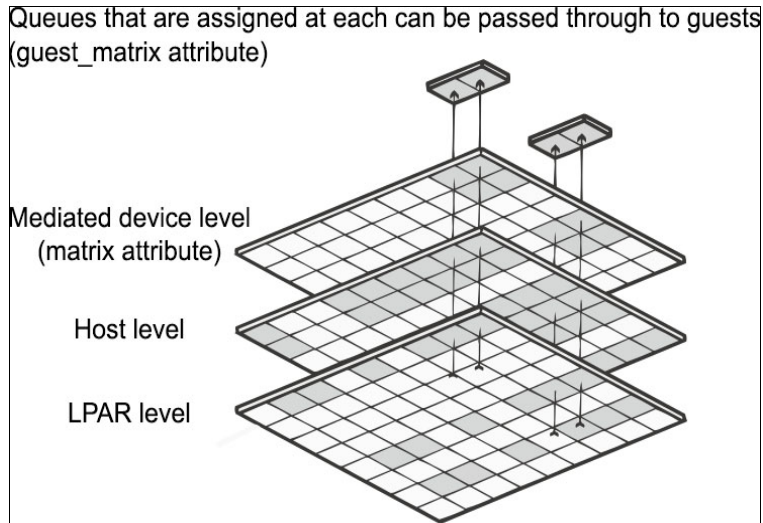


Figure 3-4 Matrices as a succession of filters

### Hotplug and hot unplug

On a running KVM guest, attaching a mediated device results in hotplug events for all those AP queues of the mediated device that are controlled by the `vfio_ap` device driver on the host, which implies that both their adapter and domain is configured for the LPAR.

With a mediated device attached to a KVM guest, hotplug or hot unplug events for queues can be triggered at the LPAR or mediated device configuration level:

- ▶ Dynamic changes to the AP configuration for the LPAR on the SE or HMC
- ▶ Dynamic changes of the mediated device

Detaching a mediated device results in hot unplug events for all AP queues of the mediated device that are used by the KVM guest.

### Persistence across host reboots

You can set up your cryptographic resources such that, after a host reboot, your mediated devices are ready to be attached to KVM guests, without further configuration steps. This setup requires a persistent assignment of AP queues to the `vfio_ap` device driver and persistent mediated devices with their assignment of AP queues.

Use the `chzdev` command to persistently assign AP queues to the `vfio_ap` device driver. Use the `nodedev-define` command to make a mediated device and its assignment of AP queues persistent.

Follow this procedure to make settings permanent, and reboot-proof:

1. Ensure that a VFIO device driver controls the resource on the host server.
2. The VFIO device driver reserves the passing device for the KVM guest and accesses the corresponding host resource on behalf of the guest.
3. Release the resource from the control of the default device driver - `zcrypt` - and then assign the resource to the appropriate VFIO device driver.
4. For KVM configuration, as discussed above, “take” crypto resources from KVM (free them up) and then pass them on to the guest servers via VFIO pass-through.

As it stands, no domains are available for guest servers. This is verified by using the command shown in Example 3-6 on page 41.

---

### Example 3-6 Verify vfio

---

```
[root@rdbkvm4 ~]# lszcrypt -V | grep vfio
[root@rdbkvm4 ~]#
```

---

The output is empty. All resources are managed by zcrypt. Further preparations are needed to identify the assets.

Each of the individual devices is clearly identified by their UUID. In this scenario, define these UUIDs. There are two ways to do this. We can use the Linux built-in UUID generator, as shown in Example 3-7, where each device must have a dedicated UUID associated with it.

---

### Example 3-7 Generate UUID

---

```
[root@rdbkvm4 ~]# uuidgen
4ecc88db-541e-4d24-a9d2-8c31e02ad86f
```

---

Alternatively, we can create a “talking” UUID. For example, for domains 13 and 17 (hex: 000d and 0011) we can manually “assign” the following two UUIDs, as shown in Example 3-8.

---

### Example 3-8 Generated UUID-s

---

```
000d000d-000d-000d-000d-000d000d000d
00110011-0011-0011-0011-001100110011
```

---

The AP queues are then assigned to the corresponding cells in the matrix, making them available to KVM guest servers.

In the Example 3-9, we use the CEX adapters 05 and 07. The corresponding assignments can be done by issuing commands one after the other, but you can also combine these commands into a similar bash script. Create and edit this script using the following command:

```
[root@rdbkvm4 ~]#vi crypto-assign.sh
```

The content of our script is shown in Example 3-9.

---

### Example 3-9 Create crypto-assign.sh

---

```
#!/bin/bash

# 1 - Free all adapters by specifying the following command:
echo 0x0 > /sys/bus/ap/apmask

# 2 - Free all domains by specifying the following command:
echo 0x0 > /sys/bus/ap/aqmask

# Domains as 13, 17 are decimal and the same 0x000d and 0x0011 are in Hex
var01="0x000d 0x0011"
var02="000d000d-000d-000d-000d-000d000d000d 00110011-0011-0011-0011-001100110011"
nbDom=`echo -n $var01 | wc -w`

for i in `seq ${nbDom}`
do
    Domain=`echo ${var01} | cut -d' ' -f${i}`
    uuid=`echo ${var02} | cut -d' ' -f${i}`

    #Create the device by writing the UUID to /sys/devices/vfio_ap/matrix with
    Domain xx
```

```

echo ${uuid} >
/sys/devices/vfio_ap/matrix/mdev_supported_types/vfio_ap-passthrough/create

echo 0x05 > /sys/devices/vfio_ap/matrix/${uuid}/assign_adapter
echo 0x07 > /sys/devices/vfio_ap/matrix/${uuid}/assign_adapter

echo ${Domain} > /sys/devices/vfio_ap/matrix/${uuid}/assign_domain
echo ${Domain} > /sys/devices/vfio_ap/matrix/${uuid}/assign_control_domain

cat /sys/devices/vfio_ap/matrix/${uuid}/matrix

echo " Domain - " ${Domain} " - " ${uuid}
sleep 1
done

```

Run the script. The result is the output of the **cat** command in the script, as shown in Example 3-10.

*Example 3-10 Output of crypto-assign.sh script*

```

[root@rdbkvm4 ~]# chmod +x crypto-assign.sh
[root@rdbkvm4 ~]# ./crypto-assign.sh
05.000d
07.000d
Domain - 0x000d - 000d000d-000d-000d-000d-000d000d000d
05.0011
07.0011
Domain - 0x0011 - 00110011-0011-0011-0011-001100110011

```

**Note:** In the script, control of all the adapters has been “taken” away from the zcrypt device driver and “handed over” to the VFIO. After running the script, the domains on adapters 5 and 7 have been assigned

Check the crypto by using the **lszcrypt -V** command again, as shown in Example 3-11.

*Example 3-11 Check the crypto*

```

[root@rdbkvm4 ~]# lszcrypt -V
CARD.DOM TYPE MODE STATUS REQUESTS PENDING HWTYPE QDEPTH FUNCTIONS DRIVER
-----
-
00 CEX8C CCA-Coproc online 3939 0 14 08 S--D--NF- cex4card
00.000d CEX8C CCA-Coproc unassigned - - 14 08 S--D--NF- vfio_ap
00.0011 CEX8C CCA-Coproc unassigned - - 14 08 S--D--NF- vfio_ap
01 CEX8A Accelerator online 0 0 14 08 -MC-A-NF- cex4card
01.000d CEX8A Accelerator unassigned - - 14 08 -MC-A-NF- vfio_ap
01.0011 CEX8A Accelerator unassigned - - 14 08 -MC-A-NF- vfio_ap
02 CEX8C CCA-Coproc online 2208 0 14 08 S--D--NF- cex4card
02.000d CEX8C CCA-Coproc unassigned - - 14 08 S--D--NF- vfio_ap
02.0011 CEX8C CCA-Coproc unassigned - - 14 08 S--D--NF- vfio_ap
03 CEX8A Accelerator online 0 0 14 08 -MC-A-NF- cex4card
03.000d CEX8A Accelerator unassigned - - 14 08 -MC-A-NF- vfio_ap
03.0011 CEX8A Accelerator unassigned - - 14 08 -MC-A-NF- vfio_ap
04 CEX8C CCA-Coproc online 2208 0 14 08 S--D--NF- cex4card
04.000d CEX8C CCA-Coproc unassigned - - 14 08 S--D--NF- vfio_ap
04.0011 CEX8C CCA-Coproc unassigned - - 14 08 S--D--NF- vfio_ap
05 CEX8P EP11-Coproc online 193 0 14 08 -----XNF- cex4card

```

05.000d	CEX8P	EP11-Coproc	assigned	-	-	14	08	-----XNF-	vfio_ap
05.0011	CEX8P	EP11-Coproc	assigned	-	-	14	08	-----XNF-	vfio_ap
06	CEX8C	CCA-Coproc	online	2208	0	14	08	S--D--NF-	cex4card
06.000d	CEX8C	CCA-Coproc	unassigned	-	-	14	08	S--D--NF-	vfio_ap
06.0011	CEX8C	CCA-Coproc	unassigned	-	-	14	08	S--D--NF-	vfio_ap
07	CEX8P	EP11-Coproc	online	243	0	14	08	-----XNF-	cex4card
07.000d	CEX8P	EP11-Coproc	assigned	-	-	14	08	-----XNF-	vfio_ap
07.0011	CEX8P	EP11-Coproc	assigned	-	-	14	08	-----XNF-	vfio_ap

You can also see how the matrix appears in the file system (Example 3-12).

### Example 3-12 Checking the matrix

```
[root@rdbkvm4 ~]# cd /sys/devices/vfio_ap/matrix/

[root@rdbkvm4 matrix]# ls -al
total 0
drwxr-xr-x. 5 root root 0 Oct 3 06:35 .
drwxr-xr-x. 3 root root 0 Oct 3 06:35 ..
drwxr-xr-x. 2 root root 0 Oct 3 09:15 000d000d-000d-000d-000d-000d000d000d
drwxr-xr-x. 2 root root 0 Oct 3 09:15 00110011-0011-0011-0011-001100110011
lrwxrwxrwx. 1 root root 0 Oct 4 08:53 driver -> ../../../../bus/matrix/drivers/vfio_ap
drwxr-xr-x. 3 root root 0 Oct 3 06:35 mdev_supported_types
lrwxrwxrwx. 1 root root 0 Oct 3 06:35 subsystem -> ../../../../bus/matrix
-rw-r--r--. 1 root root 4096 Oct 4 08:53 uevent

[root@rdbkvm4 matrix]# cd 000d000d-000d-000d-000d-000d000d000d/

[root@rdbkvm4 000d000d-000d-000d-000d-000d000d000d]# ls -al
total 0
drwxr-xr-x. 2 root root 0 Oct 3 09:15 .
drwxr-xr-x. 5 root root 0 Oct 3 06:35 ..
--w-----. 1 root root 4096 Oct 4 05:27 assign_adapter
--w-----. 1 root root 4096 Oct 4 05:27 assign_control_domain
--w-----. 1 root root 4096 Oct 4 05:27 assign_domain
-r--r--r--. 1 root root 4096 Oct 4 08:53 control_domains
lrwxrwxrwx. 1 root root 0 Oct 4 08:53 driver ->
../../../../bus/mdev/drivers/vfio_ap_mdev
lrwxrwxrwx. 1 root root 0 Oct 3 09:15 iommu_group -> ../../../../kernel/iommu_groups/9
-r--r--r--. 1 root root 4096 Oct 3 09:15 matrix
lrwxrwxrwx. 1 root root 0 Oct 3 09:15 mdev_type ->
../mdev_supported_types/vfio_ap-passthrough
--w-----. 1 root root 4096 Oct 4 08:53 remove
lrwxrwxrwx. 1 root root 0 Oct 3 09:15 subsystem -> ../../../../bus/mdev
-rw-r--r--. 1 root root 4096 Oct 4 08:53 uevent
--w-----. 1 root root 4096 Oct 4 08:53 unassign_adapter
--w-----. 1 root root 4096 Oct 4 08:53 unassign_control_domain
--w-----. 1 root root 4096 Oct 4 08:53 unassign_domain

[root@rdbkvm4 000d000d-000d-000d-000d-000d000d000d]# cd
../00110011-0011-0011-0011-001100110011/
[root@rdbkvm4 00110011-0011-0011-0011-001100110011]#
[root@rdbkvm4 00110011-0011-0011-0011-001100110011]# cat matrix
05.0011
07.0011
[root@rdbkvm4 00110011-0011-0011-0011-001100110011]# cd
../000d000d-000d-000d-000d-000d000d000d/
[root@rdbkvm4 000d000d-000d-000d-000d-000d000d000d]# cat matrix
05.000d
```

Make the mappings described in Example 3-12 on page 43 permanent and available even after a KVM server restart with a service script. This service runs the bash script created in Example 3-12 on page 43 when the KVM server is started, and thus perform the necessary mappings. The following command opens up the `crypto-assign.service` file for editing:

```
[root@rdbkvm4 ~]# vi /etc/systemd/system/crypto-assign.service
```

Make the changes shown in Example 3-13.

*Example 3-13 crypto-assign.service*

---

```
[Unit]
Description=initialize crypto
After=network.target
StartLimitIntervalSec=0

[Service]
Type=simple
User=root
ExecStart=/bin/bash /root/crypto-assign.sh

[Install]
WantedBy=multi-user.target
```

---

Example 3-14 shows the command you need to run to enable `crypto-assign.service` (and start).

*Example 3-14 Enable and start the service*

---

```
[root@rdbkvm4 ~]# systemctl enable --now crypto-assign.service
Created symlink /etc/systemd/system/multi-user.target.wants/crypto-assign.service
? /etc/systemd/system/crypto-assign.service.
```

---

### 3.1.5 Configuring the mediated device - KVM guest level

The last step is to add the mediated device to the KVM guest configuration. There are two ways to do this:

- ▶ If the server is not running, you can modify the file describing the guest server with the `virsh edit <VIRTUAL-SERVER>` command.
- ▶ Attaching a device: Dynamically hotplug attach and detach devices on the running guest server with the following commands:

```
virsh attach-device <VIRTUAL-SERVER> <device-configuration-XML-filename>
<scope>
```

Dynamic device attachment/detachment on KVM guest servers is done as follows:

#### Attaching a device

You can hotplug devices to a running virtual server, add devices to the persistent virtual server configuration, or both.



### ***Before you begin***

Ensure that the new device is not already assigned to the virtual server.

To list the devices that are assigned to a virtual server, you can:

- ▶ Display the current libvirt-internal configuration.
- ▶ Use the **virsh domblklist** command to display a list of currently assigned block devices or the **virsh domiflist** command to display a list of currently assigned interface devices.
- ▶ You need a device configuration-XML file for the device.

### ***Attaching a device***

Attach the device by using the following **virsh attach-device** command, for example:

```
# virsh attach-device <VS> <device-configuration-XML-filename> <scope>
```

- ▶ <device-configuration-XML-filename> is the name of the device configuration-XML file.
- ▶ <VS> Is the name of the virtual server as defined in the domain configuration-XML file.
- ▶ <scope> Specifies the scope of the command:
  - **--live**  
Hotplugs the device to a running virtual server. This configuration change does not persist across stopping and starting the virtual server.
  - **--config**  
Adds the device to the persistent virtual server configuration. The device becomes available when the virtual server is next started. This configuration change persists across stopping and starting the virtual server.
  - **--persistent**  
Adds the device to the persistent virtual server configuration and hotplugs it if the virtual server is running. This configuration change persists across stopping and starting the virtual server. This option is equivalent to specifying both **--live** and **--config**.

### ***Detaching a device***

You can unplug devices from a running virtual server, remove devices from the persistent virtual server configuration, or both.

### ***Before you begin***

You need a device configuration-XML file to detach a device from a virtual server. If the device has previously been attached to the virtual server, use the device configuration-XML file that was used to attach the device.

### ***Procedure***

Detach the device by using the following **virsh detach-device** command, for example:

```
# virsh detach-device <VS> <device-configuration-XML-filename> <scope>
```

- ▶ <device-configuration-XML-filename> Is the name of the device configuration-XML file.
- ▶ <VS> Is the name of the virtual server as defined in the domain configuration-XML file.
- ▶ <scope> Specifies the scope of the command:
  - **--live**  
Unplugs the device from a running virtual server. This configuration change does not persist across stopping and starting the virtual server.

- **--config**  
Removes the device from the persistent virtual server configuration. The device becomes unavailable when the virtual server is next started. This configuration change persists across stopping and starting the virtual server.
- **--persistent**  
Removes the device from the persistent virtual server configuration and unplugs it if the virtual server is running. This configuration change persists across stopping and starting the virtual server. This option is equivalent to specifying both **--live** and **--config**.

Recall the output from the `crypto-assign-sh` script, as shown in Example 3-10 on page 42:

```
05.000d
07.000d
  Domain - 0x000d - 000d000d-000d-000d-000d-000d-000d000d000d
05.0011
07.0011
  Domain - 0x0011 - 00110011-0011-0011-0011-001100110011
```

The 000d and 0011 crypto domains are assigned to the AP queue on the 05 and 07 CEX adapters. We should create the mediated device with the proper parameters (Example 3-15) by editing the correct file, in this case, by using the following command:

```
[root@rdbkvm4 ~]# vi med-dev-000d.xml
```

*Example 3-15 Mediated device XML*

---

```
<hostdev mode='subsystem' type='mdev' managed='no' model='vfio-ap'>
  <source>
    <address uuid='000d000d-000d-000d-000d-000d000d000d' />
  </source>
</hostdev>
```

---

In our environment, we run the KVM guest servers shown in Example 3-16:

*Example 3-16 Our KVM guest servers*

---

```
root@rdbkvm4 ~]# virsh list
 Id   Name      State
-----
 12   aw1       running
 13   aw2       running
 14   aw3       running
 15   bastion  running
 18   dnshcp    running
 19   iw1       running
 20   iw2       running
 21   iw3       running
 23   cp2       running
 25   cp1       running
```

---

Of these, we will choose the one named `bastion`, which runs Red Hat Enterprise Linux 9.2. Example 3-17 on page 47 shows our command to attach the device along with the output response indicating the device was attached successfully.

*Example 3-17 Attach the device*

---

```
[root@rdbkvm4 ~]# virsh attach-device bastion med-dev-000d.xml --live
Device attached successfully
```

---

Now we can log into the KVM guest server and check if the proper mapping has been done and the crypto domain is available. We verify that it is available by using the commands shown in Example 3-18:

*Example 3-18 Check the mappings*

---

```
[admin1@bastion ~]$ cd /sys/devices/ap/

[admin1@bastion ap]$ ls -l
total 0
drwxr-xr-x. 4 root root 0 Oct 4 07:16 card05
drwxr-xr-x. 4 root root 0 Oct 4 07:16 card07
.....
[admin1@bastion ap]$ cd card05
[admin1@bastion card05]$ ls -lad */
drwxr-xr-x. 3 root root 0 Oct 4 07:16 05.000d/
drwxr-xr-x. 2 root root 0 Oct 4 07:16 driver/
drwxr-xr-x. 2 root root 0 Oct 4 07:20 power/
drwxr-xr-x. 4 root root 0 Sep 23 13:37 subsystem/

[admin1@bastion card05]$ cd ../card07
[admin1@bastion card07]$ ls -lad */
drwxr-xr-x. 3 root root 0 Oct 4 07:16 07.000d/
drwxr-xr-x. 2 root root 0 Oct 4 07:16 driver/
drwxr-xr-x. 2 root root 0 Oct 4 07:21 power/
drwxr-xr-x. 4 root root 0 Sep 23 13:37 subsystem/
```

---

The directory entries in Example 3-18 show that the mapping was successful, and that the domain 000d (decimal: 13) can be used on both CEX cards 5 and 7.

### 3.1.6 Managing VFIO AP mediated devices with libvirt

The `libvirt` commands can be used to manage the life cycle of the mediated devices for the VFIO pass-through of cryptographic resources.

**Important:** Cryptographic adapter resources are managed as AP queues.

To make an AP queue suitable for VFIO traversal, it must be under the control of the `vfiu_ap` device driver. For example, AP queues must be made suitable for use by KVM guests (freed from the control of the `zcrypt` device driver).

We list all the CEX resources by using the command shown in Example 3-19, along with its output:

*Example 3-19 View CEX resources*

---

```
[root@rdbkvm4 ~]# lszcrypt -V
CARD.DOM TYPE MODE STATUS REQUESTS PENDING HWTYPE QDEPTH FUNCTIONS DRIVER
-----
00 CEX8C CCA-Coproc online 3939 0 14 08 S--D--NF- cex4card
```

---

00.000d	CEX8C	CCA-Coproc	unassigned	-	-	14	08	S--D--NF-	vfio_ap
00.0011	CEX8C	CCA-Coproc	unassigned	-	-	14	08	S--D--NF-	vfio_ap
01	CEX8A	Accelerator	online	0	0	14	08	-MC-A-NF-	cex4card
01.000d	CEX8A	Accelerator	unassigned	-	-	14	08	-MC-A-NF-	vfio_ap
01.0011	CEX8A	Accelerator	unassigned	-	-	14	08	-MC-A-NF-	vfio_ap
02	CEX8C	CCA-Coproc	online	2208	0	14	08	S--D--NF-	cex4card
02.000d	CEX8C	CCA-Coproc	unassigned	-	-	14	08	S--D--NF-	vfio_ap
02.0011	CEX8C	CCA-Coproc	unassigned	-	-	14	08	S--D--NF-	vfio_ap
03	CEX8A	Accelerator	online	0	0	14	08	-MC-A-NF-	cex4card
03.000d	CEX8A	Accelerator	unassigned	-	-	14	08	-MC-A-NF-	vfio_ap
03.0011	CEX8A	Accelerator	unassigned	-	-	14	08	-MC-A-NF-	vfio_ap
04	CEX8C	CCA-Coproc	online	2208	0	14	08	S--D--NF-	cex4card
04.000d	CEX8C	CCA-Coproc	unassigned	-	-	14	08	S--D--NF-	vfio_ap
04.0011	CEX8C	CCA-Coproc	unassigned	-	-	14	08	S--D--NF-	vfio_ap
05	CEX8P	EP11-Coproc	online	193	0	14	08	-----XNF-	cex4card
05.000d	CEX8P	EP11-Coproc	in use	-	-	14	08	-----XNF-	vfio_ap
05.0011	CEX8P	EP11-Coproc	assigned	-	-	14	08	-----XNF-	vfio_ap
06	CEX8C	CCA-Coproc	online	2208	0	14	08	S--D--NF-	cex4card
06.000d	CEX8C	CCA-Coproc	unassigned	-	-	14	08	S--D--NF-	vfio_ap
06.0011	CEX8C	CCA-Coproc	unassigned	-	-	14	08	S--D--NF-	vfio_ap
07	CEX8P	EP11-Coproc	online	243	0	14	08	-----XNF-	cex4card
07.000d	CEX8P	EP11-Coproc	in use	-	-	14	08	-----XNF-	vfio_ap
07.0011	CEX8P	EP11-Coproc	assigned	-	-	14	08	-----XNF-	vfio_ap

The AP queues with driver `vfio_ap` are eligible for a mediated device. In the sample output, these AP queues are from the 00-07 adapters, and on each adapter the 000d and 0011 domains which are shown like 00.000d, 00.0011.... 07.000d, 07.0011 in the `card.domain` column. These AP queues correspond to a matrix of adapters 00-07 with domains 000d and 0011.

Available cryptographic resources can also be listed by using `libvirt` commands. List all the available CEX cards and domain (queues) resources, shown in Example 3-20:

*Example 3-20 List devices*

```
[root@rdbkvm4 ~]# virsh nodedev-list --cap ap_card
ap_card00
ap_card01
ap_card02
ap_card03
ap_card04
ap_card05
ap_card06
ap_card07

[root@rdbkvm4 ~]# virsh nodedev-list --cap ap_queue
ap_00_000d
ap_00_0011
ap_01_000d
ap_01_0011
ap_02_000d
ap_02_0011
ap_03_000d
ap_03_0011
ap_04_000d
ap_04_0011
ap_05_000d
ap_05_0011
```

```
ap_06_000d
ap_06_0011
ap_07_000d
ap_07_0011
```

---

Use the `virsh nodedev-dumpxml` command to check and confirm that the AP queues are controlled by the `vfio_ap` device driver, as shown in Example 3-21:

*Example 3-21 Checking the domain 0011 (dec: 17) on card 01, and 03*

---

```
[root@rdbkvm4 ~]# virsh nodedev-dumpxml ap_01_0011
<device>
  <name>ap_01_0011</name>
  <path>/sys/devices/ap/card01/01.0011</path>
  <parent>ap_card01</parent>
  <driver>
    <name>vfio_ap</name>
  </driver>
  <capability type='ap_queue'>
    <ap-adapter>0x01</ap-adapter>
    <ap-domain>0x0011</ap-domain>
  </capability>
</device>
```

```
[root@rdbkvm4 ~]# virsh nodedev-dumpxml ap_03_0011
<device>
  <name>ap_03_0011</name>
  <path>/sys/devices/ap/card03/03.0011</path>
  <parent>ap_card03</parent>
  <driver>
    <name>vfio_ap</name>
  </driver>
  <capability type='ap_queue'>
    <ap-adapter>0x03</ap-adapter>
    <ap-domain>0x0011</ap-domain>
  </capability>
</device>
```

---

Use the following example to create a file for the node-device XML description file of the mediated device.

As a child element of the capabilities element, you can add attribute elements to the adapters and domains to configure which domain of which adapter (as a subset of the available AP queue matrix) you want to use. The values are the identifiers of the adapters and domains in hexadecimal form and prefixed with `0x`. For example, for the matrix of range 0011 on adapter 01 and 03, add three attribute elements, one for each adapter and one for the range.

As an additional child element of the capabilities element, add a unique userid (UUID) element specifying the UUID to be used for the mediated device.

Use the `uuidgen` command to obtain a UUID. Or, you can combine the output of `uuidgen` with a “talking part” (for example, an entry referring to the two adapters and the domain).

For example, the specification shown in Example 3-22 on page 50 will configure a stable UUID of `825b3872-0001-0003-0011-28a403f8b357` for the intermediary device.

*Example 3-22 vi 01-03-0011.xml*

---

```
<device>
  <parent>ap_matrix</parent>
  <capability type="mdev">
    <type id="vfio_ap-passthrough"/>
    <attr name="assign_adapter" value="0x01"/>
    <attr name="assign_adapter" value="0x03"/>
    <attr name="assign_domain" value="0x0011"/>
    <uuid>825b3872-0001-0003-0011-28a403f8b357</uuid>
  </capability>
</device>
```

---

After creating the definition, we create the mediated device by issuing the **virsh nodedev-define** command.

Example 3-23 assumes that the node-device XML file from Example 3-22 is located at `~/01-03-001.xml`.

*Example 3-23 Create the mediated device*

---

```
[root@rdbkvm4 ~]# virsh nodedev 01-03-0011.xml
Node device mdev_825b3872-0001-0003-0011-28a403f8b357_matrix defined from
~/01-03-001.xml
```

---

The tree view of the **virsh nodedev-list** command (shown in Example 3-24) shows that the mediated device corresponds to a matrix of AP queues. If the device is not yet activated, you need the **--all** option to include inactive devices in the command output.

*Example 3-24 Check the devices in tree view*

---

```
[root@rdbkvm4 ~]# virsh nodedev-list --tree --all
computer
|
+- ap_card00
|   +- ap_00_000d
|   +- ap_00_0011
|
+- ap_card01
|   +- ap_01_000d
|   +- ap_01_0011
|
+- ap_card02
|   +- ap_02_000d
|   +- ap_02_0011
|
+- ap_card03
|   +- ap_03_000d
|   +- ap_03_0011
...
+- ap_matrix
|   +- mdev_000d000d_000d_000d_000d_000d000d000d_matrix
```

```
| +- mdev_00110011_0011_0011_0011_001100110011_matrix  
| +- mdev_825b3872_0001_0003_0011_28a403f8b357_matrix  
.....
```

---

Use the **virsh nodedev-dumpxml** command to display the properties of the mediated device in node-device XML format, as shown in Example 3-25:

*Example 3-25 Display properties in XML format*

---

```
[root@rdbkvm4 ~]# virsh nodedev-dumpxml  
mdev_825b3872_0001_0003_0011_28a403f8b357_matrix  
<device>  
  <name>mdev_825b3872_0001_0003_0011_28a403f8b357_matrix</name>  
  <path>/sys/devices/vfio_ap/matrix/825b3872-0001-0003-0011-28a403f8b357</path>  
  <parent>ap_matrix</parent>  
  <driver>  
    <name>vfio_ap_mdev</name>  
  </driver>  
  <capability type='mdev'>  
    <type id='vfio_ap-passthrough' />  
    <uuid>825b3872-0001-0003-0011-28a403f8b357</uuid>  
    <parent_addr>matrix</parent_addr>  
    <iommuGroup number='11' />  
    <attr name='assign_adapter' value='0x01' />  
    <attr name='assign_adapter' value='0x03' />  
    <attr name='assign_domain' value='0x0011' />  
  </capability>  
</device>
```

---

The path element contains the sysfs path of the mediated device. Read the matrix attribute to display the matrix of AP queues, as shown in Example 3-26.

*Example 3-26 Reading the matrix attribute*

---

```
[root@rdbkvm4 ~]# cat  
/sys/devices/vfio_ap/matrix/825b3872-0001-0003-0011-28a403f8b357/matrix  
01.0011  
03.0011
```

---

For persistent mediated devices, you should activate the mediated device by issuing a **virsh nodedev-start** command, as shown in Example 3-27:

*Example 3-27 Start the device*

---

```
# virsh nodedev-start mdev_825b3872-0001-0003-0011-28a403f8b357_matrix  
Device mdev_825b3872-0001-0003-0011-28a403f8b357_matrix started
```

---

**Note:** Linux on IBM Z accesses cryptographic adapters through the zcrypt device driver and a generic device node. The cryptographic resources available through the device node depend on the configuration of the real or virtual hardware.

In the case of a KVM guest, a subset of the host's cryptographic resources may be assigned to a VFIO-mediated device, which is then transferred to the guest. VFIO-mediated devices are identified by a UUID.

On the guest, the cryptographic adapter resources are accessed through the general device node as usual. These resources do not require a guest device.

The remaining task is to configure the device as a VFIO mediated device that uses the **hostdev** element. You would do that by editing the `med-dev-01-03-0011.xml` file, as shown in Example 3-28:

*Example 3-28 vi med-dev-01-03-0011.xml*

---

```
<hostdev mode='subsystem' type='mdev' model='vfio-ap'>
  <source>
    <address uuid='825b3872-0001-0003-0011-28a403f8b357' />
  </source>
</hostdev>
```

---

Finally, attach the device to a guest by following our example command shown in Example 3-29:

*Example 3-29 Attach the device*

---

```
[root@rdbkkvm4 ~]# virsh attach-device bastion med-dev-01-03-0011.xml --live
Device attached successfully
```

---





## 3.2 Configuring z/VM guests to use CEX adapters

This section helps you to take advantage of the hardware cryptography support of IBM Z or IBM LinuxONE servers in z/VM environments, especially the security features associated with IBM Z Crypto-Express (CEX) adapters through in-kernel cryptography APIs and, for Linux on IBM Z, the libica cryptographic functions library. Using these features provides these benefits:

- ▶ File system encryption
- ▶ Configure z/VM guests to use CEX functionality
- ▶ Communication encryption (to applications such as IBM HTTP Server)

The way that z/VM provides this support is by granting access to the cryptographic resource, sometimes called the Adjunct Processor (AP) queue, domains to the z/VM guests. From a system implementation perspective, an AP of a CEX8S feature is one of its internal cryptography engines (cryptography coprocessor units). AP designates to the processor, while the adapter number (AP ID) specifies the number associated with it.

This chapter focuses on how to define encryption functions on the z/VM host machine and then provides help for setting up encryption on z/VM guest servers.

The chapter describes and discusses the following topics:

- ▶ Setting up and configuring CEX on a z/VM host
- ▶ Configuring z/VM guests to use CEX functionality

### 3.2.1 Setting up the z/VM host machine to use CEX functions

Once the z/VM environment has been installed and the necessary settings for using virtualization have been made, the server is ready to install z/VM guest servers on it. z/VM guest servers are able to use the tools that the z/VM host machine offers them.

To make the various encryption (CEX) tools and their functions available to the guest servers, after the basic z/VM configuration has been done, additional configuration steps need to be done on the z/VM host server.

In Chapter 2, “Overview of our environment” on page 11, we have shown how to configure CEX devices and the crypto domains on the HMC/DPM consoles of IBM Z/LinuxONE servers for each LPAR. On those interfaces, we not only had to make the device available but also specify which domains would be available in the LPAR to which we were attaching the CEX adapter(s).

### Crypto Express adapters configuration on HMC

Crypto Express adapters can be configured for each LPAR in one of the following operational modes via the HMC:

- ▶ Accelerator (clear key only).
- ▶ Common Cryptographic Architecture (coprocessor) (Clear key and secure key functions).
- ▶ Enterprise Public-Key Cryptographic Standards (PKCS) #11 Coprocessor/EP11 - Secure key functions through a PKCS#11 API only. An IBM Crypto Express adapter, which is configured with the Enterprise PKCS #11 (EP11) firmware, is called a Crypto Express EP11 coprocessor (in our case, referred to as CEX8P).

Figure 3-5 shows the Crypto Express adapter configuration for a z/VM LPAR.

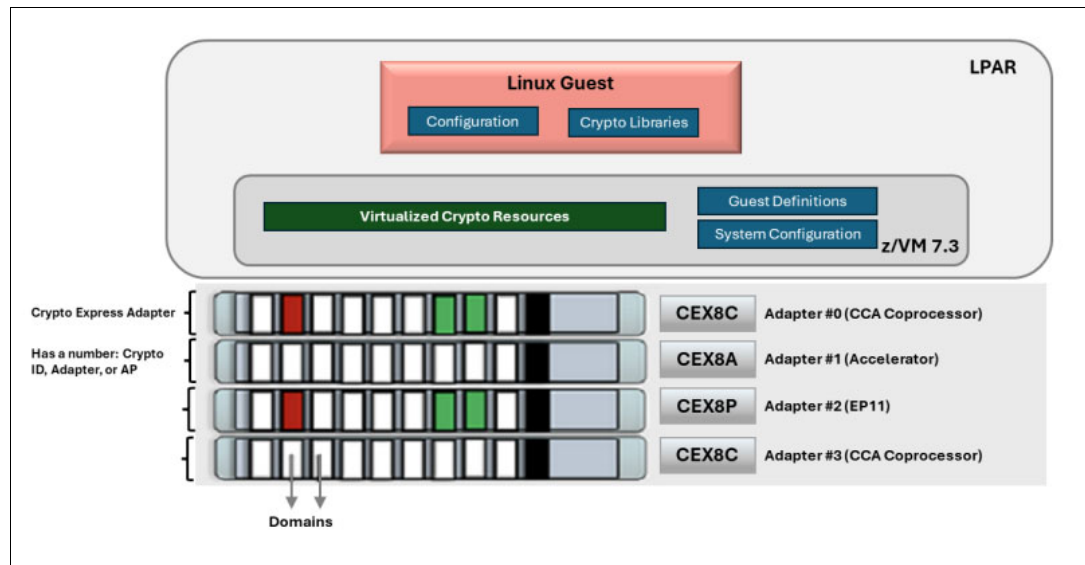


Figure 3-5 Crypto Adapter configuration for z/VM LPAR

In Figure 3-5, four CEX adapters are assigned to this z/VM LPAR by HMC:

- ▶ Adapters #000 and #003 are Crypto Express8 configured in CCA coprocessor mode
- ▶ Adapters #001 is Crypto Express8 configured in an accelerator mode
- ▶ Adapter #002 is Crypto Express8 configured in EP11 coprocessor mode

One *Domain* or *Crypto Resource* is a single logical piece of a Crypto Express adapter at a particular domain index.

Each Crypto Express adapter (AP) has 85 domains. One or more domains from crypto adapters are assigned to a z/VM LPAR. The z/VM host detects only those adapters and domains assigned to the z/VM LPAR as a virtual device represented by a crypto ID and a domain index (for example, ID 0 domain 1). Example 3-30 on page 55 shows the crypto adapters and domains assigned to a z/VM LPAR on a z/VM host, from the `query crypto domain users` command issued from CMS.

**Note:** In all examples, commands will be shown in **bold** text and the output of the commands are specified in plain text.

*Example 3-30 Query Crypto Express adapter and domain configuring on z/VM host*

```
query crypto domain users
22:23:07 AP 000 CEX8C Domain 020 operational online shared
22:23:07 AP 000 CEX8C Domain 021 operational online free
22:23:07 AP 001 CEX8P Domain 020 operational online free
22:23:07 AP 001 CEX8P Domain 021 operational online free
22:23:07 AP 002 CEX8C Domain 020 operational online shared
22:23:07 AP 002 CEX8C Domain 021 operational online free
22:23:07 AP 003 CEX8P Domain 020 operational online free
22:23:07 AP 003 CEX8P Domain 021 operational online free
22:23:07
22:23:07 There are no shared-crypto users.
Ready; T=0.01/0.01 22:23:07
```

In Example 3-30, these adapters (AP) #000 and #002 are configured as using CCA cryptographic coprocessor (CEX8C) mode and adapters #001 and #003 are configured as a PKCS #11 cryptographic coprocessor (CEX8P) mode

Domains 20 and 21 from Adapters #000, #001, #003 and #004 are assigned to this z/VM LPAR. These adapters and domains are available to the z/VM host.

**Note:** Adapter and domain pairs are shown in decimal notation.

For more information, see [Cryptographic domains](#).

Domains assigned to a z/VM LPAR are available to the z/VM host.

**Note:** In a z/VM environment,

- ▶ z/VM Guest virtual machines (VMs) can use Crypto resources on an adapter that is configured in accelerator or CCA coprocessor mode as dedicated or shared crypto resources.
- ▶ z/VM virtual machines can use Crypto resources on an adapter that is configured in EP11 coprocessor mode as only dedicated crypto resources.
- ▶ z/VM Guest uses Crypto libraries to use assigned Crypto resource. Crypto libraries will vary from OS to OS. Some may require a specific configuration to make use of certain features. Consult pertinent local documentation

## 3.2.2 Assigning crypto resources on z/VM systems

In a z/VM environment, Crypto Express adapters attached to your z/VM LPAR are virtualized for the benefit of z/VM guests. It is expected that the LPAR running z/VM has access to multiple AP queues. Each crypto resource that is available to a z/VM host is assigned to a z/VM guest (also known as a virtual machine) in one of the following categories:

1. **Shared queue support** (APVIRTual operand on the CRYPTO directory control statement)
  - Shared queue support allows you to assign multiple z/VM guests to a single pool of hypervisor-managed crypto resources.

This grants access to clear-key encryption, random number generation and digital signature operations in the Crypto Express adapters without the need to dedicate an entire AP queue to a z/VM guest.

- Only crypto resources that are configured as an accelerator or CCA coprocessor can be included in the shared pool.
- Crypto resources configured as an EP11 coprocessor cannot be added to the shared pool.
- All crypto resources in the shared pool must be the same type and mode. For example, if the first resource that is assigned to the shared pool is a CEX8 in accelerator mode, then all additional resources that are assigned to the shared pool must also be a CEX8 configured in accelerator mode.
- Virtual machines that have access to the shared pool of crypto resources are referred to as “APVIRT crypto” virtual machines.

2. **Dedicated queue support** (APDEDicated operand on the CRYPTO directory control statement)

- This resource is assigned to only one z/VM guest (virtual machine) for its exclusive use.
- Dedicated cryptographic resources are required if a guest workload requires secure key operations (such as the use of **dm-crypt** in Linux for file system encryption) or if cryptographic requirements or key materials are not allowed to be shared with other guests.
- Crypto resources that are configured as an accelerator, CCA coprocessor, or EP11 coprocessor can be dedicated to a virtual machine.
- Virtual machines that have dedicated crypto resources that are assigned are referred to as “APDED crypto” virtual machines.

3. **FREE** - This resource is not designated for any particular use. It is available to be assigned to the shared pool or to a virtual machine for its dedicated use.

Figure 3-6 shows the crypto resource assignment to z/VM guests.

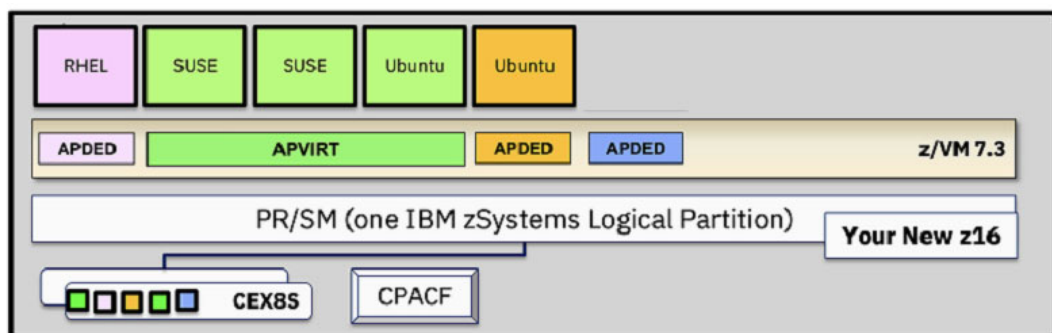


Figure 3-6 Crypto Resource assignment to z/VM Guests

In Figure 3-6, dedicated crypto resources are assigned to z/VM guests with RHEL and Ubuntu by using the APDED operand. Shared Crypto resources are assigned to a z/VM guest with SUSE and Ubuntu by using the APVIRT operand.

**Note:** In a z/VM environment, the following are some key considerations to assigning the crypto domains to z/VM guests:

- ▶ z/VM guests should not try to dedicate the same domains (first to IPL wins, all others complain).
- ▶ z/VM guests with a dedicated crypto resource may not be relocated.
- ▶ z/VM guest virtual machines can have access to the shared pool (APVIRT), or it can have dedicated resources (APDED), but not both.
- ▶ z/VM shared crypto resources are limited to clear-key (Accelerator) mode only.

The control program (CP) of the z/VM host identifies the assignment of a crypto resource with the values shown in bold in Example 3-31.

*Example 3-31 Check Crypto Express adapter and domain assignment at z/VM host level*

```

query crypto domain users
22:23:07 AP 000 CEX8C Domain 020 operational online shared
22:23:07 AP 000 CEX8C Domain 021 operational online free
22:23:07 AP 001 CEX8P Domain 020 operational online free
22:23:07 AP 001 CEX8P Domain 021 operational online free
22:23:07 AP 002 CEX8C Domain 020 operational online shared
22:23:07 AP 002 CEX8C Domain 021 operational online free
22:23:07 AP 003 CEX8P Domain 020 operational online free
22:23:07 AP 003 CEX8P Domain 021 operational online free
22:23:07
22:23:07 There are no shared-crypto users.
Ready; T=0.01/0.01 22:23:07

```

Crypto resources labeled as free are not in use.

The output of the **query crypto** command is explained in Table 3-3.

*Table 3-3 Crypto query output contains the following fields*

Attribute	Explanation
apnum	The apnum field indicates the three-digit crypto adapter number in decimal. Example: AP <b>001</b> CEX8P Domain 020 operational online free The apnum is 001
atype	The atype field indicates the crypto adapter type and mode. For dedicated resources, the value indicates the type and mode of the physical resource. For shared resources, the value indicates the maximal common subset of crypto express adapter capabilities that is available in the shared pools of all systems in the user's relocation domain that have the same mode. Shared crypto resources must be configured in accelerator or CCA-coprocessor mode. Example: AP 001 <b>CEX8P</b> Domain 020 operational online free The atype is CEX8P -> Crypto Express8 is configured in EP11 coprocessor mode.

Attribute	Explanation
domnum	<p>domnum is the three-digit domain number in decimal. If the resource is shared, then 001 is assigned to the virtual domain number. If the resource is dedicated, then the actual hardware domain number of the resource is assigned.</p> <p>Example: AP 001 CEX8P Domain 020 operational online free domnum is 020</p>
device_status	<p>device_status can be any of the following:</p> <p><b>operational</b> indicates that the crypto resource is installed and operational.</p> <p><b>checkstop</b> indicates that the crypto resource is in a checkstop condition and is unavailable.</p> <p><b>deconfigured</b> indicates that the adapter is deconfigured and unavailable. This could result from VARY OFF CRYPTO when the environment in which CP is running supports AP reconfiguration, or from an operation performed on the hardware maintenance console (HMC).</p> <p><b>busy</b> indicates that the adapter is temporarily busy initializing or doing error recovery.</p> <p><b>resetting</b> indicates that the resource is being reset.</p> <p><b>revoked</b> indicates the resource was detected by CP, but has since been unassigned from the configuration. If the resource is added back into the configuration, the updated status will be reported. Otherwise, when CP no longer detects this resource in the configuration, it will not be reported in Q CRYPTO output.</p> <p><b>unsupported</b> indicates the crypto resource status is unsupported by CP</p> <p>Example: AP 001 CEX8P Domain 020 operational online free device_status is operational</p>
config_status	<p>CP's logical view of the resource state, as controlled by VARY CRYPTO. Can be any of the following:</p> <p><b>online</b> indicates that the crypto resource is online and available for use.</p> <p><b>offline</b> indicates that the crypto resource is offline.</p> <p>Example: AP 001 CEX8P Domain 020 operational online free contig_status is online</p>

Attribute	Explanation
device_assignment	<p>device_assignment is can be any of the following:</p> <p><b>free, dedication planned</b>  indicates that the crypto resource is not in use, however, it has been specified on a CRYPTO APDED statement in the online user directory.</p> <p><b>attached to userid</b>  indicates that the crypto resource is dedicated to a logged on virtual machine.</p> <p><b>free</b>  indicates that the crypto resource is not in use.</p> <p><b>shared</b>  indicates that the crypto resource is attached to the system for shared use.</p> <p>Example:  AP 001 CEX8P Domain 020 operational online free  device assignment is free</p>

### 3.2.3 IBM Z operational keys: Clear, protected, or secure

In 1.2, “Cryptographic terms” on page 4, we define the terms clear key, protected key and secure key.

IBM cryptographic hardware supports three types of keys: clear key, secure key and protected key. This section focuses on how the hardware provides additional protection for secure keys. Understanding the difference between the three will help in designing the right cryptographic solutions and in determining the hardware requirements for the cryptographic work.

Figure 3-8 on page 62 shows the IBM Z operational key types, where they are located and how they are protected.

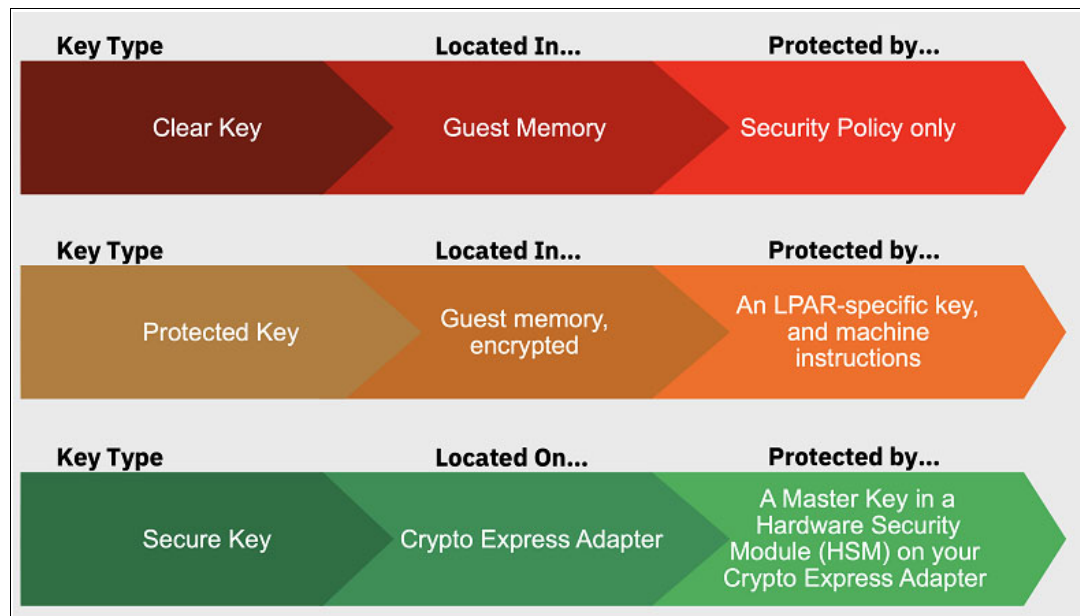


Figure 3-7 IBM Z operational key types

1.2.1, “Clear key versus secure key versus protected key” on page 4 outlines the differences between these keys.

## 3.3 Setup and configure Linux guests to use crypto resources

In this section, we describe how crypto resources can be assigned to z/VM guests in one of the following ways:

- ▶ Dynamic assignment (hotplug) or removal (hot unplug) of crypto resources to a z/VM guest.
- ▶ Persistence assignment or removal (persistence across z/VM host or guest reboots) of crypto resources to z/VM guests.

In our lab environment, BASTION and BASTION2 are z/VM guests on Linux operating systems.

### 3.3.1 Dynamic assignment of crypto resources to z/VM guests

Dynamic crypto support enables changes to the z/VM crypto environment without requiring an IPL of z/VM or its guests (for example, a Linux guest on IBM Z). Dynamic crypto assignment to a z/VM guest persists until the next z/VM guest start or z/VM host restart.

Dynamic crypto resource assignment provides the following benefits:

- ▶ Less disruptive addition or removal of Crypto Express hardware to or from a z/VM system and its guests.
- ▶ Less disruptive maintenance and repair of Crypto Express hardware attached and in use by a z/VM system.
- ▶ Re-assignment and allocation of crypto resources without requiring a system IPL or user log off or log on.
- ▶ Greater flexibility to change crypto resources between shared and dedicated use.
- ▶ The following are reliability, availability and serviceability (RAS) benefits for shared-use crypto resources:
  - Better detection of Crypto Express Adapter errors with “silent” retrying of shared pool requests to alternate resources.
  - Ability to recover failed Crypto Express adapters.
  - Improved internal diagnostics for IBM service.
  - Improved logoff and live guest relocation latency for users of shared crypto.

The following are a list of z/VM dynamic crypto commands that are executed at the z/VM host level:

- ▶ Bring a Crypto Express Adapter online

Example 3-32 shows how to make new Crypto Express adapter available to the z/VM host.

*Example 3-32 Configure Crypto Express adapter #003 online*

---

**VARY ON CRYPTO AP 003**  
Crypto AP 003 varied online.

---

- ▶ Take a Crypto Express adapter offline (device associations remain in place)



Example 3-33 shows how to deconfigure an assigned Crypto Express adapter.

*Example 3-33 Deconfigure Crypto Express adapter #002 offline*

---

```
VARY OFF CRYPTO AP 002
Crypto AP 002 varied offline.
```

---

The adapter will be listed as offline, and will not be available for use.

Use **VARY ON** to bring the adapter back online to an active configuration.

- ▶ Assign crypto resource(s) to z/VM guest (or APVIRT)

Example 3-34 shows how to assign a crypto resource to the z/VM guest, BASTION (Dedicated mode).

*Example 3-34 Assign crypto resource to z/VM guest bastion (Dedicated mode)*

---

```
ATTACH CRYPTO AP 2 to BASTION
```

---

This does not change your z/VM User Directory. A static configuration does not update automatically.

- ▶ Assign/reassign shared crypto resource access to a z/VM guest.

Example 3-35 shows how to assign new crypto resources for sharing.

*Example 3-35 Assign new crypto resource in shared mode*

---

```
VARY ON CRYPTO 3
ATTACH CRYPTO AP 3 DOMAIN 30 31 to SYSTEM
```

---

- ▶ Remove crypto resources from the z/VM Guest

Example 3-36 provides the command to configure new crypto adapter, #003, and assign domain 30 and 31 of adapter #003 in shared mode.

*Example 3-36 Remove shared crypto resources from a shared pool*

---

```
DETACH CRYPTO AP 3 DOMAIN 30 31 from SYSTEM FORCE
Crypto AP 003 Domain 30 031 detached from SYSTEM
```

---

- ▶ Query crypto domain users.

When the **USERS** operand is specified after the **DOMAIN** operand, the users enabled for **CRYPTO APVIRTUAL** are listed.

Example 3-37 shows available crypto resources on a z/VM host.

*Example 3-37 Query crypto resources on z/VM host*

---

```
QUERY CRYPTO DOMAIN USERS
22:23:07 AP 000 CEX8C Domain 020 operational online shared
22:23:07 AP 000 CEX8C Domain 021 operational online free
22:23:07 AP 001 CEX8P Domain 020 operational online free
22:23:07 AP 001 CEX8P Domain 021 operational online free
22:23:07 AP 002 CEX8C Domain 020 operational online shared
22:23:07 AP 002 CEX8C Domain 021 operational online free
22:23:07 AP 003 CEX8P Domain 020 operational online free
22:23:07 AP 003 CEX8P Domain 021 operational online free
22:23:07
22:23:07 There are no shared-crypto users.
Ready; T=0.01/0.01 22:23:07
```

---

**Note:**

- ▶ Attachments persist even when a device is taken offline.
- ▶ Resource assignment (dedicated/shared) does not change when an adapter is varied on/off.

### 3.3.2 Crypto resource assignment to z/VM guests for dedicated use

Crypto resources can be assigned to z/VM guests in exclusive mode. Crypto resources that are configured as an Accelerator, CCA coprocessor, or EP11 coprocessor can be dedicated to a virtual machine.

Virtual machines that have dedicated crypto resources that are assigned are referred to as “APDED crypto” virtual machines.

Figure 3-8 shows our lab environment’s crypto resource assignment to z/VM guests for dedicated use.

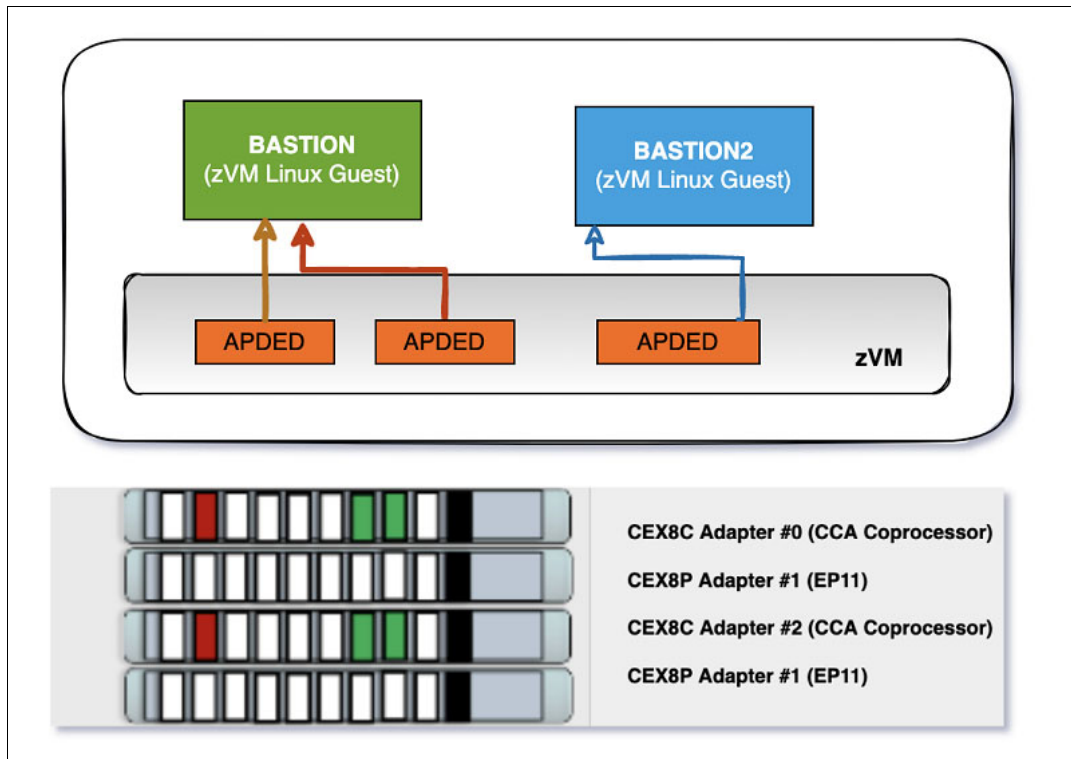


Figure 3-8 Crypto Resource assignment to z/VM guests in dedicated mode

Example 3-38 shows the dynamic assignment of Crypto domain 21 from adapters #0 and #3 to “BASTION” guest VM for dedicated use. The command shown in this example is issued from the z/VM host.

Example 3-38 Crypto domain assignment

```
ATTACH CRYPTO AP 0 3 DOMAIN 21 to BASTION
08:56:21 Crypto AP 000 Domain 021 attached to BASTION.
08:56:21 Crypto AP 003 Domain 021 attached to BASTION.
Ready; T=0.01/0.01 08:56:21
```

---

Example 3-39 shows the assignment of crypto domain 21 from adapter #1 (001) to “BASTION2” guest for dedicated use. The command shown in this example is issued from the z/VM LPAR.

*Example 3-39 Crypto domain assignment*

---

```
ATTACH CRYPTO AP 001 DOMAIN 21 to BASTION2
08:58:37 Crypto AP 001 Domain 021 attached to BASTION2.
Ready; T=0.01/0.01 08:58:37
```

---

Example 3-40 shows the crypto resources assigned to z/VM guests. The command shown in this example was issued from the z/VM host.

*Example 3-40 Check the crypto domains assigned to BASTION and BASTION2*

---

```
QUERY CRYPTO DOMAIN USERS
08:59:06 AP 000 CEX8C Domain 020 operational online shared
08:59:06 AP 000 CEX8C Domain 021 operational online attached to BASTION
08:59:06 AP 001 CEX8P Domain 020 operational online free
08:59:06 AP 001 CEX8P Domain 021 operational online attached to BASTION2
08:59:06 AP 002 CEX8C Domain 020 operational online shared
08:59:06 AP 002 CEX8C Domain 021 operational online free
08:59:06 AP 003 CEX8P Domain 020 operational online free
08:59:06 AP 003 CEX8P Domain 021 operational online attached to BASTION
08:59:06
08:59:06 There are no shared-crypto users.
Ready; T=0.01/0.01
08:59:08
```

---

The following commands query the crypto assignment on z/VM guests. Note that adapter and domain pairs are shown in hexadecimal notation.

Example 3-41 shows the crypto assignment to the z/VM guest, BASTION. For this example, you would SSH to BASTION and run the **lszcrypt** command.

*Example 3-41 Check the crypto assignment on z/VM guest BASTION*

---

```
[root@bastion ~]# hostname
bastion
[root@bastion ~]# uname -a
Linux bastion 5.14.0-162.6.1.el9_1.s390x #1 SMP Fri Sep 30 09:42:53 EDT 2022 s390x s390x s390x
GNU/Linux
[root@bastion ~]# lszcrypt
CARD.DOM TYPE  MODE.      STATUS. REQUESTS
-----
00          CEX8C CCA-Coproc  online     1
00.0015    CEX8C CCA-Coproc  online     1
03          CEX8P EP11-Coproc  online     0
00.0015    CEX8P EP11-Coproc  online     0
[root@bastion ~]#
```

---

Example 3-42 on page 64 shows the crypto assignment to z/VM guest, BASTION2. For this example, you would SSH to BASTION2 and run the **lszcrypt** command.

Example 3-42 Check the crypto assignment on z/VM guest BASTION2

```
[root@bastion2 ~]# hostname
bastion2
[root@bastion2 ~]# uname -a
Linux bastion2 5.14.0-162.6.1.el9_1.s390x #1 SMP Fri Sep 30 09:42:53 EDT 2022 s390x
s390x s390x GNU/Linux
[root@bastion2 ~]# lszcrypt
CARD.DOM TYPE MODE STATUS REQUESTS
-----
01 CEX8P EP11-Coproc online 0
01.0015 CEX8P EP11-Coproc online 0
```

### 3.3.3 Removing dedicated crypto resources from z/VM guests

Crypto resource assigned to z/VM guests in exclusive mode can also be removed (detached) from z/VM guests. Figure 3-9 shows the detachment of crypto resources from z/VM guests.

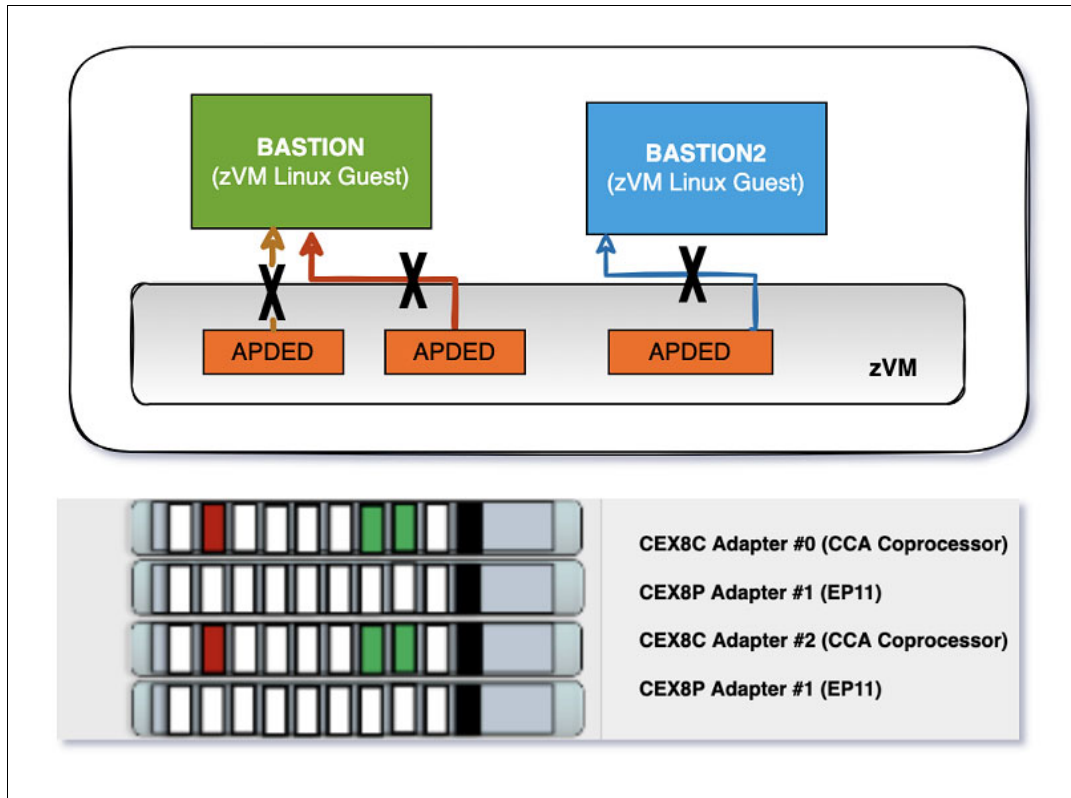


Figure 3-9 Detachment of crypto resources from z/VM guests

Example 3-43 shows the detachment of crypto domain 21 from adapters 00 and 03 from z/VM guest, BASTION.

Example 3-43 Detach crypto domain 21 from BASTION z/VM guest

```
DETACH CRYPTO AP 00 03 from BASTION
13:34:06 Crypto AP 000 Domain 021 detached from BASTION
13:34:06 Crypto AP 003 Domain 021 detached from BASTION
```

Ready; T=0.01/0.01 13:34:06

---

Example 3-44 shows the detachment of crypto domain 21 from adapter 01 of z/VM guest, BASTION2.

*Example 3-44 Detach Crypto domain 21 from bastion2 z/VM Guest*

---

**DETACH CRYPTO AP 01 from BASTION2**

13:35:33 Crypto AP 001 Domain 021 detached from BASTION2

Ready; T=0.01/0.01 13:35:33

---

Example 3-45 shows the command to query the crypto resources after detached from z/VM guests.

*Example 3-45 Query the crypto resources after detached from z/VM guest*

---

**QUERY CRYPTO DOMAIN USERS**

13:35:56	AP 000	CEX8C	Domain 020	operational	online	shared
13:35:56	AP 000	CEX8C	Domain 021	operational	online	free
13:35:56	AP 001	CEX8P	Domain 020	operational	online	free
13:35:56	AP 001	CEX8P	Domain 021	operational	online	free
13:35:56	AP 002	CEX8C	Domain 020	operational	online	shared
13:35:56	AP 002	CEX8C	Domain 021	operational	online	free
13:35:56	AP 003	CEX8P	Domain 020	operational	online	free
13:35:56	AP 003	CEX8P	Domain 021	operational	online	free

13:35:56

13:35:56 There are no shared-crypto users.

Ready; T=0.01/0.01 13:35:56

---

Example 3-46 shows the command to query the crypto resources on z/VM guest BASTION after crypto resource detachment.

*Example 3-46 Check the crypto resources on z/VM guest BASTION*

---

```
[root@bastion ~]# hostname
bastion
[root@bastion ~]# lszcrypt
lszcrypt: No crypto card devices found.
[root@bastion ~]#
```

---

Example 3-47 shows the command to query the crypto resources on z/VM guest BASTION2 after crypto resource detachment.

*Example 3-47 Check the crypto resources on z/VM guest BASTION2*

---

```
[root@bastion2 ~]# hostname
bastion2
[root@bastion2 ~]# lszcrypt
lszcrypt: No crypto card devices found.
[root@bastion2 ~]#
```

---

### 3.3.4 Persistence across z/VM host or guest reboots

You can set up your cryptographic resources such that, after a z/VM host or guest reboot, your crypto resources are ready to be attached to z/VM guests without further configuration steps. This set up requires the **CRYPTO APVIRTUAL** statement to enable access to shared crypto resources or the **CRYPTO APDEDICATED** command to enable access to dedicated crypto resources statement entry in the z/VM guest's user directory.

With this statement in the user directory, the virtual machine is given access to the shared or dedicated crypto resources at login. Run the **dirmaint (dirm)** command to add or remove crypto entries to the user directory of z/VM guests.

**dirmaint** requires the z/VM host or guest to be rebooted or logged off for the assignment or removal of crypto resources to take effect. See Chapter 5 of [z/VM CP Planning and Administration, SC24-6271](#) for more information on the use of the **CRYPTO** statement.

The CRYPTO user directory statement grants a z/VM userid access to crypto resources associated with the Crypto Express adapters. The format for this command is shown in Example 3-48:

*Example 3-48* CRYPTO user directory statement

---

```
CRYPTO+-+DOMAIN---+---domains+-+APDEDicated+-+AP
      |_____APVIRTua1
```

---

#### z/VM guests with shared-queue support

For a Linux guest requiring access to clear key cryptography operations, shared access to AP queues is the preferred method for implementation. In this case, the **CRYPTO** statement in the user directory entry for the guest needs to indicate the desire for access to virtual queues. No domain or AP queue need be specified. The Linux guest receives one virtualized card and one random virtual queue on a randomly chosen virtual AP. The AP number and domain are selected by z/VM and are not identical to those of the z/VM LPAR.

**Note:** You can now specify a **CRYPTO APVIRT** statement in your system configuration file which allows the system administrator to designate particular AP domains that are attached to the LPAR as “Reserved for APVIRT”.

For this support, z/VM uses all available AP queues, which are not dedicated to other guests, and these are shared between all guests that use the shared support. If multiple AP types are available for z/VM, then z/VM chooses the best AP type for acceleration of the Linux guest. When a type is selected, z/VM routes all cryptography requests from the guest to however many queues or cards of that type are available. The statement in the directory looks similar to **CRYPTO APVIRT**.

The AP queue number and the domain number, which z/VM provides to these two guests, are virtual numbers and do not correspond to the “real” domains and APs, which z/VM uses to run the cryptography requests of these guests.

#### Example: Assigning shared crypto resources to z/VM guests BASTION and BASTION2

Figure 3-10 on page 67 shows the shared crypto resources assignment to z/VM Guests BASTION and BASTION2

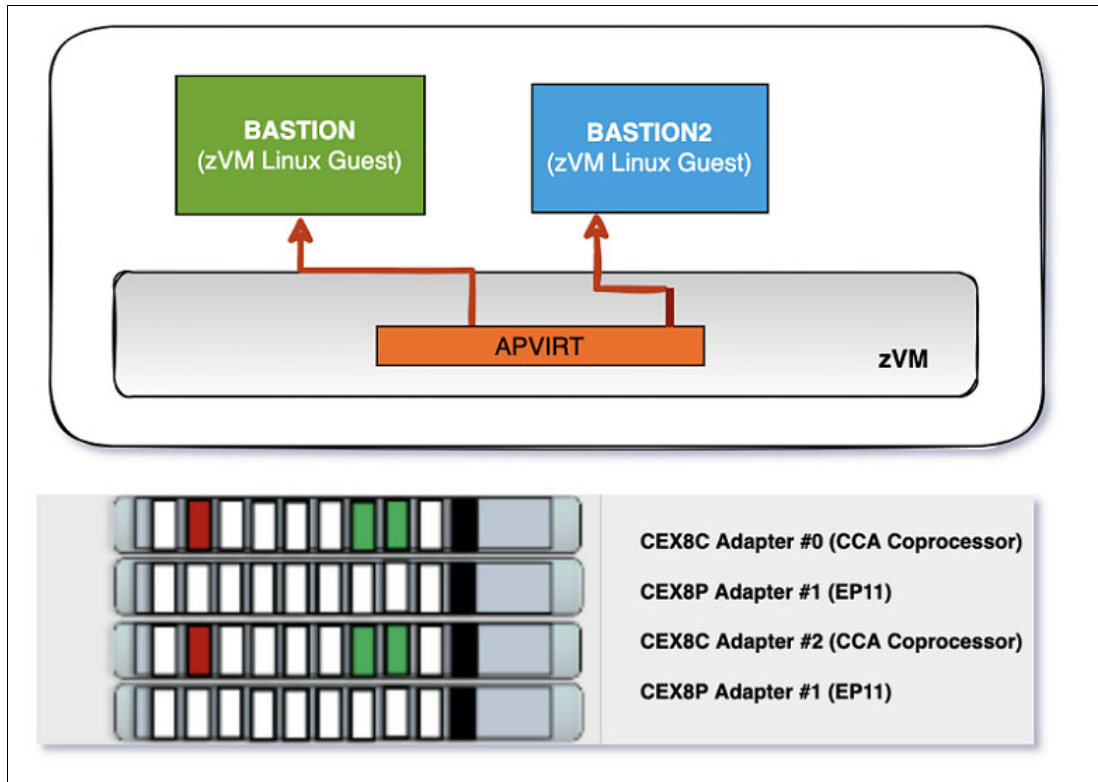


Figure 3-10 Shared crypto resource assignment to z/VM guests BASTION and BASTION2

Example 3-49 illustrates the command used to assign shared crypto domains to the BASTION z/VM guest. Issue this command on the z/VM host.

*Example 3-49 Crypto domain assignment to BASTION z/VM guest*

**DIRM FOR BASTION CRYPTO APVIRT**

```
DVHXMT1191I YOUR CRYPTO request has been sent for processing to DIRMAINT
DVHXMT1191I at RDBKZVMA via DIRMSAT4.
Ready; T=0.01/0.01 05:02:05
DVHREQ2288I Your CRYPTO request for BASTION at *
DVHREQ2288I has been accepted.
DVHBIV3450I The source for directory entry
DVHBIV3450I BASTION has been updated.
DVHBIV3203W Unable to notify
DVHBIV3203W ASYNCHRONOUS_UPDATE_NOTIFICATION_EXIT
DVHBIV3203W recipient of directory update.
DVHBIV3203W Recipient is unreachable.
DVHBIV3203W Unable to notify
DVHBIV3203W ASYNCHRONOUS_UPDATE_NOTIFICATION_EXIT
DVHBIV3203W recipient of directory update.
DVHBIV3203W Recipient is unreachable.
DVHBIV3203W Unable to notify
DVHBIV3203W ASYNCHRONOUS_UPDATE_NOTIFICATION_EXIT
DVHBIV3203W recipient of directory update.
DVHBIV3203W Recipient is unreachable.
DVHBIV3424I The next ONLINE will take place
DVHBIV3424I immediately.
DVHDRC3451I The next ONLINE will take place
```

```
DVHDRC3451I the delta object directory.  
DVHRLA3891I Your DSATCTL request has been relayed  
DVHRLA3891I for processing.  
DVHRLA3891I Your DSATCTL request has been relayed  
DVHRLA3891I for processing.  
DVHRLA3891I Your DSATCTL request has been relayed  
DVHRLA3891I for processing.  
DVHRLA3891I Your DMVCTL request has been relayed  
DVHRLA3891I for processing.  
DVHRLA3891I Your DMVCTL request has been relayed  
DVHRLA3891I for processing.  
DVHBIV3428I changes made to directory entry BASTION  
DVHBIV3428I have been placed online.  
DVHREQ2289I your CRYPTO request for BASTION at *  
DVHREQ2289I has completed; with RC=0
```

---

Example 3-50 shows the directory entry with shared cryptography queues.

*Example 3-50 Directory entry with shared cryptography queues*

---

```
USER BASTION xxxxxx 256M 1G G  
INCLUDE IBMDFLT  
IPL CMS  
MACH XA  
NICDEF C200 TYPE QDIO LAN SYSTEM VSWITCH1  
CRYPTO APVIRT
```

---

Example 3-51 illustrates the command used to assign shared crypto domains to the BASTION2 z/VM guest. Issue this command on the z/VM host:

*Example 3-51 Crypto domain assignment to BASTION2 z/VM guest*

---

```
DIRM FOR BASTION2 CRYPTO APVIRT  
DVHXMT1191I YOUR CRYPTO request has been sent for processing to DIRMAINT  
DVHXMT1191I at RDBKZVMA via DIRMSAT4.  
Ready; T=0.01/0.01 06:35:12  
DVHREQ2288I Your CRYPTO request for BASTION2 at *  
DVHREQ2288I has been accepted.  
DVHBIV3450I The source for directory entry  
DVHBIV3450I BASTION2 has been updated.  
DVHBIV3203W Unable to notify  
DVHBIV3203W ASYNCHRONOUS_UPDATE_NOTIFICATION_EXIT  
DVHBIV3203W recipient of directory update.  
DVHBIV3203W Recipient is unreachable.  
DVHBIV3203W Unable to notify  
DVHBIV3203W ASYNCHRONOUS_UPDATE_NOTIFICATION_EXIT  
DVHBIV3203W recipient of directory update.  
DVHBIV3203W Recipient is unreachable.  
DVHBIV3203W Unable to notify  
DVHBIV3203W ASYNCHRONOUS_UPDATE_NOTIFICATION_EXIT  
DVHBIV3203W recipient of directory update.  
DVHBIV3203W Recipient is unreachable.  
DVHBIV3424I The next ONLINE will take place  
DVHBIV3424I immediately.  
DVHDRC3451I The next ONLINE will take place  
DVHDRC3451I the delta object directory.
```



```

DVHRLA3891I Your DSATCTL request has been relayed
DVHRLA3891I for processing.
DVHRLA3891I Your DSATCTL request has been relayed
DVHRLA3891I for processing.
DVHRLA3891I Your DSATCTL request has been relayed
DVHRLA3891I for processing.
DVHRLA3891I Your DMVCTL request has been relayed
DVHRLA3891I for processing.
DVHRLA3891I Your DMVCTL request has been relayed
DVHRLA3891I for processing.
DVHBIV3428I changes made to directory entry BASTION2
DVHBIV3428I have been placed online.
DVHREQ2289I your CRYPTO request for BASTION2 at *
DVHREQ2289I has completed; with RC=0

```

---

Example 3-52 shows the directory entry with shared cryptography queues.

*Example 3-52 Directory entry with shared cryptography queues*

---

```

USER BASTION2 xxxxxx 256M 1G G
INCLUDE IBMDFLT
IPL CMS
MACH XA
NICDEF C200 TYPE QDIO LAN SYSTEM VSWITCH1
CRYPTO APVIRT

```

---

Example 3-53 shows the command used to display the crypto resources after the assignment (attachment) of the shared resource to z/VM guests. Issue this command on the z/VM host.

*Example 3-53 Query the crypto resources after shared crypto attachment to z/VM guests*

---

```

QUERY CRYPTO DOMAIN USERS
06:47:44 AP 000 CEX8C   Domain 020 operational   online   shared
06:47:44 AP 000 CEX8C   Domain 021 operational   online   free
06:47:44 AP 001 CEX8P   Domain 020 operational   online   free
06:47:44 AP 001 CEX8P   Domain 021 operational   online   free
06:47:44 AP 002 CEX8C   Domain 020 operational   online   shared
06:47:44 AP 002 CEX8C   Domain 021 operational   online   free
06:47:44 AP 003 CEX8P   Domain 020 operational   online   free
06:47:44 AP 003 CEX8P   Domain 021 operational   online   free
06:47:44
06:47:44 Shared-Crypto Users:
06:47:44 BASTION2 BASTION
Ready; T=0.01/0.01 06:47:44

```

---

Example 3-54 shows the command to display the crypto assignment on the z/VM guest BASTION. You would SSH to BASTION to check the crypto assignment.

*Example 3-54 Check the crypto assignment on z/VM guest BASTION*

---

```

[root@bastion ~]# hostname
bastion
[root@bastion ~]# uname -a

```

```
Linux bastion 5.14.0-162.6.1.e19_1.s390x #1 SMP Fri Sep 30 09:42:53 EDT 2022 s390x
s390x s390x GNU/Linux
[root@bastion ~]# lszcrypt
CARD.DOM TYPE      MODE.              STATUS. REQUESTS
-----
01                CEX8C CCA-Coproc  online           1
01.001           CEX8C CCA-Coproc  online           1
```

Example 3-54 on page 69 shows the command to display the crypto assignment on the z/VM guest BASTION2. SSH to BASTION2 to run this command.

*Example 3-55 Check the crypto assignment on z/VM guest BASTION2*

```
[root@bastion2 ~]# hostname
bastion2
[root@bastion2 ~]# uname -a
Linux bastion2 5.14.0-162.6.1.e19_1.s390x #1 SMP Fri Sep 30 09:42:53 EDT 2022
s390x s390x s390x GNU/Linux
[root@bastion2 ~]# lszcrypt
CARD.DOM TYPE      MODE.              STATUS. REQUESTS
-----
01                CEX8C CCA-Coproc  online           1
01.001           CEX8C CCA-Coproc  online           1
```

### Example: Removing the shared crypto resources from z/VM guests

Figure 3-11 shows the detachment of a shared crypto resource from z/VM guests BASTION and BASTION2.

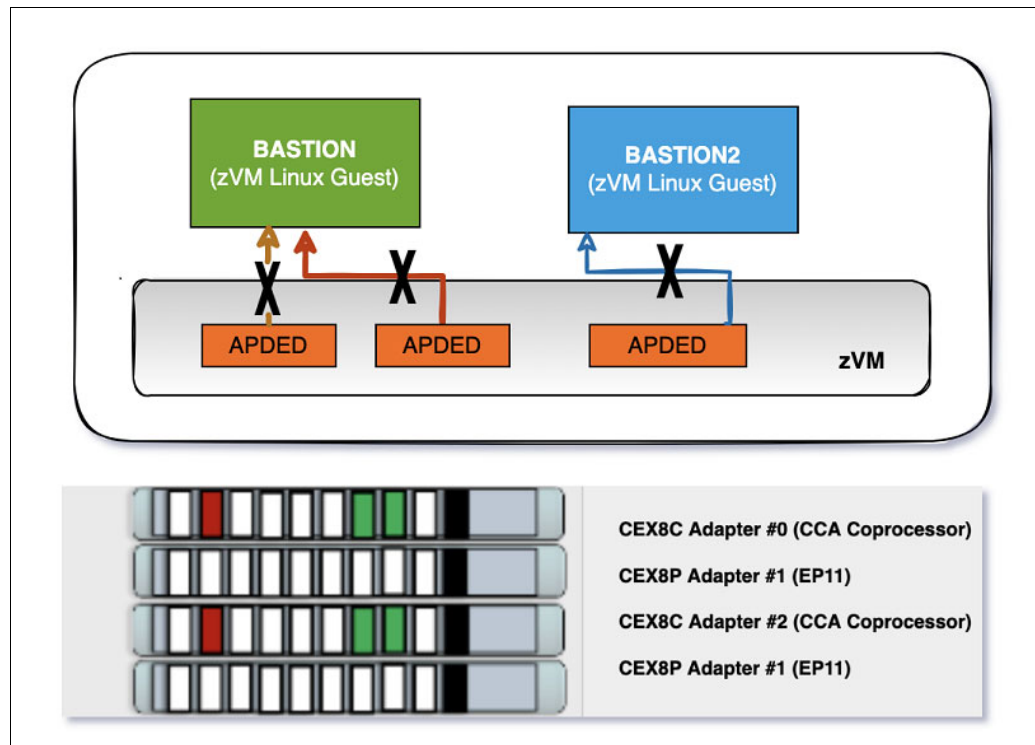


Figure 3-11 Detachment of shared crypto resources from z/VM guests

Example 3-56 shows the detachment of shared crypto resources from z/VM guest BASTION. Issue this command on the z/VM host:

*Example 3-56 Detach shared crypto resources of z/VM guest*

---

**DIRM FOR BASTION CRYPTO DELETE**

DVHXMT1191I Your CRYPTO request has been sent for processing to DIRMAINT  
DVHXMT1191I at RDBKZXVMA via DIRMSAT4.

Ready; T=0.01/0.01 19:45:17

DVHREQ2288I Your CRYPTO request for BASTION at \*  
DVHREQ2288I has been accepted.  
DVHBIV3450I The source for directory entry  
DVHBIV3450I BASTION has been updated.  
DVHBIV3203W Unable to notify  
DVHBIV3203W ASYNCHRONOUS\_UPDATE\_NOTIFICATION\_EXIT  
DVHBIV3203W recipient of directory update.  
DVHBIV3203W Recipient is unreachable.  
DVHBIV3203W Unable to notify  
DVHBIV3203W ASYNCHRONOUS\_UPDATE\_NOTIFICATION\_EXIT  
DVHBIV3203W recipient of directory update.  
DVHBIV3203W Recipient is unreachable.  
DVHBIV3203W Unable to notify  
DVHBIV3203W ASYNCHRONOUS\_UPDATE\_NOTIFICATION\_EXIT  
DVHBIV3203W recipient of directory update.  
DVHBIV3203W Recipient is unreachable.  
DVHBIV3424I The next ONLINE will take place  
DVHBIV3424I immediately.  
DVHDRC3451I The next ONLINE will take place  
DVHDRC3451I the delta object directory.  
DVHRLA3891I Your DSATCTL request has been relayed  
DVHRLA3891I for processing.  
DVHRLA3891I Your DSATCTL request has been relayed  
DVHRLA3891I for processing.  
DVHRLA3891I Your DSATCTL request has been relayed  
DVHRLA3891I for processing.  
DVHRLA3891I Your DMVCTL request has been relayed  
DVHRLA3891I for processing.  
DVHRLA3891I Your DMVCTL request has been relayed  
DVHRLA3891I for processing.  
DVHBIV3428I changes made to directory entry BASTION  
DVHBIV3428I have been placed online.  
DVHREQ2289I your CRYPTO request for BASTION at \*  
DVHREQ2289I has completed; with RC=0

---

Example 3-57 shows the detachment of shared crypto resources from z/VM Guest BASTION2. Issue the command shown in this example of the z/VM host:

*Example 3-57 Detach shared crypto resources from z/VM guest BASTION2*

---

**DIRM FOR BASTION2 CRYPTO DELETE**

DVHXMT1191I Your CRYPTO request has been sent for processing to DIRMAINT  
DVHXMT1191I at RDBKZXVMA via DIRMSAT4.

Ready; T=0.01/0.01 11:35:47

DVHREQ2288I Your CRYPTO request for BASTION2 at \*  
DVHREQ2288I has been accepted.

```

DVHBIV3450I The source for directory entry
DVHBIV3450I BASTION2 has been updated.
DVHBIV3203W Unable to notify
DVHBIV3203W ASYNCHRONOUS_UPDATE_NOTIFICATION_EXIT
DVHBIV3203W recipient of directory update.
DVHBIV3203W Recipient is unreachable.
DVHBIV3203W Unable to notify
DVHBIV3203W ASYNCHRONOUS_UPDATE_NOTIFICATION_EXIT
DVHBIV3203W recipient of directory update.
DVHBIV3203W Recipient is unreachable.
DVHBIV3203W Unable to notify
DVHBIV3203W ASYNCHRONOUS_UPDATE_NOTIFICATION_EXIT
DVHBIV3203W recipient of directory update.
DVHBIV3203W Recipient is unreachable.
DVHBIV3424I The next ONLINE will take place
DVHBIV3424I immediately.
DVHDRC3451I The next ONLINE will take place
DVHDRC3451I the delta object directory.
DVHRLA3891I Your DSATCTL request has been relayed
DVHRLA3891I for processing.
DVHRLA3891I Your DSATCTL request has been relayed
DVHRLA3891I for processing.
DVHRLA3891I Your DSATCTL request has been relayed
DVHRLA3891I for processing.
DVHRLA3891I Your DMVCTL request has been relayed
DVHRLA3891I for processing.
DVHRLA3891I Your DMVCTL request has been relayed
DVHRLA3891I for processing.
DVHBIV3428I changes made to directory entry BASTION2
DVHBIV3428I have been placed online.
DVHREQ2289I your CRYPTO request for BASTION2 at *
DVHREQ2289I has completed; with RC=0

```

---

***Check Crypto resources from z/VM HOST after detaching the shared resource***

To check the crypto resources from the z/VM host after detaching the shared resource, issue the command shown in Example 3-58 on the z/VM host.

*Example 3-58 Query shared Crypto resources*

---

```

QUERY CRYPTO DOMAIN USERS
14:35:56 AP 000 CEX8C   Domain 020 operational  online  shared
14:35:56 AP 000 CEX8C   Domain 021 operational  online   free
14:35:56 AP 001 CEX8P   Domain 020 operational  online   free
14:35:56 AP 001 CEX8P   Domain 021 operational  online   free
14:35:56 AP 002 CEX8C   Domain 020 operational  online  shared
14:35:56 AP 002 CEX8C   Domain 021 operational  online   free
14:35:56 AP 003 CEX8P   Domain 020 operational  online   free
14:35:56 AP 003 CEX8P   Domain 021 operational  online   free
14:35:56
14:35:56 There are no shared-crypto users.
Ready; T=0.01/0.01  14:35:56

```

---

### Query shared crypto resources on z/VM guest BASTION

Use the command shown in Example 3-59 to show any shared crypto resources on z/VM guest BASTION. SSH to bastion to check the crypto assignments.

*Example 3-59 Query shared crypto resources on z/VM guest BASTION*

```
[root@bastion ~]# hostname
bastion
[root@bastion ~]# lszcrypt
lszcrypt: No crypto card devices found.
[root@bastion ~]#
```

Use the command shown in Example 3-60 to show any shared crypto resources on z/VM guest BASTION2. SSH to bastion to check the crypto assignments.

*Example 3-60 Query shared crypto resources from z/VM guest BASTION2*

```
[root@bastion2 ~]# hostname
bastion2
[root@bastion2 ~]# lszcrypt
lszcrypt: No crypto card devices found.
[root@bastion2 ~]#
```

### Example: Assigning crypto domains to z/VM guests for dedicated use

Figure 3-12 shows crypto domain assignments to z/VM guests for dedicated use.

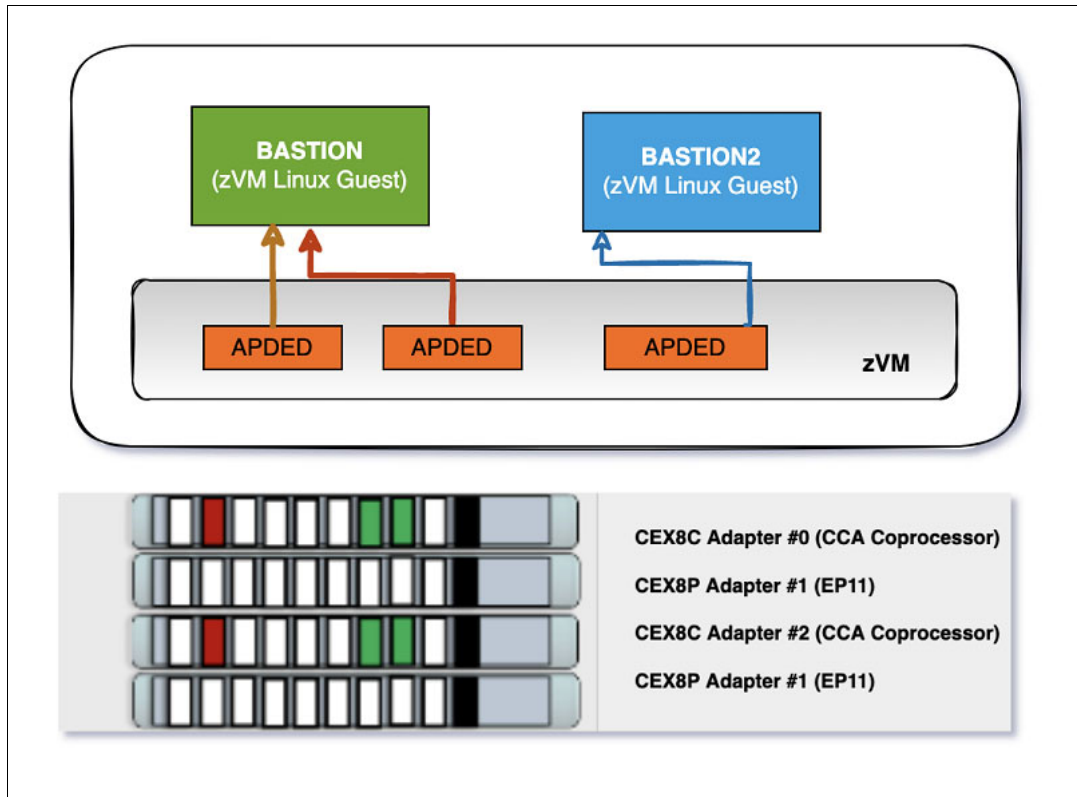


Figure 3-12 Crypto domain assignment to z/VM guests for dedicated use

For a Linux guest that needs access to dedicated queues, the CRYPTO statement in the USER entry for the guest must contain which domain and which AP number is used, which means one or more AP queues are identified and reserved for this guest. There is no virtualization for these dedicated queues, no sharing is done, and the queues are not available for other guests. With dedicated queues, secure key and clear key operations can be performed by the Linux guest. The statement in the user directory looks like: **CRYPTO DOMAIN x APDED y** where **DOMAIN x** would be one or more domains that are defined for the z/VM LPAR and **APDED y** can be one or more APs (CEX8S cards) that are defined for the z/VM LPAR.

The combination of AP and domain numbers should be unique across all cryptography users in the directory. Although you can use directory processing to specify the same AP and DOMAIN combination for multiple users, these users should not be logged on simultaneously. If they are, more than one user might have concurrent access to the same AP queue. Directory processing does not enforce this restriction because duplicate definitions can be useful for backup configurations.

You would use the following two statements to define AP 1 and 3 to the domains 20 and 21:

```
CRYPTO DOMAIN 20 APDED 1
CRYPTO DOMAIN 21 APDED 3
```

You can have multiple CRYPTO statements within one single user statement. To combine them into one command, you would use the following:

```
CRYPTO DOMAIN 20 21 APDED 1 3
```

The directory entry for the guests in this example is shown in Example 3-61.

*Example 3-61 Sample directory entries for dedicated-queues for cryptography access*

---

```
USER BASTION xxxxxxxx 64M 96M ABCDEFG
INCLUDE IBMDFLT
CRYPTO DOMAIN 20 APDED 01
CRYPTO DOMAIN 21 APDED 03
IPL CMS PARM FILEPOOL VMSYS AUTOOCR
OPTION LNKNOPA QUICKDSP
MDISK 0191 3390 71 10 ZVMUSR MR
```

---

Example 3-62 shows the assignment of crypto domains from adapter #1 and #3 to BASTION guest. Issue this command on the z/VM host:

*Example 3-62 Assign crypto domains from adapter #1 and #3 to the BASTION guest*

---

```
DIRM FOR BASTION CRYPTO DOMAIN 20 21 APDED 1 3
DVHXMT1191I Your CRYPTO request has been sent for processing to DIRMAINT
DVHXMT1191I at RDBKZXVMA via DIRMSAT4.
Ready; T=0.01/0.01 14:44:50
DVHREQ2288I Your CRYPTO request for BASTION at *
DVHREQ2288I has been accepted.
DVHBIV3450I The source for directory entry
DVHBIV3450I BASTION has been updated.
DVHBIV3203W Unable to notify
DVHBIV3203W ASYNCHRONOUS_UPDATE_NOTIFICATION_EXIT
DVHBIV3203W recipient of directory update.
DVHBIV3203W Recipient is unreachable.
DVHBIV3203W Unable to notify
DVHBIV3203W ASYNCHRONOUS_UPDATE_NOTIFICATION_EXIT
DVHBIV3203W recipient of directory update.
```

```
DVHBIV3203W Recipient is unreachable.
DVHBIV3203W Unable to notify
DVHBIV3203W ASYNCHRONOUS_UPDATE_NOTIFICATION_EXIT
DVHBIV3203W recipient of directory update.
DVHBIV3203W Recipient is unreachable.
DVHBIV3424I The next ONLINE will take place
DVHBIV3424I immediately.
DVHDRC3451I The next ONLINE will take place
DVHDRC3451I the delta object directory.
DVHRLA3891I Your DSATCTL request has been relayed
DVHRLA3891I for processing.
DVHRLA3891I Your DSATCTL request has been relayed
DVHRLA3891I for processing.
DVHRLA3891I Your DSATCTL request has been relayed
DVHRLA3891I for processing.
DVHRLA3891I Your DMVCTL request has been relayed
DVHRLA3891I for processing.
DVHRLA3891I Your DMVCTL request has been relayed
DVHRLA3891I for processing.
DVHBIV3428I changes made to directory entry BASTION
DVHBIV3428I have been placed online.
DVHREQ2289I your CRYPTO request for BASTION2 at *
DVHREQ2289I has completed; with RC=0
```

---

### **Assignment of Crypto domain 21 from adapters #0 and #2 to BASTION2 guest VM for dedicated use**

Example 3-63 shows a crypto domain assignment for dedicated use from adapters #0 and #02 to BASTION2. Issue this command on the z/VM host:

*Example 3-63 Assign crypto domains from adapter #2 and #4 to BASTION2 guest*

---

#### **DIRM FOR BASTION2 CRYPTO DOMAIN 21 APDED 0 2**

```
DVHXMT1191I Your CRYPTO request has been sent for processing to DIRMAINT
DVHXMT1191I at RDBKZXVMA via DIRMSAT4.
```

```
Ready; T=0.01/0.01 14:50:56
```

```
DVHREQ2288I Your CRYPTO request for BASTION2 at *
DVHREQ2288I has been accepted.
DVHBIV3450I The source for directory entry
DVHBIV3450I BASTION2 has been updated.
DVHBIV3203W Unable to notify
DVHBIV3203W ASYNCHRONOUS_UPDATE_NOTIFICATION_EXIT
DVHBIV3203W recipient of directory update.
DVHBIV3203W Recipient is unreachable.
DVHBIV3203W Unable to notify
DVHBIV3203W ASYNCHRONOUS_UPDATE_NOTIFICATION_EXIT
DVHBIV3203W recipient of directory update.
DVHBIV3203W Recipient is unreachable.
DVHBIV3203W Unable to notify
DVHBIV3203W ASYNCHRONOUS_UPDATE_NOTIFICATION_EXIT
DVHBIV3203W recipient of directory update.
DVHBIV3203W Recipient is unreachable.
DVHBIV3424I The next ONLINE will take place
DVHBIV3424I immediately.
DVHDRC3451I The next ONLINE will take place
DVHDRC3451I the delta object directory.
```

```

DVHRLA3891I Your DSATCTL request has been relayed
DVHRLA3891I for processing.
DVHRLA3891I Your DSATCTL request has been relayed
DVHRLA3891I for processing.
DVHRLA3891I Your DSATCTL request has been relayed
DVHRLA3891I for processing.
DVHRLA3891I Your DMVCTL request has been relayed
DVHRLA3891I for processing.
DVHRLA3891I Your DMVCTL request has been relayed
DVHRLA3891I for processing.
DVHBIV3428I changes made to directory entry BASTION2
DVHBIV3428I have been placed online.
DVHREQ2289I your CRYPTO request for BASTION2 at *
DVHREQ2289I has completed; with RC=0

```

Example 3-64 shows the command to query the crypto domains assigned to z/VM guests. Issue this command on the z/VM host:

*Example 3-64 Query the crypto domains assigned to z/VM guests*

**Query CRYPTO DOMAIN USERS**

```

14:53:00 AP 000 CEX8C   Domain 020 operational online   shared
14:53:00 AP 000 CEX8C   Domain 021 operational online   attached to BASTION2
14:53:00 AP 001 CEX8P   Domain 020 operational online   attached to BASTION
14:53:00 AP 001 CEX8P   Domain 021 operational online   attached to BASTION
14:53:00 AP 002 CEX8C   Domain 020 operational online   shared
14:53:00 AP 002 CEX8C   Domain 021 operational online   attached to BASTION2
14:53:00 AP 003 CEX8P   Domain 020 operational online   attached to BASTION
14:53:00 AP 003 CEX8P   Domain 021 operational online   attached to BASTION
14:53:00
14:53:00 There are no shared-crypto users.
Ready; T=0.01/0.01  14:53:00

```

Example 3-65 shows the crypto domains assigned to z/VM guest, BASTION. SSH to BASTION and execute the command from there:

*Example 3-65 Query crypto domains assigned to z/VM guest BASTION*

```

[root@bastion ~]# hostname
bastion
[root@bastion ~]# lszcrypt
CARD.DOM TYPE  MODE          STATUS        REQUESTS
-----
01      CEX8P EP11-Coproc online         0
01.0014 CEX8P EP11-Coproc online         0
01.0015 CEX8P EP11-Coproc online         0
03      CEX8P EP11-Coproc online         0
03.0014 CEX8P EP11-Coproc online         0
03.0015 CEX8P EP11-Coproc online         0
[root@bastion ~]#

```

Example 3-66 on page 77 shows the crypto domains assigned to z/VM guest, BASTION2. SSH to BASTION2 and execute the query from there.



*Example 3-66 Query crypto domain assigned to z/VM guest BASTION2*

```
[root@bastion2 ~]# hostname
bastion2
[root@bastion2 ~]# 1szcrypt
CARD.DOM TYPE MODE STATUS REQUESTS
-----
00 CEX8C CCA-Coproc online 1
00.0015 CEX8C CCA-Coproc online 1
02 CEX8C CCA-Coproc online 0
02.0015 CEX8C CCA-Coproc online 0
[root@bastion2 ~]#
```

To remove a crypto domain assigned from BASTION, execute the command shown in Example 3-67 from the z/VM host:

*Example 3-67 Remove crypto domain assignment from z/VM guest BASTION*

**DIRM FOR BASTION CRYPTO DELETE**

```
DVHXMT1191I Your CRYPTO request has been sent for processing to DIRMAINT
DVHXMT1191I at RDBKZXVMA via DIRMSAT4.
```

```
Ready; T=0.01/0.01 11:33:17
```

```
DVHREQ2288I Your CRYPTO request for BASTION at *
DVHREQ2288I has been accepted.
DVHBIV3450I The source for directory entry
DVHBIV3450I BASTION has been updated.
DVHBIV3203W Unable to notify
DVHBIV3203W ASYNCHRONOUS_UPDATE_NOTIFICATION_EXIT
DVHBIV3203W recipient of directory update.
DVHBIV3203W Recipient is unreachable.
DVHBIV3203W Unable to notify
DVHBIV3203W ASYNCHRONOUS_UPDATE_NOTIFICATION_EXIT
DVHBIV3203W recipient of directory update.
DVHBIV3203W Recipient is unreachable.
DVHBIV3203W Unable to notify
DVHBIV3203W ASYNCHRONOUS_UPDATE_NOTIFICATION_EXIT
DVHBIV3203W recipient of directory update.
DVHBIV3203W Recipient is unreachable.
DVHBIV3424I The next ONLINE will take place
DVHBIV3424I immediately.
DVHDRC3451I The next ONLINE will take place
DVHDRC3451I the delta object directory.
DVHRLA3891I Your DSATCTL request has been relayed
DVHRLA3891I for processing.
DVHRLA3891I Your DSATCTL request has been relayed
DVHRLA3891I for processing.
DVHRLA3891I Your DSATCTL request has been relayed
DVHRLA3891I for processing.
DVHRLA3891I Your DMVCTL request has been relayed
DVHRLA3891I for processing.
DVHRLA3891I Your DMVCTL request has been relayed
DVHRLA3891I for processing.
DVHBIV3428I changes made to directory entry BASTION
DVHBIV3428I have been placed online.
DVHREQ2289I your CRYPTO request for BASTION at *
```

DVHREQ2289I has completed; with RC=0

---

To remove a crypto domain assigned from BASTION2, execute the command shown in Example 3-68 from the z/VM host.

*Example 3-68 Remove crypto domains assigned to z/VM guest BASTION2*

---

**DIRM FOR BASTION2 CRYPTO DELETE**

DVHXMT1191I Your CRYPTO request has been sent for processing to DIRMAINT  
DVHXMT1191I at RDBKZXVMA via DIRMSAT4.

Ready; T=0.01/0.01 11:33:17

DVHREQ2288I Your CRYPTO request for BASTION2 at \*  
DVHREQ2288I has been accepted.  
DVHBIV3450I The source for directory entry  
DVHBIV3450I BASTION2 has been updated.  
DVHBIV3203W Unable to notify  
DVHBIV3203W ASYNCHRONOUS\_UPDATE\_NOTIFICATION\_EXIT  
DVHBIV3203W recipient of directory update.  
DVHBIV3203W Recipient is unreachable.  
DVHBIV3203W Unable to notify  
DVHBIV3203W ASYNCHRONOUS\_UPDATE\_NOTIFICATION\_EXIT  
DVHBIV3203W recipient of directory update.  
DVHBIV3203W Recipient is unreachable.  
DVHBIV3203W Unable to notify  
DVHBIV3203W ASYNCHRONOUS\_UPDATE\_NOTIFICATION\_EXIT  
DVHBIV3203W recipient of directory update.  
DVHBIV3203W Recipient is unreachable.  
DVHBIV3424I The next ONLINE will take place  
DVHBIV3424I immediately.  
DVHDRC3451I The next ONLINE will take place  
DVHDRC3451I the delta object directory.  
DVHRLA3891I Your DSATCTL request has been relayed  
DVHRLA3891I for processing.  
DVHRLA3891I Your DSATCTL request has been relayed  
DVHRLA3891I for processing.  
DVHRLA3891I Your DSATCTL request has been relayed  
DVHRLA3891I for processing.  
DVHRLA3891I Your DMVCTL request has been relayed  
DVHRLA3891I for processing.  
DVHRLA3891I Your DMVCTL request has been relayed  
DVHRLA3891I for processing.  
DVHBIV3428I changes made to directory entry BASTION2  
DVHBIV3428I have been placed online.  
DVHREQ2289I your CRYPTO request for BASTION2 at \*  
DVHREQ2289I has completed; with RC=0

---

Example 3-69 show Crypto resources assigned to z/VM guests after detachment. Execute the command shown in Example 3-69 from the z/VM host.

*Example 3-69 Query the crypto domain assigned to z/VM guests*

---

**QUERY CRYPTO DOMAIN USERS**

11:37:28	AP 000 CEX8C	Domain 020 operational	online	shared
11:37:28	AP 000 CEX8C	Domain 021 operational	online	free

```
11:37:28 AP 001 CEX8P Domain 020 operational online free
11:37:28 AP 001 CEX8P Domain 021 operational online free
11:37:28 AP 002 CEX8C Domain 020 operational online shared
11:37:28 AP 002 CEX8C Domain 021 operational online free
11:37:28 AP 003 CEX8P Domain 020 operational online free
11:37:28 AP 003 CEX8P Domain 021 operational online free
11:37:28
11:37:28 There are no shared-crypto users.
Ready; T=0.01/0.01 11:37:28
```

---

To validate that the crypto assignments have been removed from the z/VM guests on BASTION, run the command shown in Example 3-70. SSH to BASTION and execute the command from there.

*Example 3-70 Query crypto assignment from z/VM Guest BASTION*

---

```
[root@bastion ~]# hostname
bastion
[root@bastion ~]# lszcrypt
lszcrypt: No crypto card devices found.
[root@bastion ~]#
```

---

To validate that the crypto assignments have been removed from the z/VM guests on BASTION2, run the command shown in Example 3-71. SSH to BASTION2 and execute the command from there.

*Example 3-71 Query crypto assignment from z/VM guest BASTION2*

---

```
[root@bastion2 ~]# hostname
bastion2
[root@bastion2 ~]# lszcrypt
lszcrypt: No crypto card devices found.
[root@bastion2
~]#
```

---

After detaching the crypto resources for z/VM guests from the z/VM host, the crypto resources are no longer seen on the z/VM guests.





## Using a CEX resource within a containerized environment

This chapter discusses the use of a CEX resource within a containerized environment:

- ▶ “CEX resource deployment in a Docker environment” on page 82
- ▶ “CEX deployment configuration in Kubernetes and Red Hat OpenShift Container Platform” on page 89

## 4.1 CEX resource deployment in a Docker environment

A full-fledged virtualization environment like KVM or z/VM is not always needed or necessary. A real virtualization environment provides perfect guest separation but is sometimes too clumsy and requires much more effort to manage.

The Linux kernel offers some base functionality such as name spaces and control groups for lightweight process encapsulation. Docker and the more recent Podman packages exploit these base functions and provide a simple way to use containerized applications.

This section gives an overview of how to set up applications which require CEX resources with the Docker and Podman container runtimes.

### 4.1.1 Installation and simple usage examples for Podman

On a Linux command line a user with **root** privileges can install the Podman package as usual with the **dnf** package manager:

```
dnf install -y podman
```

Podman is a container orchestration tool which is mostly compatible with docker. For convenience and compatibility there exists a package **podman-docker** that tries to fully emulate the docker CLI.

All container administration tools distinguish between an *image* and the *container instance*. An image is simple the base or the template from which containers instances are created. A container (short for container instance) may be altered and may have a state like *running* or *stopped*. Podman knows commands to manipulate images (**podman image ...**) and containers (**podman container ...**) but for historical reasons, the command line does not obey strictly to this distinction.

Here is a condensed walkthrough of important **podman** commands:

► **podman pull <image\_name>**

Pulls an image from a remote repository into the local image repository. Popular repositories are [docker.io](https://docker.io) or [quay.io](https://quay.io).

► **podman images**

List all local images with tag and image id.

► **podman run <image>**

Starts a new container from a given image. Each image has a default *entrypoint* that is executed after the container runtime has been established. This application is executed and with termination of the application the container is also torn down.

The application “inside of the container” is facing only a subset of what the system would offer to a non-containerized application. Other applications are not visible and because of namespacing all system calls show only limited resources. Even user ids and access rights like **root** permissions are only a fake.

The **run** subcommand is the major Podman command and thus knows a whole bunch of options to customize the runtime environment the container will show to the application.

The most often used option with the Podman run command may be to run the container in background with the **--detach** option. It is also possible to give an explicit command with parameters to be executed after instantiation of the image and thus overwrite the default entrypoint. The Podman example session at Figure 4-1 on page 83 shows an invocation

with an endless loop to keep the bash application running in the container busy until killed from “outside”. Outside here means on system level, whereas inside refers to the perception shown by the containerized application.

```

File Edit View Bookmarks Plugins Settings Help
[root@rdbkvm1 ~]# podman pull docker.io/library/bash
Trying to pull docker.io/library/bash:latest...
Getting image source signatures
Copying blob ca38c499cc93 done
Copying blob 47539bffe0f4 done
Copying blob 1f7a5ee186b2 done
Copying config 079c0ff234 done
Writing manifest to image destination
Storing signatures
079c0ff234266d61f35275fa7d826cc654a41f796e7d79ffe24691a52ac4a6c5
[root@rdbkvm1 ~]# podman images
REPOSITORY          TAG          IMAGE ID      CREATED      SIZE
docker.io/library/bash latest      079c0ff23426 3 days ago  14.3 MB
[root@rdbkvm1 ~]# podman run --detach --name mybash bash -c "while true; do sleep 10; echo Hello; done"
3f5f6628141b7f07c5bf9a4fd3114574370d19c2e441178b308c2e337014e321
[root@rdbkvm1 ~]# podman ps
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS        NAMES
3f5f6628141b  docker.io/library/bash:latest -c while true; do...  13 seconds ago Up 13 seconds          mybash
[root@rdbkvm1 ~]# podman stop mybash
WARN[0010] StopSignal SIGTERM failed to stop container mybash in 10 seconds, resorting to SIGKILL
mybash
[root@rdbkvm1 ~]# podman ps --all
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS
3f5f6628141b  docker.io/library/bash:latest -c while true; do...  59 seconds ago Exited (137) 16 seconds ago
[root@rdbkvm1 ~]# podman container rm mybash
mybash
[root@rdbkvm1 ~]# podman rmi 079c0ff23426
Untagged: docker.io/library/bash:latest
Deleted: 079c0ff234266d61f35275fa7d826cc654a41f796e7d79ffe24691a52ac4a6c5
[root@rdbkvm1 ~]# █

```

Figure 4-1 A sample Podman session

- ▶ `podman exec -it <container> <application>`  
 Execute the **<application>** inside the running **<container>**. For example, `podman exec -it mybash /bin/sh` executes a simple shell inside the running mybash container from Figure 4-1 and thus makes it possible to investigate the container runtime (see Figure 4-2 on page 84).
- ▶ `podman cp <[container:]src_path> <[container:]dst_path>`  
 The **cp** subcommand is used to copy files from the local system into the container and vice versa. It is useful during development for example when starting with a bash image and building and deploying the application during development into the running container. Together with the **exec** subcommand a simple way to develop containerized applications.
- ▶ `podman stop <container>`  
 Stops a running container by sending a **SIGTERM** signal to the running application. If the graceful stop does not work, the application gets killed with a **SIGKILL** signal after a time-out of 10 seconds.
- ▶ `podman ps` and `podman ps --all`  
 The **ps** subcommand lists all currently running containers. Together with the **--all** option not anymore running containers (like stopped containers) are also displayed.

```

File Edit View Bookmarks Plugins Settings Help
[root@rdbkvm1 ~]# podman exec -it mybash /bin/sh
/ # pwd
/
/ # ls -la
total 8
dr-xr-xr-x 1 root root 40 Oct 2 12:51 .
dr-xr-xr-x 1 root root 40 Oct 2 12:51 ..
drwxr-xr-x 1 root root 4096 Sep 28 22:00 bin
drwxr-xr-x 5 root root 340 Oct 2 12:51 dev
drwxr-xr-x 1 root root 25 Oct 2 12:51 etc
drwxr-xr-x 2 root root 6 Sep 28 11:18 home
drwxr-xr-x 1 root root 17 Sep 28 22:00 lib
drwxr-xr-x 5 root root 44 Sep 28 11:18 media
drwxr-xr-x 2 root root 6 Sep 28 11:18 mnt
drwxr-xr-x 2 root root 6 Sep 28 11:18 opt
dr-xr-xr-x 524 root root 0 Oct 2 12:51 proc
drwx----- 1 root root 26 Oct 2 12:52 root
drwxr-xr-x 1 root root 42 Oct 2 12:51 run
drwxr-xr-x 2 root root 4096 Sep 28 11:18 sbin
drwxr-xr-x 2 root root 6 Sep 28 11:18 srv
dr-xr-xr-x 12 root root 0 Sep 21 14:35 sys
drwxrwxrwt 1 root root 6 Sep 28 22:00 tmp
drwxr-xr-x 1 root root 19 Sep 28 22:00 usr
drwxr-xr-x 1 root root 17 Sep 28 11:18 var
/ # █

```

Figure 4-2 Executing a shell to glimpse into the container

- ▶ `podman container rm <container>`  
Deletes a stopped container instance eventually freeing the resources (disc space) associated with this instance.
- ▶ `podman rmi <image>`  
Deletes an image from the local image repository freeing the resources (disk space) associated with this image.

These commands are only a subset, scratching the surface about what is possible with Docker or Podman container orchestration. Images may be altered and persistently stored, and there can be local networks defined to have containers communicate with each other and much more. For more information, see [Getting Started with Podman](#).

### 4.1.2 Simple deployment of CEX resources

The screenshot in Figure 4-3 on page 85 shows the pull and instantiation of an *Red Hat Universal Base Image* (UBI), the package installation of *s390 tools* the `1szcrypt` listing within the container instance, and more commands.

A first glimpse to this output leads to the impression that all the crypto resources of the system are as well available within the container. If someone would run a crypto load which tries to access a Crypto Express card the application would fail however. The reason is also shown in the terminal output: The device node `/dev/z90crypt` that is the communication channel between the application and the crypto resources is not available.



Note also that **lszcrypt** seems to work fine. The API for most of the user space administrative tools is the *sysfs* pseudo file system provided by the Linux kernel. For example **lszcrypt** gathers its information there and **chzcrypt** manipulates the crypto resources via writing into some files within the *sysfs* hierarchy. Also by default the user id of the user running in the container is shown as the *root* user, this is a fake: The *sysfs* within the container is read-only and refuses any write attempts and the *root* user inside the container is mapped to an ordinary user with limited capacities outside the container.

```
File Edit View Bookmarks Plugins Settings Help
[root@rdbkvm1 ~]# podman pull docker.io/redhat/ubi9
Trying to pull docker.io/redhat/ubi9:latest...
Getting image source signatures
Copying blob e414739e5d8e done
Copying config 3dabd17513 done
Writing manifest to image destination
Storing signatures
3dabd17513f8c4f300151fb6c9af9a095485c9c0b41b78c13ee4b3057af0ac69
[root@rdbkvm1 ~]# podman images
REPOSITORY          TAG          IMAGE ID        CREATED        SIZE
docker.io/library/bash latest       079c0ff23426   3 days ago    14.3 MB
docker.io/redhat/ubi9 latest       3dabd17513f8   3 weeks ago   214 MB
[root@rdbkvm1 ~]# podman run --detach --name myubi9 ubi9 /bin/sh -c "while true; do sleep 10; done"
952b56f23c74344c2e3356ca65c84c33f79921c3c3bea563827be8765fdd6b06
[root@rdbkvm1 ~]# podman exec -it myubi9 /bin/sh
sh-5.1# dnf install -y s390utils-core s390utils-base >/dev/null
sh-5.1# lszcrypt
CARD.DOM TYPE MODE STATUS REQUESTS
-----
00 CEX8C CCA-Coproc online 1342
00.000a CEX8C CCA-Coproc online 1342
01 CEX8A Accelerator online 0
01.000a CEX8A Accelerator online 0
02 CEX8C CCA-Coproc online 756
02.000a CEX8C CCA-Coproc online 756
03 CEX8A Accelerator online 0
03.000a CEX8A Accelerator online 0
04 CEX8C CCA-Coproc online 756
04.000a CEX8C CCA-Coproc online 756
05 CEX8P EP11-Coproc online 79
05.000a CEX8P EP11-Coproc online 79
06 CEX8C CCA-Coproc online 756
06.000a CEX8C CCA-Coproc online 756
07 CEX8P EP11-Coproc online 67
07.000a CEX8P EP11-Coproc online 67
sh-5.1# id
uid=0(root) gid=0(root) groups=0(root)
sh-5.1# ls -la /dev/z90*
ls: cannot access '/dev/z90*': No such file or directory
sh-5.1# █
```

Figure 4-3 UBI container with crypto resources

For the missing **/dev/z90crypt**, there is a way to forward this into the container. Make it available at container start with the **--device /dev/z90crypt** option:

```
podman run <image> --device /dev/z90crypt [...]
```

and consider to forward **/dev/hwrng** (hardware random generator), **/dev/trng** (true random generator) and **/dev/prandom** (pseudo random generator) as well. For example:

```
podman run <image> --device /dev/z90crypt --device /dev/hwrng --device /dev/trng --device /dev/prandom [...]
```

With the mentioned shadowing of the **z90crypt** device into the container environment, all crypto load gains access to the whole bunch of crypto resources available on the system. There is no restriction in the crypto load or any limitation with the accessibility like limiting to only accelerator cards possible. However, all attempts to manipulate the AP bus which is the maintainer and book keeper of the crypto resources will fail due to the read-only *sysfs* replica. For example, some commands try to temporarily change the online state of crypto cards or queues which will fail in containerized environment.

A simple forward of the zcrypt device for containerized applications does not attain additional security. There is no way to restrict or filter either the crypto load or the crypto resources. However, running an application within container environment may be suitable for other nonspecific CEX reasons.

### 4.1.3 A more sophisticated CEX deployment

Since Linux Kernel 4.20 and s390-tools 2.7 there is a feature integrated which allows to create and customize additional zcrypt device driver nodes. The CEX Device Plug-in for Kubernetes exploits this extension, but it can be used with Docker or Podman containerization as well.

The idea of the additional zcrypt device driver node is to create an alternate device node from the generic `/dev/z90crypt` device and customize which resources can be accessed through this. This new device node is then used as a replacement for the `z90crypt` device inside the container. The following paragraphs lead through this process step by step.

#### Checking for availability and creation of an alternate zcrypt device node

For additional zcrypt device node support check that either directory `/sys/class/zcrypt` exists or simply by running `zcryptctl list`. However all recent Linux distributions have this feature available and enabled since several years now.

Create your own `z90crypt` device node with the command:

```
zcryptctl create <my_zcrypt_node>
```

There should be a new device node `<my_zcrypt_node>` in the devices directory `/dev`. The screenshot in Figure 4-4 on page 87 shows an example with listing the available alternate zcrypt devices with `zcryptctl list` immediately after creation. The created device needs customization.

```

File Edit View Bookmarks Plugins Settings Help
[root@rdbkkvm1 ~]# zcryptctl create my_zcrypt_node
Device node created
[root@rdbkkvm1 ~]# zcryptctl list
zcrypt node name:      my_zcrypt_node
device node:  /dev/my_zcrypt_node
major:minor:  245:0
adapter:
domains:
control domains:
ioctls:
[root@rdbkkvm1 ~]# zcryptctl addap my_zcrypt_node 7
Adapter 7 added
[root@rdbkkvm1 ~]# zcryptctl adddom my_zcrypt_node 10
Domain 10 added
[root@rdbkkvm1 ~]# zcryptctl addioctl my_zcrypt_node ZSENDER11CPRB
Ioctl ZSENDER11CPRB added
[root@rdbkkvm1 ~]# zcryptctl list
zcrypt node name:      my_zcrypt_node
device node:  /dev/my_zcrypt_node
major:minor:  245:0
adapter:      7
domains:     10
control domains:
ioctls:      ZSENDER11CPRB
[root@rdbkkvm1 ~]# ls -la /dev/my*
crw-----. 1 root root 245, 0 Oct  3 06:27 /dev/my_zcrypt_node
[root@rdbkkvm1 ~]# chmod 666 /dev/my_zcrypt_node
[root@rdbkkvm1 ~]# ls -la /dev/my*
crw-rw-rw-. 1 root root 245, 0 Oct  3 06:27 /dev/my_zcrypt_node
[root@rdbkkvm1 ~]# █

```

Figure 4-4 A sample zcryptctl session

## Customizing the alternate device node

The newly created alternate device node needs to get customized before usage. By default this device does not allow any ioctl command addressing any adapter or domain. So, at least one crypto card should be enabled:

```
zcryptctl addap <my_zcrypt_node> <adapter>
```

and one domain should be enabled:

```
zcryptctl adddom <my_zcrypt_node> <domain>
```

Additionally, the **ioctl** commands permitted on this device need to be adjusted. Table 4-1 shows the available **ioctl** commands, their meaning, and a mnemonic that can be used with **zcryptctl**. The **ioctl** commands can and sometimes need to be combined. For example, enable **ICARSAMODEXPO** and **ICARSACRT** for an Accelerator APQN:

```
zcryptctl addioctl <nodename> ICARSAMODEXPO
zcryptctl addioct  <nodename> ICARSACRT
```

A CCA APQN usually just needs to have the **ZSENDER11CPRB** ioctl command enabled, whereas an APQN in EP11 mode should have only **ZSENDER11CPRB** active.

Table 4-1 zcryptctl ioctl command numbers, mnemonics and meaning

ioctl command number	mnemonic	remark
0-255	ALL	Enables or disables the whole range of possible ioctl commands

ioctl command number	mnemonic	remark
5	ICARSAMODEXPO	Enables or disables sending of clear key RSA ME requests
6	ICARSACRT	Enables or disables sending of clear key RSA CRT requests
129	ZSESEND CPRB	Enables or disables sending of secure key CCA requests
4	ZSENDEP11 CPRB	Enables or disables sending of secure key EP11 requests
95	ZCRYPT_DEVICE_STATUS	Legacy ioctl command - only necessary for ancient CCA library versions
88	ZCRYPT_STATUS_MASK	Legacy ioctl command - only necessary for ancient CCA library or libica library versions
89	ZCRYPT_QDEPTH_MASK	Legacy ioctl - not used any more
90	ZCRYPT_PERDEV_REQCNT	Legacy ioctl - not used any more
<any number in range 0...255>	-	Enables or disables exactly this ioctl command

It is not a high risk to simply enable all ioctls with the ALL mnemonic as the zcrypt device driver will anyway refuse to call ioctl commands that do not fit to the card mode. For example, the invocation of ioctl **ZSENDEP11 CPRB** will be abandoned when an Accelerator or CCA APQN is addressed.

As an additional step the file permissions of the created device node `/dev/<my_zcrypt_node>` should get some attention. The permissions need to fit to the executing user capabilities inside the container. As already stated in the previous section, the executing root user within the container is only a fake. The mapping here depends on the container runtime and can get adjusted with the container start, so best practice is to change the node permissions to **0666** to enable read-write access for everybody. With some more knowledge about the container runtime used this can be fine tuned with individual file permissions if desired.

The terminal output in Figure 4-4 on page 87 shows a session snippet from the creation of an alternate zcrypt device node up to all customization steps.

## Running a container with an alternate zcrypt device node

When the customization of the alternate zcrypt device node is complete, the container can be started:

```
podman run <image> --device /dev/<my_zcrypt_node>:/dev/z90crypt [...]
```

The command-line option `--device <source_device>:<target_device>` tells Podman to shadow the system device `<source_device>` into the container as `<target_device>`. So in the end the `/dev/z90crypt` inside the container is in fact the alternate zcrypt device created in the previous steps.

The sysfs pseudo file system appearing inside the container lacks still some adaptations. The issues already mentioned in 4.1.2, “Simple deployment of CEX resources” on page 84 still apply. The sysfs shows all cards and domains visible on the system and `1szcrypt` lists them without any respect to the limited accessibility inside the container.

Some applications or libraries may also need further guidance to avoid confusion between what is visible in sysfs and what is accessible via `/dev/z90crypt`.

► CCA library

The CCA library uses the first CCA adapter of the highest generation and always addresses the chosen default domain (displayed with `1szcrypt -b`). This can be changed via environment variables:

```
export CSU_DEFAULT_ADAPTER=CRPxx
export CSU_DEFAULT_DOMAIN=dd
```

The CCA library counts only CCA adapters, starting with 1, so CRP01 means to use the first CCA adapter. For more details see the [Common Cryptographic Architecture Application Programmer's Guide](#).

► EP11 library

The programming API of the EP11 library requires to provide a target parameter. So, the library itself knows no default adapter or domain but requires the application to provide target addressing parameters.

► OpenCryptoki

The CCA token is based on the CCA library and thus obeys to the statement above. For the EP11 token one can provide a configuration file `ep11tok.conf` that lists adapter and domain pairs to use. For more details, see [openCryptoki overview](#).

## Cleaning up after container termination

When an alternate zcrypt device node is not used any more, it should be destroyed. The Linux kernel needs to hold some structures and thus memory for each alternate zcrypt device node and thus limits the maximum number of these devices to 256:

```
zcryptctl destroy <my_zcrypt_node>
```

The alternate device node is marked for deletion and destroyed with the last application releasing the use of the device node.

## 4.2 CEX deployment configuration in Kubernetes and Red Hat OpenShift Container Platform

In Kubernetes (K8s) and Red Hat OpenShift Container Platform environments, the functionalities of IBM Crypto Express cards are not available by default. Essentially, this means that without configuring specific settings and intermediate solutions, containerized applications (PODs) cannot access the resources of CEX adapters on IBM Z & LinuxONE systems.

In this section, we present an extension for Kubernetes and Red Hat OpenShift Container Platform that allows any container running in a POD to access a Crypto Express adapter domain. In this context, a “domain” represents a “virtualized crypto resource”. The purpose of this section is to demonstrate how to set up the basic infrastructure and configure which cryptographic resource belongs to which workload, and how the application within the container can access the cryptographic resource.

To achieve all this, a new component is required on the compute nodes, called the IBM Crypto Express Card Kubernetes device plug-in.

This component is available as a certified and supported container image in the Red Hat image catalog, and there is also a community version available.

In our lab environment, we worked with the community version of the CEX device plug-in, which can be found at [Kubernetes device plug-in for IBM CryptoExpress \(CEX\) cards](#), but the described steps are applicable to the certified and supported image from the Red Hat registry as well.

The specifics discussed in this section are demonstrated in our lab environment on a Kubernetes-based Red Hat OpenShift cluster.

### 4.2.1 Kubernetes on a Red Hat OpenShift cluster

Chapter 1, “Introduction” on page 1 provided a brief introduction into the cryptographic components, terms and concepts, including CEX8S features. Now we will apply those terms and concepts in this section as we discuss Kubernetes (K8s) on a Red Hat OpenShift cluster and how to configure and set up CEX8S resources in Kubernetes orchestrated containers. A Linux on IBM Z and IBM LinuxONE crypto stack is shown in Figure 4-5.

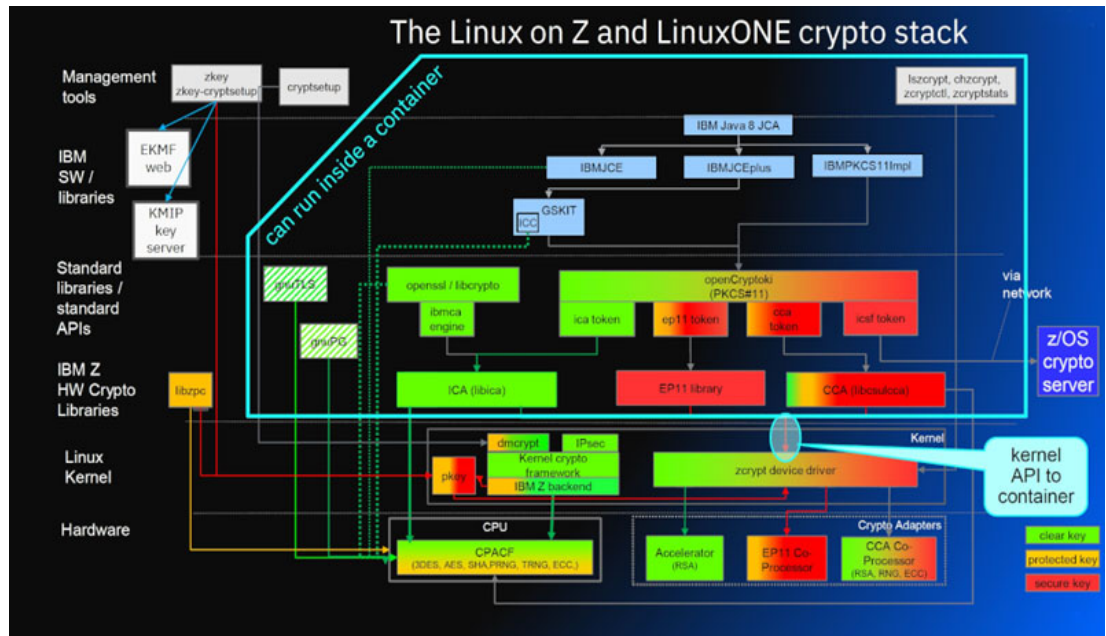


Figure 4-5 The Crypto stack

A Kubernetes Red Hat OpenShift cluster is a framework for deploying workloads as containers and accomplishes the following:

- ▶ High availability
- ▶ Flexible deployment
- ▶ Based on K8s technology

Workloads are organized onto what are called nodes, where the nodes can be (based on its role):

- ▶ Control planes or master nodes, serving as management nodes
- ▶ Compute nodes or worker nodes, responsible for running actual workloads

Pods are running on these nodes, where a pod is defined as a set of containers implementing the workload.

## IBM Z and LinuxONE crypto hardware support in cluster environments

The use of CPACF (by kernel crypto, OpenSSL, IBM Java 8 JCE, Go) is transparent in:

- ▶ Control plane
- ▶ Compute node components
- ▶ Containers running in compute nodes

## Requirements to use CEX adapters for Kubernetes clusters

The following are requirements to enable the use of CEX adapters for Kubernetes clusters.

- ▶ Crypto Express cards must be connected to the IBM Z or LinuxONE server hosting the compute nodes.
  - The mode of operation (Accelerator, CCA, EP11) for each crypto module (adapter) must be configured on the Support Element.
- ▶ An LPAR running either KVM or z/VM hypervisor
  - Assign an adapter set and a domain set on the HMC to this LPAR. This defines all APQNs available to the guest servers on this LPAR.
- ▶ Each guest machine running a compute node (KVM or z/VM)
  - Must be assigned an adapter and domain list. For z/VM guests the assignment exists in the user directory. For KVM guests, we create an mdev device with adapters and domains assigned. This defines the APQNs that are “dedicated” to each compute node.

## 4.2.2 CEX resources in Kubernetes orchestrated containers

In this section, we discuss setting up, configuring and using CEX resources in Kubernetes orchestrated containers.

### Virtualization layer, guest level

First, we should check what resources are available at the LPAR level. The rest of the steps will be presented in a KVM virtualization environment, the procedure is similar for z/VM guest machines, the difference is how to attach the APQN to the guest machine, which is shown in 3.2, “Configuring z/VM guests to use CEX adapters” on page 53.

The KVM configuration is done as discussed in 3.1, “Configuring a KVM host to provide CEX functionality” on page 32. Only the relevant changes will be pointed out in this section. Example 4-1 shows the command to check the CEX adapters with its results.

#### Example 4-1 Checking the CEX adapters

```
[root@rdbkvm4 ~]# lszcrypt -V
```

CARD.DOM	TYPE	MODE	STATUS	REQUESTS	PENDING	HWTYPE	QDEPTH	FUNCTIONS	DRIVER
00	CEX8C	CCA-Coproc	online	3939	0	14	08	S--D--NF-	cex4card
00.000d	CEX8C	CCA-Coproc	in use	-	-	14	08	S--D--NF-	vfio_ap
00.0011	CEX8C	CCA-Coproc	in use	-	-	14	08	S--D--NF-	vfio_ap
01	CEX8A	Accelerator	online	0	0	14	08	-MC-A-NF-	cex4card
01.000d	CEX8A	Accelerator	in use	-	-	14	08	-MC-A-NF-	vfio_ap
01.0011	CEX8A	Accelerator	in use	-	-	14	08	-MC-A-NF-	vfio_ap
02	CEX8C	CCA-Coproc	online	2208	0	14	08	S--D--NF-	cex4card

02.000d	CEX8C	CCA-Coproc	in use	-	-	14	08	S--D--NF-	vfiopap
02.0011	CEX8C	CCA-Coproc	in use	-	-	14	08	S--D--NF-	vfiopap
03	CEX8A	Accelerator	online	0	0	14	08	-MC-A-NF-	cex4card
03.000d	CEX8A	Accelerator	in use	-	-	14	08	-MC-A-NF-	vfiopap
03.0011	CEX8A	Accelerator	in use	-	-	14	08	-MC-A-NF-	vfiopap
04	CEX8C	CCA-Coproc	online	2208	0	14	08	S--D--NF-	cex4card
04.000d	CEX8C	CCA-Coproc	unassigned	-	-	14	08	S--D--NF-	vfiopap
04.0011	CEX8C	CCA-Coproc	unassigned	-	-	14	08	S--D--NF-	vfiopap
05	CEX8P	EP11-Coproc	online	193	0	14	08	-----XNF-	cex4card
05.000d	CEX8P	EP11-Coproc	in use	-	-	14	08	-----XNF-	vfiopap
05.0011	CEX8P	EP11-Coproc	in use	-	-	14	08	-----XNF-	vfiopap
06	CEX8C	CCA-Coproc	online	2208	0	14	08	S--D--NF-	cex4card
06.000d	CEX8C	CCA-Coproc	unassigned	-	-	14	08	S--D--NF-	vfiopap
06.0011	CEX8C	CCA-Coproc	unassigned	-	-	14	08	S--D--NF-	vfiopap
07	CEX8P	EP11-Coproc	online	243	0	14	08	-----XNF-	cex4card
07.000d	CEX8P	EP11-Coproc	in use	-	-	14	08	-----XNF-	vfiopap
07.0011	CEX8P	EP11-Coproc	in use	-	-	14	08	-----XNF-	vfiopap

In our lab environment, there are several worker nodes in our Red Hat OpenShift cluster, but for simplicity we will use only worker-1 for setup and testing.

We create a new mdev device and allocate the resources shown in example 4-2 to the aw1 node.

#### Example 4-2 mdev definition and starting

```
[root@rdbkvm4 ~]# vi 00-02-01-03-05-07-0011.xml

<device>
  <parent>ap_matrix</parent>
  <capability type="mdev">
    <type id="vfiopap-passthrough"/>
    <attr name="assign_adapter" value="0x00"/>
    <attr name="assign_adapter" value="0x02"/>
    <attr name="assign_adapter" value="0x01"/>
    <attr name="assign_adapter" value="0x03"/>
    <attr name="assign_adapter" value="0x05"/>
    <attr name="assign_adapter" value="0x07"/>
    <attr name="assign_domain" value="0x00d"/>
    <attr name="assign_domain" value="0x0011"/>
    <uuid>00020103-0507-000d-0011-28a403f8b357</uuid>
  </capability>
</device>

[root@rdbkvm4 ~]# virsh nodedev-define 00-02-01-03-05-07-0011.xml
Node device 'mdev_00020103_0507_000d_0011_28a403f8b357_matrix' defined from
'00-02-01-03-05-07-0011.xml'
[root@rdbkvm4 ~]# virsh nodedev-start mdev_00020103_0507_000d_0011_28a403f8b357_matrix
Device mdev_00020103_0507_000d_0011_28a403f8b357_matrix started
```

Attach it to a aw1 worker node by using the following command:

```
[root@rdbkvm4 ~]# virsh attach-device aw1 med-dev-all.xml
Device attached successfully
```

Example 4-3 shows the command to check the resources at the KVM level. Note the status in this example shows all assigned resources are in use.

#### Example 4-3 Checking online resources

```
[root@rdbkvm4 ~]# lszycrypt -V
```



CARD.DOM	TYPE	MODE	STATUS	REQUESTS	PENDING	HWTYPE	QDEPTH	FUNCTIONS	DRIVER
-									
00	CEX8C	CCA-Coproc	online	3939	0	14	08	S--D--NF-	cex4card
00.000d	CEX8C	CCA-Coproc	in use	-	-	14	08	S--D--NF-	vfio_ap
00.0011	CEX8C	CCA-Coproc	in use	-	-	14	08	S--D--NF-	vfio_ap
01	CEX8A	Accelerator	online	0	0	14	08	-MC-A-NF-	cex4card
01.000d	CEX8A	Accelerator	in use	-	-	14	08	-MC-A-NF-	vfio_ap
01.0011	CEX8A	Accelerator	in use	-	-	14	08	-MC-A-NF-	vfio_ap
02	CEX8C	CCA-Coproc	online	2208	0	14	08	S--D--NF-	cex4card
02.000d	CEX8C	CCA-Coproc	in use	-	-	14	08	S--D--NF-	vfio_ap
02.0011	CEX8C	CCA-Coproc	in use	-	-	14	08	S--D--NF-	vfio_ap
03	CEX8A	Accelerator	online	0	0	14	08	-MC-A-NF-	cex4card
03.000d	CEX8A	Accelerator	in use	-	-	14	08	-MC-A-NF-	vfio_ap
03.0011	CEX8A	Accelerator	in use	-	-	14	08	-MC-A-NF-	vfio_ap
04	CEX8C	CCA-Coproc	online	2208	0	14	08	S--D--NF-	cex4card
04.000d	CEX8C	CCA-Coproc	unassigned	-	-	14	08	S--D--NF-	vfio_ap
04.0011	CEX8C	CCA-Coproc	unassigned	-	-	14	08	S--D--NF-	vfio_ap
05	CEX8P	EP11-Coproc	online	193	0	14	08	-----XNF-	cex4card
05.000d	CEX8P	EP11-Coproc	in use	-	-	14	08	-----XNF-	vfio_ap
05.0011	CEX8P	EP11-Coproc	in use	-	-	14	08	-----XNF-	vfio_ap
06	CEX8C	CCA-Coproc	online	2208	0	14	08	S--D--NF-	cex4card
06.000d	CEX8C	CCA-Coproc	unassigned	-	-	14	08	S--D--NF-	vfio_ap
06.0011	CEX8C	CCA-Coproc	unassigned	-	-	14	08	S--D--NF-	vfio_ap
07	CEX8P	EP11-Coproc	online	243	0	14	08	-----XNF-	cex4card
07.000d	CEX8P	EP11-Coproc	in use	-	-	14	08	-----XNF-	vfio_ap
07.0011	CEX8P	EP11-Coproc	in use	-	-	14	08	-----XNF-	vfio_ap

We can use a secure shell (SSH) to connect to the `aw1` node and check the resources that are visible at the OS level, as shown in Example 4-4. This example shows that there are six adapters online, with `00.000d` and `00.0011` domains.

*Example 4-4 Check the resources that are visible at the OS level*

```
[root@bastion ~]# ssh core@aw1
[core@aw1 ~]$ cd /sys/devices/ap/
[core@aw1 ap]$ ls -l
total 0
drwxr-xr-x. 4 root root    0 Nov 15 14:34 card00
drwxr-xr-x. 4 root root    0 Nov 15 14:34 card01
drwxr-xr-x. 4 root root    0 Nov 15 14:34 card02
drwxr-xr-x. 4 root root    0 Nov 15 14:34 card03
drwxr-xr-x. 4 root root    0 Nov 15 14:34 card05
drwxr-xr-x. 4 root root    0 Nov 15 14:34 card07
-rw-r--r--. 1 root root 4096 Oct 25 10:38 uevent

[core@aw1 ap]$ for i in {0,1,2,3,5,7}; do ls -l ./card0$i/|grep '0.\.'; done
drwxr-xr-x. 2 root root    0 Nov 15 14:34 00.000d
drwxr-xr-x. 2 root root    0 Nov 15 14:34 00.0011
drwxr-xr-x. 2 root root    0 Nov 15 14:34 01.000d
drwxr-xr-x. 2 root root    0 Nov 15 14:34 01.0011
drwxr-xr-x. 2 root root    0 Nov 15 14:34 02.000d
drwxr-xr-x. 2 root root    0 Nov 15 14:34 02.0011
drwxr-xr-x. 2 root root    0 Nov 15 14:34 03.000d
drwxr-xr-x. 2 root root    0 Nov 15 14:34 03.0011
drwxr-xr-x. 2 root root    0 Nov 15 14:34 05.000d
drwxr-xr-x. 2 root root    0 Nov 15 14:34 05.0011
drwxr-xr-x. 2 root root    0 Nov 15 14:34 07.000d
drwxr-xr-x. 2 root root    0 Nov 15 14:34 07.0011
```

Example 4-5 shows these resources sorted by type. Six adapters are online, each with 00.000d and 00.0011 domains.

*Example 4-5 Resources sorted by “type”*

CARD	DOMAIN	TYPE	MODE	STATUS	REQUESTS	PENDING	HWTYP	QDEPTH	FUNCTIONS	DRIVER
01		CEX8A	Accelerator	online	0	0	14	08	-MC-A-NF-	
cex4card										
01	000d	CEX8A	Accelerator	in use	-	-	14	08	-MC-A-NF-	vfiop_ap
01	0011	CEX8A	Accelerator	in use	-	-	14	08	-MC-A-NF-	vfiop_ap
03		CEX8A	Accelerator	online	0	0	14	08	-MC-A-NF-	
cex4card										
03	000d	CEX8A	Accelerator	in use	-	-	14	08	-MC-A-NF-	vfiop_ap
03	0011	CEX8A	Accelerator	in use	-	-	14	08	-MC-A-NF-	vfiop_ap
00		CEX8C	CCA-Coproc	online	3939	0	14	08	S--D--NF-	
cex4card										
00	000d	CEX8C	CCA-Coproc	in use	-	-	14	08	S--D--NF-	vfiop_ap
00	0011	CEX8C	CCA-Coproc	in use	-	-	14	08	S--D--NF-	vfiop_ap
02		CEX8C	CCA-Coproc	online	2208	0	14	08	S--D--NF-	
cex4card										
02	000d	CEX8C	CCA-Coproc	in use	-	-	14	08	S--D--NF-	vfiop_ap
02	0011	CEX8C	CCA-Coproc	in use	-	-	14	08	S--D--NF-	vfiop_ap
05		CEX8P	EP11-Coproc	online	193	0	14	08	-----XNF-	
cex4card										
05	000d	CEX8P	EP11-Coproc	in use	-	-	14	08	-----XNF-	vfiop_ap
05	0011	CEX8P	EP11-Coproc	in use	-	-	14	08	-----XNF-	vfiop_ap
07		CEX8P	EP11-Coproc	online	243	0	14	08	-----XNF-	
cex4card										
07	000d	CEX8P	EP11-Coproc	in use	-	-	14	08	-----XNF-	vfiop_ap
07	0011	CEX8P	EP11-Coproc	in use	-	-	14	08	-----XNF-	vfiop_ap

As shown in these examples, we see three distinct types of resources visible at the operating system level on the aw1 node. By definition, we have the 000d and 0011 domains (equivalent to 13 and 17 in decimal) available on two cards each.

**Note:** To clarify:

Cards 01 and 03:

- ▶ Domains: 000d(13) and 0011(17)
- ▶ Distinct types of resource: CEX8A (accelerator)

Cards 00 and 02:

- ▶ Domains: 000d(13) and 0011(17)
- ▶ Distinct types of resource: CEX8C (CCA coprocessor)

Cards 05 and 07:

- ▶ Domains: 000d(13) and 0011(17)
- ▶ Distinct types of resource: CEX8P (EP11 coprocessor)

## CEX device plug-in

Now that we are satisfied that the desired resources are available, we can move on to configuring the CEX plug-in. In this section, we present the steps required to configure the CEX device plug-in, accompanied by additional summaries.

The Kubernetes CEX device plug-in environment enables IBM Crypto Express cards to be accessible on Kubernetes nodes for use by containers. The CEX device plug-in categorizes available CEX resources into CEX configuration sets.

A CEX resource is a single domain within a CEX adapter. CEX resources are identified by their host and their APQN (the pair of an adapter ID and a domain ID).

- ▶ They can have additional attributes like adapter mode (accelerator, CCA, EP11) or CEX generation (CEX5S, CEX6S, CEX7S, CEX8S) or machine ID.

From a container perspective, APQNs within the CEX configuration set must be equivalent, meaning any APQN can be used for any cryptographic task. Equivalent CEX resources can be configured to belong to a crypto configuration set.

In Kubernetes, CEX configuration sets need to be defined in a clusterwide ConfigMap. This definition could be managed by a cluster administrator.

In our example:

- ▶ Each container can be assigned a single resource from a specific crypto config set.

**Note:**

- ▶ A pod comprising multiple containers can use multiple CEX resources — one for each of its containers.
- ▶ A crypto resource can be assigned to  $n \geq 1$  containers if the APQN overcommit limit  $n$  is configured in the deployment.

Example 4-6 provides our sample ConfigMap.

*Example 4-6 Sample ConfigMap of resources*

```
[root@bastion configmap]# cat cex_resources.json
{
  "cryptoconfigsets":
  [
    {
      "setname": "Accelerator_for_Lancelot",
      "project": "knights",
      "cexmode": "accel",
      "overcommit": 3,
      "apqns":
      [
        {
          "adapter": 1,
          "domain": 13,
          "machineid": ""
        },
        {
          "adapter": 3,
          "domain": 13,
          "machineid": ""
        },
        {
          "adapter": 1,
          "domain": 11,
          "machineid": ""
        }
      ]
    }
  ]
}
```

```

        {
            "adapter": 3,
            "domain": 11,
            "machineid": ""
        }
    ]
},
{
    "setname": "EP11_for_Galahad",
    "project": "knights",
    "cexmode": "ep11",
    "apqns":
    [
        {
            "adapter": 5,
            "domain": 13,
            "machineid": ""
        },
        {
            "adapter": 7,
            "domain": 13,
            "machineid": ""
        },
        {
            "adapter": 5,
            "domain": 11,
            "machineid": ""
        },
        {
            "adapter": 7,
            "domain": 11,
            "machineid": ""
        }
    ]
},
{
    "setname": "CCA-Coproc_for_Bedivere",
    "project": "knights",
    "cexmode": "cca",
    "overcommit": 5,
    "apqns":
    [
        {
            "adapter": 0,
            "domain": 13,
            "machineid": ""
        },
        {
            "adapter": 2,
            "domain": 13,
            "machineid": ""
        },
        {
            "adapter": 0,
            "domain": 11,

```

```

    "machineid": ""
  },
  {
    "adapter": 2,
    "domain": 11,
    "machineid": ""
  }
]
}

```

The ConfigMap defines a list of configuration sets. Each of the three types of APQNs requires a separate set to be created in order to be used. Each configuration set is created with the entries shown in Table 4-2.

Table 4-2 Basic parameters

Variable	Required/Optional	Explanation
setname	required	Can be any string value, but must be unique within all the configuration sets. This is the identifier used by the container to request one of the CEX crypto resources from within the set.
project	required	Can be any string value or namespace of the configuration set. Only containers with matching namespaces can access CEX crypto resources of the configuration set. For version 1, this is not fully implemented as there are limits on the existing API preventing this.
cexmode	optional	Specifies the CEX mode. If specified, one of the following choices is required: EP11, CCA, or ACCEL. Adds an extra verification step every time the APQNs on each node are screened by the CEX device plug-in. All APQNs of the configuration set must match the specified CEX mode.
mincexgen	optional	Specifies the minimum CEX card generation for the configuration set, for example, mincexgen: "cex6"

Variable	Required/Optional	Explanation
overcommit	optional	<p>Specifies the overcommit limit for resources in this ConfigSet.</p> <p>If the parameter is omitted, it defaults to the value specified through the environment variable APQN_OVERCOMMIT_LIMIT.</p> <p>If the environment variable is not specified, the default value for overcommit is 1 (no overcommit).</p>

### APQN parameters

A list of equivalent APQN entries is shown in Table 4-3. The exact meaning of equivalent depends on the crypto workload to be run with the crypto configuration set. However, it forms a set of APQNs where anyone is sufficient to fulfill the needs of the requesting crypto workload container.

#### Note:

An APQN must not be member of more than one crypto configuration set.

It is valid to provide an empty list. It is also valid to provide APQNs, which might currently not exist but might come into existence sometime in future when new crypto cards are plugged.

Table 4-3 APQN parameters

Variable	Required/Optional	Explanation
adapter	required	The CEX card number. Can be in the range of 0-255. Typically referred to as adapter number
domain	required	The domain on the adapter. Can be in the range of 0-255. decimal value of the domain.
machineid	optional	This is only required when the compute nodes are physically located on different hardware instances and the APQN pairs (adapter, domain) are not unique. It should be created based on the <code>/proc/sysinfo</code> <code>&lt;manufacturer&gt;-&lt;machinetype&gt;-&lt;sequencecode&gt;</code>

A cluster-wide CEX ConfigMap can be installed using a properly filled yaml file. The command to do this is shown in Example 4-7.

#### Example 4-7 Deploy ConfigMap

```
oc create -f <my_cex_resources.yaml>
```

Until now, we have defined resource assignments. However, this is not the end of the tasks because we still need to install the CEX-plugin as well.

### **Obtaining the CEX device plug-in**

We used the community version of the plugin. The source files of the CEX device plug-in can be found at [Kubernetes device plug-in for IBM CryptoExpress \(CEX\) cards](#).

### **Installing the CEX device plug-in**

The CEX device plug-in must be run with administrative privileges on each compute node.

Kubernetes on Red Hat Openshift uses a DaemonSet for this kind of cluster-wide services. A DaemonSet ensures that all (or some) Nodes run a copy of a Pod. In the Git repository, you can see a sample daemonset.yaml file that provides all the necessary settings and options to run the CEX tool plug-in as a Kubernetes on Red Hat Openshift DaemonSet.

Fortunately, the git repository also contains an “automated” solution at the [k8s-cex-dev-plugin GitHub repository](#).

This website shows you how to use a customized deployment to fully install the CEX device plug-in into a running cluster.

This folder provides various sub-folders for different deployments based on “customize overlays”. To install the IBM CEX device plug-in, first customize the ConfigMap in the tconfigmap/cex\_resources.json file, then use the customized ConfigMap and run the following command:

```
$ oc create -k rhocp-create
```

After successful deployment, check that all worker nodes are running the CEX device plug-in pod by using the command shown in Example 4-8.

#### *Example 4-8 Checking the CEX device plugin pods*

---

```
[root@bastion ~]# oc get pods -o wide
```

NAME	READY	STATUS	RESTARTS	AGE	NODE
cex-plugin-daemonset-4w4hg	1/1	Running	0	13d	iw2
cex-plugin-daemonset-d491d	1/1	Running	0	13d	iw1
cex-plugin-daemonset-qhdlb	1/1	Running	0	13d	aw3
cex-plugin-daemonset-scdss	1/1	Running	0	13d	aw1
cex-plugin-daemonset-wwdxz	1/1	Running	0	13d	iw3
cex-prometheus-exporter-79cfd9f78d-vcpgt	1/1	Running	0	13d	aw1

---

Figure 4-6 on page 100 shows our visualization of this deployment, anticipating that the pods to be launched are using CEX resources.

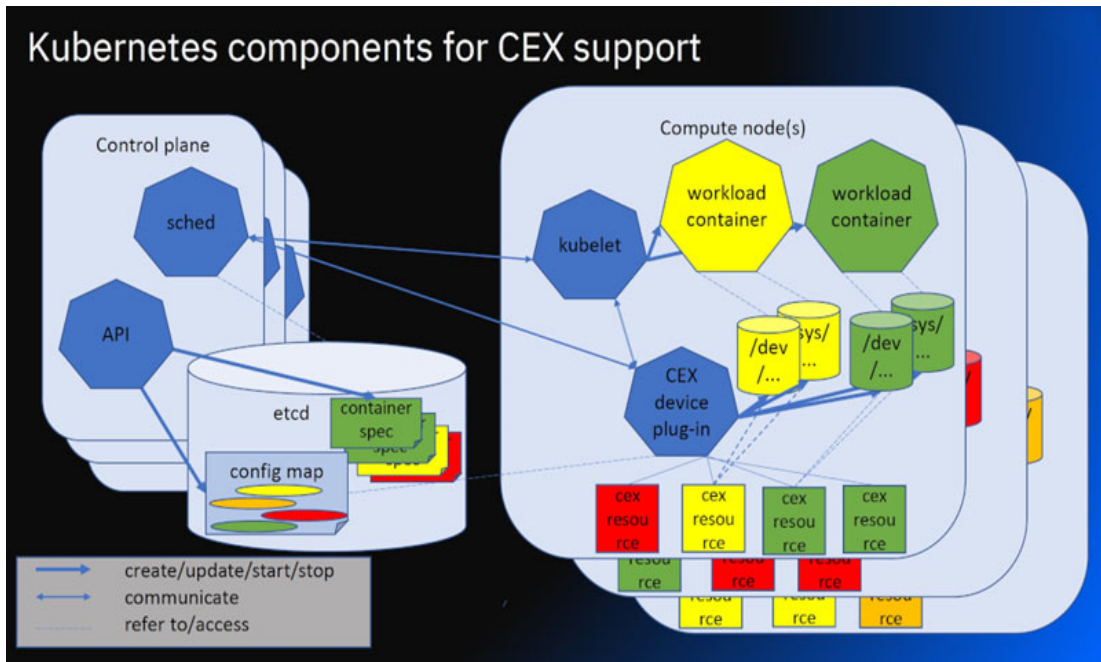


Figure 4-6 Kubernetes components for CEX support

The instances of the CEX device plug-in performs the following tasks on all compute nodes:

- ▶ Verify the availability of existing crypto resources on the nodes.
- ▶ Handle requests for CEX resources from containers.
- ▶ Claim the resource.
- ▶ Ensure that containers are scheduled to the correct compute node with the requested CEX crypto resources.

The CEX device plug-in performs the following tasks on each compute node:

- ▶ Filter the available CEX resources on the compute node and submit this information to the Kubernetes infrastructure service.
- ▶ Reserve and release a CEX resource at the request of the Kubernetes infrastructure based on the needs of a pod.

The application container would only need to determine that it needs a CEX resource from a given CEX configuration set by using the Kubernetes resource constraint declaration. The cluster and CEX device plug-in handle the details, requiring a CEX resource, and scheduling the pod to the appropriate compute node.

### ***Test the CEX device plug-in functionality***

Next, we demonstrate how to test whether what we envisioned works. For this purpose, in the lab environment, we created a simple RHEL image based on a RHEL ubi-9 base image. The Docker file for the RHEL image is shown in Example 4-9.

#### ***Example 4-9 Sample Docker file***

```
[root@bastion cex-plugin]# cat Dockerfile-ubi
FROM registry.access.redhat.com/ubi9/ubi
USER root
```



```
ENV BUILD_DEPS='cryptsetup kmod perl libica openssl openssl-libs openssl-pkcs11
openssl-ibmca time s390utils opencryptoki'
```

```
# Update image/install pkgs
RUN dnf update
RUN dnf install $BUILD_DEPS -y && rm -rf /var/cache/yum
```

```
RUN set -x \
&& mv /etc/pki/tls/openssl.cnf /etc/pki/tls/openssl.cnf.orig
```

```
RUN set -x \
&& echo "openssl_conf = openssl_def" >> /etc/pki/tls/openssl.cnf \
&& echo "" >> /etc/pki/tls/openssl.cnf \
&& echo "[openssl_def]" >> /etc/pki/tls/openssl.cnf \
&& echo "providers = provider_sect" >> /etc/pki/tls/openssl.cnf \
&& echo "alg_section = evp_properties" >> /etc/pki/tls/openssl.cnf \
&& echo "" >> /etc/pki/tls/openssl.cnf \
&& echo "[provider_sect]" >> /etc/pki/tls/openssl.cnf \
&& echo "default = default_sect" >> /etc/pki/tls/openssl.cnf \
&& echo "ibmca_provider = ibmca_sect" >> /etc/pki/tls/openssl.cnf \
&& echo "" >> /etc/pki/tls/openssl.cnf \
&& echo "[default_sect]" >> /etc/pki/tls/openssl.cnf \
&& echo "activate = 1" >> /etc/pki/tls/openssl.cnf \
&& echo "" >> /etc/pki/tls/openssl.cnf \
&& echo "[ibmca_sect]" >> /etc/pki/tls/openssl.cnf \
&& echo "identity = ibmca" >> /etc/pki/tls/openssl.cnf \
&& echo "module = ibmca-provider.so" >> /etc/pki/tls/openssl.cnf \
&& echo "activate = 1" >> /etc/pki/tls/openssl.cnf \
&& echo "fips = no" >> /etc/pki/tls/openssl.cnf \
&& echo "#debug = yes" >> /etc/pki/tls/openssl.cnf \
&& echo "algorithms = RSA,EC,DH" >> /etc/pki/tls/openssl.cnf \
&& echo "# Algorithm EC omitted for systems starting with IBM z15" >>
/etc/pki/tls/openssl.cnf \
&& echo "" >> /etc/pki/tls/openssl.cnf \
&& echo "#fallback-properties = provider=default" >> /etc/pki/tls/openssl.cnf \
&& echo "" >> /etc/pki/tls/openssl.cnf \
&& echo "[evp_properties]" >> /etc/pki/tls/openssl.cnf \
&& echo "default_properties = ?provider=ibmca" >> /etc/pki/tls/openssl.cnf
```

```
ADD openssl-test /opt/
```

```
CMD ["/bin/bash", "-c", "sleep infinity"]
```

---

Example 4-9 on page 100 simply creates a RHEL image, installs some useful packages (for the tests), and configures the openssl to use the ibmca provider.

### ***How can your containerized application take advantage of CEX resources?***

Container deployment can request a CEX resource from a crypto config set with a resource definition, an example of which is shown in Example 4-10.

#### *Example 4-10 Resource definition*

---

```
...
spec:
  containers:
    - image ...
      ...
      resources:
        limits:
```

**cex.s390.ibm.com/Accelerator\_for\_Lancelot : 1**

...

where:

<b>cex.s390.ibm.com</b>	<i>Resource type CEX resource</i>
<b>Accelerator_for_Lancelot</b>	<i>Name of crypto config set</i>
<b>: 1</b>	<i>Must be 1 and only one CEX resource per container allowed</i>

**Note:**

- ▶ At container start a compute node will be determined that has a free CEX resource from the specified crypto config set and the container will be assigned one of the CEX resources from the crypto config set.
- ▶ The CEX resource is chosen according to the config map valid at container start time.
- ▶ Later changes to the crypto config map do not affect running containers with CEX resources.
- ▶ When a container stops, its CEX resource is released.

Now we can deploy this image. Ensure that you have provided the definitions shown in Example 4-11.

*Example 4-11 Sample deployment configurations*

```
[root@bastion cex-plugin]# cat ubi-acc.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: ubi-acc
  namespace: knights
spec:
  replicas: 1
  selector:
    matchLabels:
      app: ubi-acc
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: ubi-acc
    spec:
      containers:
      - image: '<HERE is the repository and the IMAGE>'
        imagePullPolicy: Always
        name: <PODNAME>
        command: ["/bin/sh", "-c", "while true; do echo do-nothing-loop; sleep 30; done"]
      resources:
        limits:
          cex.s390.ibm.com/Accelerator_for_Lancelot: 1

[root@bastion cex-plugin]# cat ubi-cca.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
```

```

name: ubi-cca
namespace: knights
spec:
  replicas: 1
  selector:
    matchLabels:
      app: ubi-cca
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: ubi-cca
    spec:
      containers:
      - image: '<HERE is the repository and the IMAGE>'
        imagePullPolicy: Always
        name: <PODNAME>
        command: ["/bin/sh", "-c", "while true; do echo do-nothing-loop; sleep 30; done"]
      resources:
        limits:
          cex.s390.ibm.com/CCA-Coproc_for_Bedivere: 1

```

```

[root@bastion cex-plugin]# cat ubi-e11.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: ubi-ep11
  namespace: knights
spec:
  replicas: 1
  selector:
    matchLabels:
      app: ubi-ep11
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: ubi-ep11
    spec:
      containers:
      - image: '<HERE is the repository and the IMAGE>'
        imagePullPolicy: Always
        name: <PODNAME>
        command: ["/bin/sh", "-c", "while true; do echo do-nothing-loop; sleep 30; done"]
      resources:
        limits:
          cex.s390.ibm.com/EP11_for_Galahad: 1

```

---

There are several ways to verify a successful deployment. First, you can check the pod description to see if the desired resource is available by using the **oc describe pod** command, as shown in Example 4-12 and Example 4-13 on page 104.

*Example 4-12 Part of **oc describe pod** output of the pod where we requested EP11*

---

```

Limits:
  cex.s390.ibm.com/EP11_for_Galahad: 1
Requests:

```

```
cex.s390.ibm.com/EP11_for_Galahad: 1
```

*Example 4-13 Part of `oc describe pod` output of the pod where we requested accelerator*

```
Limits:
  cex.s390.ibm.com/Accelerator_for_Lancelot: 1
Requests:
  cex.s390.ibm.com/Accelerator_for_Lancelot: 1
```

We can check the node where we deployed the resources by using the `oc describe node aw1` command, as shown in Example 4-14.

*Example 4-14 Partial output of the `oc describe node aw1` command*

```
Allocated resources:
(Total limits may be over 100 percent, i.e., overcommitted.)
Resource                               Requests    Limits
-----
cpu                                     244m (6%)   0 (0%)
memory                                  1228Mi (8%) 0 (0%)
ephemeral-storage                       0 (0%)      0 (0%)
hugepages-1Mi                           0 (0%)      0 (0%)
cex.s390.ibm.com/Accel_for_Bedivere      0           0
cex.s390.ibm.com/Accelerator_for_Lancelot 1           1
cex.s390.ibm.com/CCA-Coproc_for_Bedivere 0           0
cex.s390.ibm.com/CCA_for_Lancelot        0           0
cex.s390.ibm.com/EP11_for_Galahad         1           1
Events:                                  <none>
```

Since our test pods provide a bash prompt when we log in (for example, in the Red Hat Openshift GUI, the pod's response to Terminal), there are several ways to check for functionality.

First, we can check what kind of crypto resource is available by using the `lsccrypt` command, as shown in Figure 4-7 and Figure 4-8 on page 105. Remember we are working *within* the pod now.

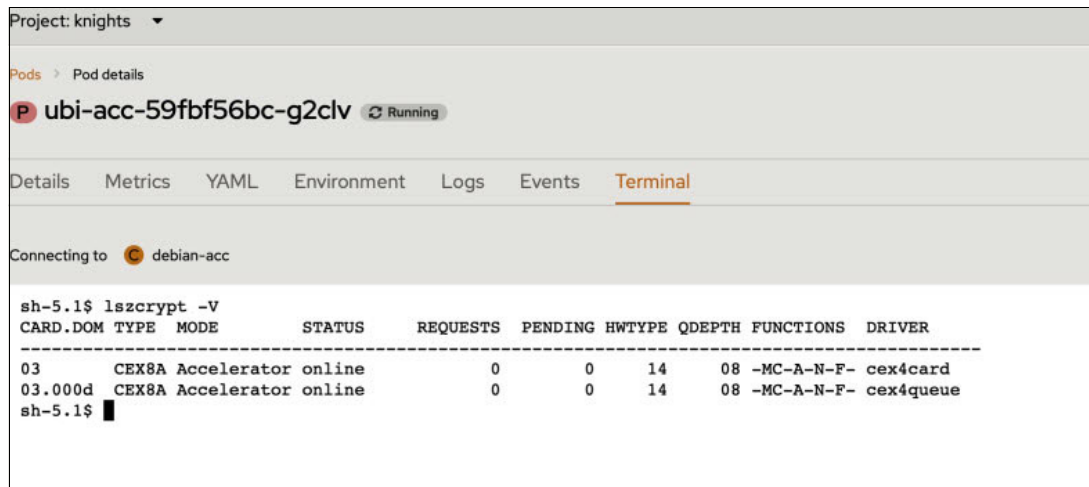


Figure 4-7 Checking resources

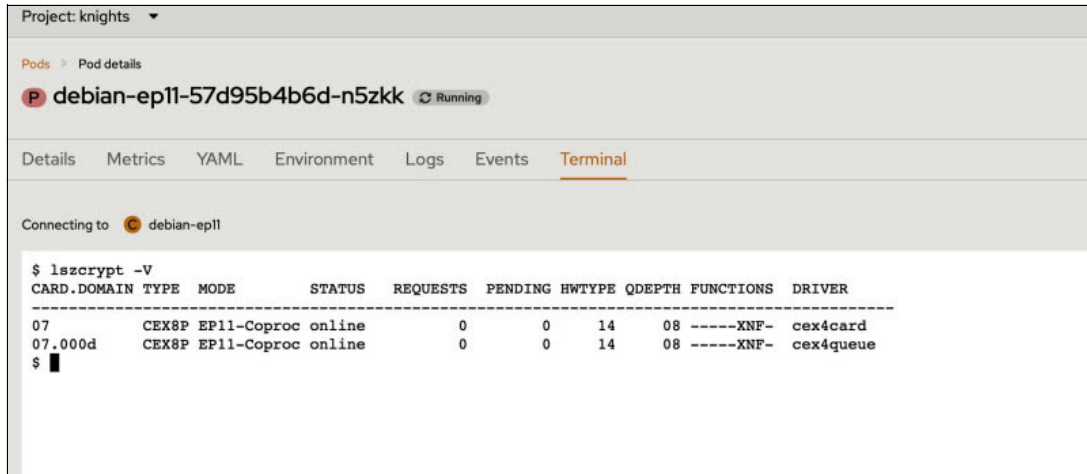


Figure 4-8 Checking resource

Another test would be if we use the configured OpenSSL, and we are checking the speed there. Before and after running the test we can check a status by using the command shown in Example 4-15 (focus in example is on the relevant portion only).

Example 4-15 Testing openssl-rsa command

```
sh-5.1$ icastats
```

function	hardware			software		
	ENC	CRYPT	DEC	ENC	CRYPT	DEC
SHA-1		0			0	
SHA-224		0			0	
...						
X25519 Keygen		0			0	
X25519 Derive		0			0	
X448 Keygen		0			0	
X448 Derive		0			0	
RSA-ME		0			0	
RSA-CRT		0			0	
DES ECB	0		0	0		0
DES CBC	0		0	0		0
...						
AES GCM	0		0	0		0

```
sh-5.1$ openssl speed rsa4096
Doing 4096 bits private rsa's for 10s: 2626 4096 bits private RSA's in 0.36s
Doing 4096 bits public rsa's for 10s: 58739 4096 bits public RSA's in 0.04s
version: 3.0.7
```

```

          sign    verify    sign/s verify/s
rsa 4096 bits 0.000137s 0.000001s 7294.4 1468475.0

```

```
sh-5.1$ icastats
```

function	hardware			software		

	ENC	CRYPT	DEC	ENC	CRYPT	DEC
SHA-1		0			0	
SHA-224		0			0	
...						
X448 Keygen		0			0	
X448 Derive		0			0	
RSA-ME		58823			0	
RSA-CRT		2627			0	
...						
AES GCM	0		0	0		0

The openssl-rsa test has driven the CEX resource within the pod.

We also compared non-accelerated and accelerated openssl-rsa speeds inside pods. The results were convincing for us and proved that using CEX resources within a pod works. The results from our lab environment are shown in Example 4-16.

*Example 4-16 Test result comparing non-accelerated and accelerated openssl-rsa speeds*

	sign	verify	sign/s	verify/s
Accelerated rsa 2048 bits	0,00004	0,00000	24022,60	1322975,00
Non-accele. rsa 2048 bits	0,00238	0,00006	419,60	18001,70
Multiplicator	57	56	57	73
Accelerated rsa 4096 bits	0,00013	0,00000	7517,60	130100,00
Non-accele. rsa 4096 bits	0,01528	0,00019	65,40	5226,10
Multiplicator	115	191	115	216

There are other use cases more useful than the ones presented here. Our goal is to demonstrate the configuration and the solution, and to test the success of the operation.



## Guest and workload considerations for using an HSM in the cloud

In this chapter, we discuss:

- ▶ “Determining the right HSM” on page 108
- ▶ “openCryptoki” on page 110
- ▶ “dm-crypt” on page 116
- ▶ “Crypto Express support for Secure Execution” on page 117

## 5.1 Determining the right HSM

In 2.2, “Crypto Express configuration” on page 13, we showed how to configure CEX devices and the crypto domains on the HMC/DPM consoles for each LPAR on an IBM Z or LinuxONE server. On those consoles, we not only had to make the device available but also specify which domains would be available in the LPAR to which we were attaching the CEX adapters. A z/VM or KVM host will only detect those adapters and domains assigned to the z/VM or KVM LPARs. They appear to the hypervisor and can be used by virtual machines. This section shows how z/VM “sees” crypto resources as virtual devices represented by a *crypto ID* and a *domain index*. Additionally, we show how a user can determine if the guest uses the right HSM and domains.

Example 5-1 provides the query that will show the crypto domains assigned to a z/VM host (LPAR). In this example, the guests that were using the adapters were not running. Adapters #000 and #002 are configured as CCA cryptographic coprocessor (CEX8C) mode and adapters #001 and #003 are configured as a PKCS #11 cryptographic coprocessor or EP11(CEX8P) mode.

*Example 5-1 Domains and adapters assigned to the z/VM LPAR*

---

```
query crypto domain users
22:23:07 AP 000 CEX8C Domain 020 operational online shared
22:23:07 AP 000 CEX8C Domain 021 operational online free
22:23:07 AP 001 CEX8P Domain 020 operational online free
22:23:07 AP 001 CEX8P Domain 021 operational online free
22:23:07 AP 002 CEX8C Domain 020 operational online shared
22:23:07 AP 002 CEX8C Domain 021 operational online free
22:23:07 AP 003 CEX8P Domain 020 operational online free
22:23:07 AP 003 CEX8P Domain 021 operational online free
22:23:07
22:23:07 There are no shared crypto users.
Ready; T=0.01/0.01 22:23:07
```

---

Domains 20 and 21 from adapters (HSM) 000, 001, 002 and 003 are assigned to the z/VM LPAR and are available to z/VM guests.

Example 5-2 shows the command used to display the crypto domains assigned to z/VM guests, BASTION and BASTION2. In this example, the guests that were using the adapters were running. This command is issued on the z/VM host.

*Example 5-2 Domain and adapters assigned to z/VM guests BASTION and BASTION2*

---

```
query crypto domain users
14:53:00 AP 000 CEX8C Domain 020 operational online shared
14:53:00 AP 000 CEX8C Domain 021 operational online attached to BASTION2
14:53:00 AP 001 CEX8P Domain 020 operational online attached to BASTION
14:53:00 AP 001 CEX8P Domain 021 operational online attached to BASTION
14:53:00 AP 002 CEX8C Domain 020 operational online shared
14:53:00 AP 002 CEX8C Domain 021 operational online attached to BASTION2
14:53:00 AP 003 CEX8P Domain 020 operational online attached to BASTION
14:53:00 AP 003 CEX8P Domain 021 operational online attached to BASTION
14:53:00
14:53:00 There are no shared crypto users.
```

---



Domains 20 and 21 from adapters (HSM) 001 and 003 have been assigned to the z/VM guest, BASTION. Domain 21 from adapters (HSM) 000 and 002 have been assigned to the z/VM guest, BASTION2.

To verify whether the guest, BASTION, is using the correct HSM and domains, issue the command shown in Example 5-3. This command is issued via SSH to the BASTION guest and lists all devices grouped by cryptographic device. Card and domain IDs are in hexadecimal.

*Example 5-3 Query the crypto status from z/VM guest, BASTION*

---

```
[root@bastion ~]# hostname
bastion
[root@bastion ~]# lszcrypt
```

CARD.DOM	TYPE	MODE	STATUS	REQUESTS
01	CEX8P	EP11-Coproc	online	0
01.0014	CEX8P	EP11-Coproc	online	0
01.0015	CEX8P	EP11-Coproc	online	0
03	CEX8P	EP11-Coproc	online	0
03.0014	CEX8P	EP11-Coproc	online	0
03.0015	CEX8P	EP11-Coproc	online	0

```
[root@bastion ~]#
```

---

Domains 20 (Hex 14) and 21 (Hex 15) from adapters 001 and 003 are available to z/VM Guest BASTION, and that matches the output shown in Example 5-2 on page 108.

From Example 5-2 on page 108, domain 21 from HSM 000 and 002 are assigned to the BASTION2 guest.

To verify whether the guest, BASTION2, use the correct HSM and domains, use the command shown in Example 5-4. This command is issued using SSH to the BASTION2 guest and lists all devices grouped by cryptographic device. Card and domain IDs are hexadecimal values.

*Example 5-4 Query the crypto status from z/VM guest, BASTION2*

---

```
[root@bastion2 ~]# hostname
bastion2
[root@bastion2 ~]# lszcrypt
```

CARD.DOM	TYPE	MODE	STATUS	REQUESTS
00	CEX8C	CCA-Coproc	online	1
00.0015	CEX8C	CCA-Coproc	online	1
02	CEX8C	CCA-Coproc	online	0
02.0015	CEX8C	CCA-Coproc	online	0

```
[root@bastion2 ~]#
```

---

From Example 5-4, domain 21 (Hex 15) from adapters 000 and 002 are available to z/VM guest, BASTION2, and that matches the output shown in Example 5-2 on page 108.

## 5.2 openCryptoki

openCryptoki is an open-source implementation of Cryptoki (cryptographic token interface). openCryptoki provides a standard programming interface between applications and all kinds of portable cryptographic devices (such as a Crypto Express adapter) that hold cryptographic information and perform cryptographic functions. For more information, see [Linux on IBM Z and IBM LinuxONE openCryptoki - An Open Source Implementation of PKCS #11, SC34-7730](#).

### 5.2.1 Slots and tokens

openCryptoki consists of an implementation of the PKCS #11 API, a slot manager, an API for slot token dynamic link libraries (STDLLs), and a set of STDLLs (or tokens).

A slot is similar to a smart card reader. In the same way a smart card is inserted into a smart card reader, a PKCS #11 token is inserted into a PKCS #11 slot, where the slot is identified by its ID.

A token is a library code that knows how to interface with the cryptographic hardware.

The slot manager provides the number of configured tokens to applications and it interacts with the tokens that are used by the applications. For each device with which a token should be associated, this token must be defined in a slot in the openCryptoki configuration file, `/etc/opencryptoki/opencryptoki.conf`. This allows for proper sharing of state information between applications to help ensure conformance with the PKCS #11 specification. The openCryptoki library loads the tokens that provide hardware or software-specific support for cryptographic functions.

openCryptoki allows for the management of multiple tokens that can be used in parallel by one or more processes or applications. openCryptoki supports different token types for software tokens and for various forms of hardware support, for example, IBM Crypto Express adapters. These multiple tokens can have the same or different types. For example, the EP11 token type is an STDLL introduced with openCryptoki version 3.1.

For more information on openCryptoki token types, see chapters 13-19 in [Linux on IBM Z and IBM LinuxONE openCryptoki - An Open Source Implementation of PKCS #11, SC34-7730](#).

Figure 5-1 on page 111 shows the process flow within the Linux on IBM Z and IBM LinuxONE crypto stack.

An application sends an encryption request to the crypto adapter. Through various interfaces, the request is propagated from the application layer down to the target crypto adapter. The request passes through the following layers:

- ▶ The standard openCryptoki interfaces
- ▶ The relevant IBM Z crypto libraries
- ▶ The operating system kernel

The zcrypt device driver finally sends the request to the appropriate cryptographic coprocessor. The resulting request output is sent back to the application in reverse, through the layer interfaces.

For more information, see [Linux on IBM Z and IBM LinuxONE openCryptoki - An Open Source Implementation of PKCS #11, SC34-7730](#).

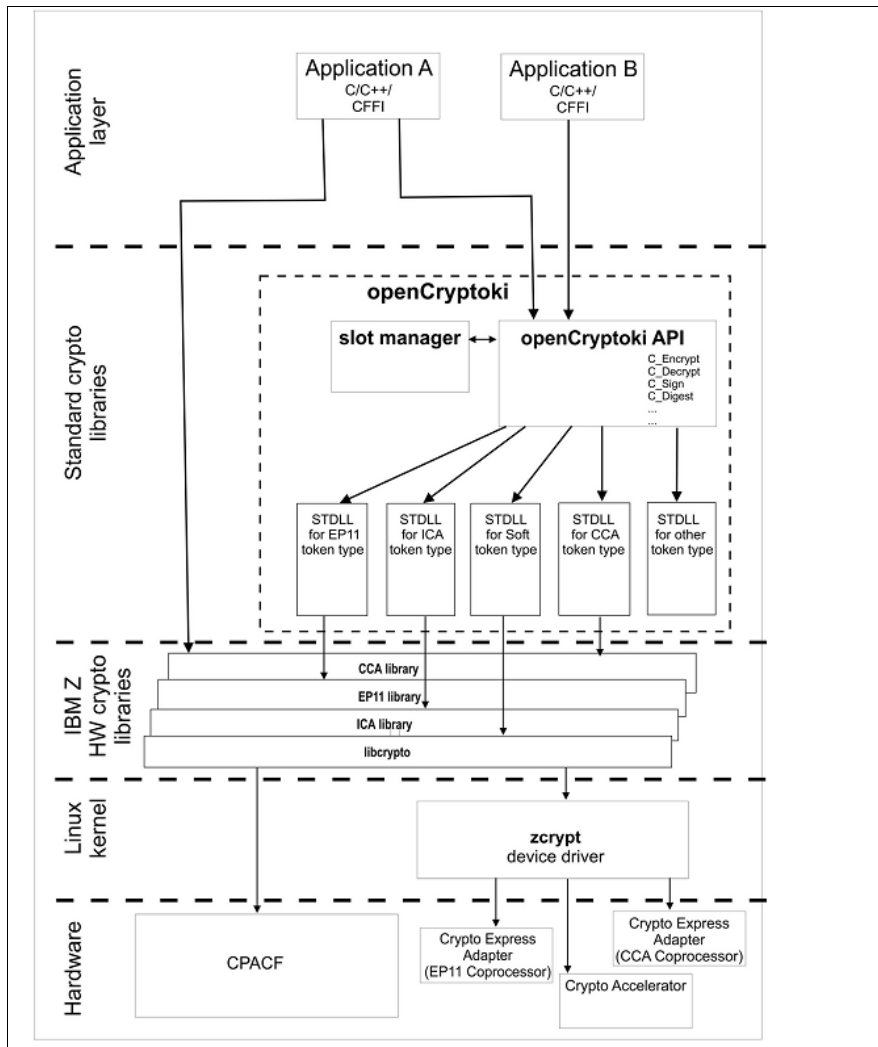


Figure 5-1 Linux on IBM Z crypto infrastructure

For more information on the openCryptoki architecture, see [Common features of openCryptoki](#).

## 5.2.2 Installation of openCryptoki

To install, follow the instructions provided in “Chapter 4. Installing openCryptoki” in [Linux on IBM Z and IBM LinuxONE openCryptoki - An Open Source Implementation of PKCS #11, SC34-7730](#).

## 5.2.3 Configuration of openCryptoki

You can configure openCryptoki (the set of tokens including their respective token types) by using the `opencrytpoki.conf` file. You can define and exploit multiple tokens of any token type, each with a different token name. For multiple EP11 tokens, you can configure them by using a token-specific configuration file for each token instance. For the most current default `opencrytpoki.conf` file, see the [opencrytpoki GitHub](#) repository.

A sample token-specific configuration file is shown in Example 5-5. For the complete file, see [Adjusting the openCryptoki configuration file](#).

*Example 5-5 Sample token-specific opencryptoki.conf file*

---

```
version opencryptoki-3.22
# The following defaults are defined:
# hwversion = "0.0"
# firmwareversion = "0.0"
# description = Linux
# manufacturer = IBM
#
# The slot definitions below may be overridden and/or customized.
# For example:
# slot 0
# {
# stdll = libpkcs11_cca.so
# description = "OCK CCA Token"
# manufacturer = "MyCompany Inc."
# hwversion = "2.32"
# firmwareversion = "1.0"
# }
# See man(5) opencryptoki.conf for further information.
#
disable-event-support # not part of the default config file. Enabled by default.
statistics (on,implicit,internal) # not part of the default config file. enable or
# disable the collection of statistics about mechanism usage
slot 0
{
stdll = libpkcs11_tpm.so
tokversion = 3.12
}
slot 1
{
stdll = libpkcs11_ica.so
tokversion = 3.12
}
slot 2
{
stdll = libpkcs11_cca.so
confname = ccatok.conf
tokversion = 3.12
}
slot 3
{
stdll = libpkcs11_sw.so
tokversion = 3.12
}
slot 4
{
stdll = libpkcs11_ep11.so
confname = ep11tok.conf
tokversion = 3.12
}
}
```

---

Each token uses a unique token directory. This token directory receives the token-individual information (like for example, key objects, user PIN, SO PIN, or hashes). Thus, the information for a certain token is separated from all other tokens. For example, for most Linux distributions, the CCA token directory is `/var/lib/openssl/cctok`. The CCA token is called `cctok`, if there is only one instance of a CCA token, and no explicit name is defined in the `openssl` configuration file.

For more information on the `openssl` configuration file, see “Chapter 5. Adjusting the `openssl` configuration file” in [Linux on IBM Z and IBM LinuxONE openssl - An Open Source Implementation of PKCS #11, SC34-7730](#).

## 5.2.4 Managing tokens

[Linux on IBM Z and IBM LinuxONE openssl - An Open Source Implementation of PKCS #11, SC34-7730](#) and [pkcsconf man page](#) discuss the `pkcsconf` utility. This utility is a command-line program (`/sbin/pkcsconf`) that can be used to configure and administer tokens that are supported within the system.

Options available in this command include the ability to do the following:

- ▶ Display slot information (Example 5-6).

*Example 5-6 Display slot information:*

---

```
# pkcsconf -s

Slot #1 Info
  Description: ICA Token
  Manufacturer: IBM
  Flags: 0x1 (TOKEN_PRESENT)
  Hardware Version: 4.0
  Firmware Version: 2.11
...
...
Slot #4 Info
  Description: EP11 Token
  Manufacturer: IBM
  Flags: 0x1 (TOKEN_PRESENT)
  Hardware Version: 4.0
  Firmware Version: 2.10
```

---

- ▶ Show which slot is available (Example 5-7).

*Example 5-7 Display available slot*

---

```
# pkcsconf -tis
PKCS#11 Info
  Version 2.20
  Manufacturer: IBM
  Flags: 0x0
  Library Description: Meta PKCS11 LIBRARY
  Library Version 3.10
Token #3 Info:
  Label: IBM OS PKCS#11 1
  Manufacturer: IBM Corp.
  Model: IBM SoftTok
  Serial Number: 123
```

```

Flags: 0x880045
(RNG|LOGIN_REQUIRED|CLOCK_ON_TOKEN|USER_PIN_TO_BE_CHANGED|SO_PIN_TO_BE_CHANGED)
Sessions: 0/18446744073709551614
R/W Sessions: 18446744073709551615/18446744073709551614
PIN Length: 4-8
Public Memory: 0xFFFFFFFFFFFFFFFF/0xFFFFFFFFFFFFFFFF
Private Memory: 0xFFFFFFFFFFFFFFFF/0xFFFFFFFFFFFFFFFF
Hardware Version: 1.0
Firmware Version: 1.0
Time: 12:35:01
Slot #3 Info
Description: Linux
Manufacturer: IBM
Flags: 0x1 (TOKEN_PRESENT) 1
Hardware Version: 0.0
Firmware Version: 0.0

```

---

- ▶ Display token information (Example 5-8).

*Example 5-8 Display token information*

```

# pkcsconf -t
...
...
Token #4 Info:
Label: ep11tok
Manufacturer: IBM
Model: EP11
Serial Number: 93AABC5H53107366
Flags: 0x880045
(RNG|LOGIN_REQUIRED|CLOCK_ON_TOKEN|USER_PIN_TO_BE_CHANGED|SO_PIN_TO_BE_CHANGED)
Sessions: 0/[effectively infinite]
R/W Sessions: [information unavailable]/[effectively infinite]
PIN Length: 4-8
Public Memory: [information unavailable]/[information unavailable]
Private Memory: [information unavailable]/[information unavailable]
Hardware Version: 7.24
Firmware Version: 3.1
Time: 2021031912021700

```

---

- ▶ Initialize a token (Example 5-9).

*Example 5-9 Initialize a token*

```

$ pkcsconf -I -c <slot> /* Initialize the Token and set up a Token Label */
$ pkcsconf -P -c <slot> /* change the SO PIN (recommended) */
$ pkcsconf -u -c <slot> /* Initialize the User PIN (SO PIN required) */
$ pkcsconf -p -c <slot> /* change the User PIN (optional) */

```

---

## 5.2.5 Generating and listing keys

[Linux on IBM Z and IBM LinuxONE openCryptoki - An Open Source Implementation of PKCS #11, SC34-7730](#) and [Managing token keys - p11sak utility](#) discuss the p11sak utility.

This utility can be used to manage token keys and certificates in an openCryptoki token repository with their PKCS #11 attributes.

Subcommands of this utility allow you to do the following:

- ▶ Generate keys in the openCryptoki repository.
- ▶ List the keys in the repository.
  - The tool supports the listing of:
    - symmetric keys (AES, 3DES, DES) with PKCS #11 attributes
    - asymmetric keys (RSA, EC) with PKCS #11 attributes
    - public, private and secure keys
    - all keys of any type
- ▶ Remove the keys from the repository.
- ▶ Set attributes of keys.
- ▶ Copy the keys in the repository.
- ▶ Import and export keys from and to a binary file or a PEM file (PEM is a container file format often used to store cryptographic keys).
- ▶ List certificates in the repository.
- ▶ Remove certificates from the repository.
- ▶ Set or update attributes of certificates.
- ▶ Copy certificates in the repository.
- ▶ Import and export certificates from and to a binary file or a PEM file.
- ▶ Extract public keys from certificates.

## 5.2.6 Token specifications

[Linux on IBM Z and IBM LinuxONE openCryptoki - An Open Source Implementation of PKCS #11, SC34-7730](#) and [Token specifications](#) discuss token specifications. Token specifications refer to the characteristics and configurations of the crypto tokens supported by the openCryptoki framework.

Token specifications include details such as the following:

- ▶ Information about the cryptographic algorithms supported by the token, such as encryption algorithms (AES, DES), hash functions (SHA-1, SHA-256), and digital signature algorithms (RSA, ECDSA).
- ▶ Specifications regarding the storage of cryptographic keys, including the maximum number of keys supported, key lengths, and key types (symmetric, asymmetric).
- ▶ Details about security measures provided by the token, such as PIN policies (length, complexity, retry limits), authentication mechanisms, tamper resistance, and compliance with industry standards (FIPS 140-2, for example).
- ▶ Information about supported functionalities and operations, such as key generation, key import/export, encryption, decryption, signing, and verification.
- ▶ Specifications related to token management operations, including initialization, re-initialization, backup, and restore procedures.

These specifications assist developers and administrators in ensuring compatibility, security and effective utilization of cryptographic tokens through the standardized interface provided by PKCS#11.

## 5.3 dm-crypt

In Linux, the most popular method for end-to-end data at-rest encryption is full volume encryption using the dm-crypt kernel component.

dm-crypt is a transparent disk encryption system that allows you to encrypt entire block devices. It operates as part of the device mapper (hence the “dm” in dm-crypt), which is a kernel framework for mapping physical block devices into higher-level virtual block devices. This encryption layer intercepts data going to and from a block device, encrypting it before it is written to the disk and decrypting it as it's read back.

Key features of dm-crypt include:

- ▶ Full Disk Encryption: It enables you to encrypt entire disks or partitions, protecting the data stored on them.
- ▶ Transparent Encryption: Once set up, encryption and decryption are done automatically and transparently to the user and applications. Users don't need to manage encryption and decryption processes directly.
- ▶ Pluggable Encryption Algorithms: It supports various encryption algorithms such as AES (Advanced Encryption Standard), Twofish, Serpent, and others, which allows users to choose the encryption algorithm that suits their security and performance needs.
- ▶ Passphrase and Keyfile Support: dm-crypt allows the use of passphrases or keyfiles to unlock encrypted devices.
- ▶ Integration with LUKS: Often, dm-crypt is used in conjunction with LUKS (Linux Unified Key Setup), which provides a standard format for disk encryption, managing encryption keys, and storing metadata about the encrypted volumes.

Encrypting disks by using dm-crypt can provide a significant level of security, especially for systems that handle sensitive data. It helps protect data at-rest, preventing unauthorized access to the information if the device is lost or stolen. Users can set up dm-crypt during the installation of Linux or manually configure it post-installation to encrypt disks or partitions, adding an extra layer of security to their system.

Figure 5-2 on page 117 provides an overview of dm-crypt and end-to-end data encryption. To encrypt volumes, use dm-crypt with the protected-key cipher, `paes`. The encryption keys are protected by a Crypto Express adapter. Protected volume encryption requires that the Linux kernel include support for the protected AES cipher (`paes_s390`) that automatically includes the `pkey` module.



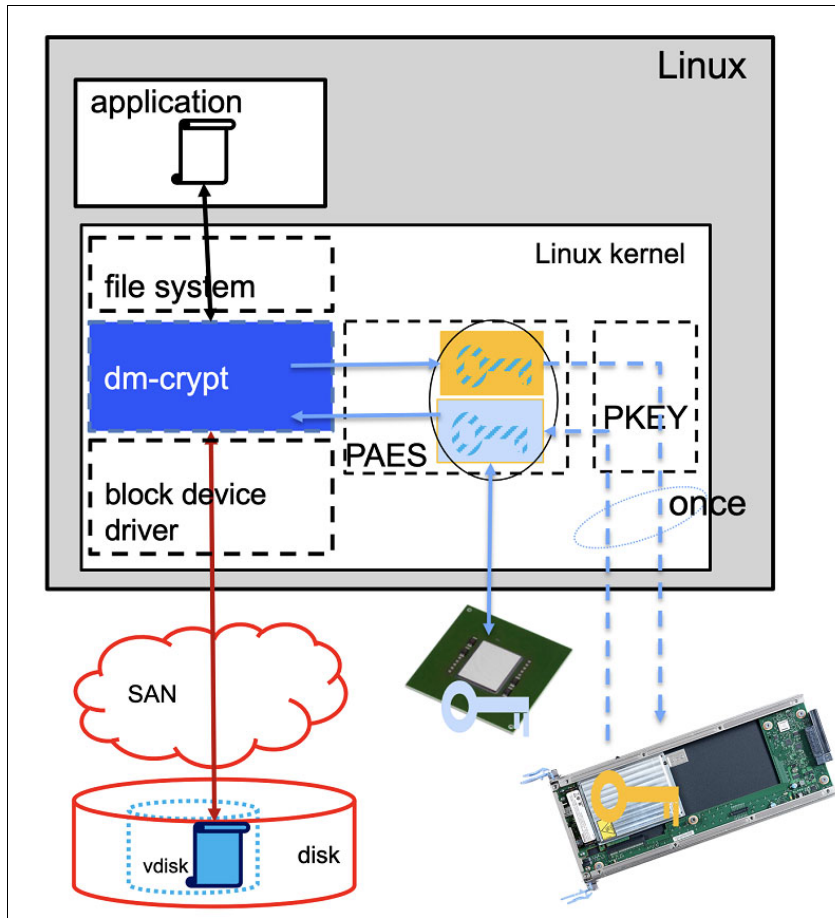


Figure 5-2 End-to-end data encryption

For a discussion of protected and secure volume encryption, see [Protected and secure volume encryption](#).

For additional guidance on installation and configuration, see [Encrypting volumes by using dm-crypt](#).

### 5.3.1 Installation and configuration overview

To configure the disk partitions and optionally set up an encrypted partition or logical volume to encrypt all sensitive data, use the following steps.

## 5.4 Crypto Express support for Secure Execution

This chapter describes the setup procedure required to run a crypto load within Secure Execution (SE) guests with Crypto Express (CEX) support. Secure Execution is an implementation of Confidential Computing. Crypto Express supports means to make the Crypto resources often also named HSMs (Hardware Security Modules) available within SE guests. This chapter comprises the additional steps to prepare the SE guest and the arrangements required at runtime of the SE guest to securely access these crypto resources.

CEX support for Secure Execution combines two already existing concepts:

- ▶ Secure Execution support for KVM guests:  
Secure Execution is a feature to run secured KVM guests in a cloud environment protected from access from the cloud infrastructure provider. More details can be found at the IBM online documentation [IBM Secure Execution for Linux](#), and [Introducing IBM Secure Execution for Linux, SC34-7721](#).
- ▶ AP pass-through support for KVM guests:  
*VFIO pass-through for AP devices* or short *AP pass-through* is an KVM feature to provide crypto resources available on the KVM host to KVM guests. Each guest defines a mediated device which comprises the crypto resources to claim from the host and forward to the guest. More details can be found at the IBM online documentation, [Setting up cryptographic adapter resources for VFIO](#), and in the IBM publication [KVM Virtual Server Management, SC34-2752-08](#).

CEX support for secure execution (SE) is a new feature available with z16 with GA 1.5 in combination with CEX8S only.

Before using CEX support for Secure Execution make sure you have read and understood the security requirements stated in 5.4.8, “Important considerations for the secure use of Crypto Express adapters in EP11 mode” on page 126.

## 5.4.1 Terms and concepts related to secure execution with CEX support

The following are commonly used terms and concepts related to secure execution.

- ▶ Ultravisor  
The Ultravisor (UV) is some part of the IBM Z firmware which handles most of the low level parts for Secure Execution and AP pass-through. The KVM host and the KVM SE guests interact with the UV through a firmware API. The UV is considered trustworthy whereas the KVM host running the hypervisor is not. The states of crypto resources related to SE guests with AP pass-through and the crypto operations on these resources are supervised by the UV to make sure that a) only the permitted guest may access the resource and b) only allowed operations (based on the state) are executed. So the UV does the book keeping of each SE guest and its associated crypto resources (APQNs) and holds space for up to 12 *AP Association Secrets* per SE guest.
- ▶ AP Association Secret  
An *AP association secret* or short *secret* in the context of SE with AP pass-through is a 32 byte random value. It is used by the UV to derive unique authentication data to create an EP11 session for each EP11 crypto resource (APQN). An *AP Association Secret* should be prepared on a trusted machine only and encrypted with the public key (certificate) of the target host. The encrypted secret can be transferred to the SE guest without any security risks. The SE guest will push the encrypted secrets to use through UV API into the UV on the target host maybe during startup. The UV can decrypt the secret with the help of the private host key available in the HSA memory. So the secret value is not exposed to the hypervisor or even the SE guest but in clear only visible to the creating party (a trusted machine) and the consuming UV (within a trusted firmware environment).
- ▶ Master Key, Wrapping Key, MKVP  
The *Master Key*, with EP11 often named as *Wrapping Key*, is used inside an HSM to encrypt (wrap) and decrypt (unwrap) the key value of a secure key. For more details see 1.2, “Cryptographic terms” on page 4. A secure key usually comprises not only the encrypted key value but also key attributes. An EP11 secure key for example holds 16 bytes Master Key Verification Pattern (MKVP). This is a cryptographic hash value over the Master Key used to encrypt the secure key.

By examining this MKVP value within the secure key it is determinable which HSM can work with the secure key. The MKVP is retrievable from the HSM, for example Linux exposes this value in sysfs at `/sys/devices/ap/card<xx>/<xx>.<yyyy>/mkvps`.

► EP11 Session, Session-bound Key

An EP11 HSM allows to tighten the usage of EP11 secure keys by restricting the key to an EP11 session. To open up an EP11 session one needs to create or open a session by login in with credential information given. All secure keys generated within a session are then constrained to this session and called *session-bound* keys. An EP11 session is not a one-time session but may be closed and reopened at a later time. So a session-bound working key may be reused with a session reopened with same login credentials on the same APQN or another APQN with matching MK setup.

## 5.4.2 Secret preparation for SE guests with CEX support

To run crypto load on an EP11 APQN within an SE guest, one needs to provide an AP association secret. This secret is used during the association step, which is described in more detail in “EP11 coprocessor APQN” on page 122.

On a trusted machine (this may be a trusted Linux system on s390 with the s390-tools installed or a local PC with the x86 version of the s390-tools) AP association secrets should be generated for each EP11 crypto resource intended to be used within the SE guest.

To create a new AP association secret use the `pvsecret` application:

```
pvsecret create --output <encrypted_secret_filename> --hdr
<SE_guest_image_header_file> --host-key-document <public_host_key> --no-verify
association "<secret_name>"
```

This generates a new random secret value, encrypts the value with the public host key of the target host `<public_host_key>` and stores it into the given file `<encrypted_secret_filename>`.

The required `<SE_guest_image_header_file>` can be extracted from the Secure Execution image with:

```
pvextract-hdr -o <SE_guest_image_header_file> <SE_guest_image>
```

Figure 5-3 on page 120 shows a real-world example of the invocation. The sample shows all the steps from the SE guest image creation up to the creation of one secret named “SECRET1”. The sample also shows that the `pvsecret` command additionally creates an association secret info file `<secret_name>.yaml` which shows additional information like the secret id. The secret id is a simple sha-256 hash of the `<secret_name>` but may be used to uniquely identify this association secret.

```
File Edit View Bookmarks Plugins Settings Help
root@t96lp03:~/freude/guest# genprotimg -i vmlinuz -r initrd.img -p params --no-verify -k HKD-3931-a96.crt -o sequest.img
WARNING: host-key document verification is disabled. Your workload is not secured.
root@t96lp03:~/freude/guest#
root@t96lp03:~/freude/guest# pvextract-hdr -o sequesthdr.bin sequest.img
SE header found at offset 0x014000
SE header written to 'sequesthdr.bin' (640 bytes)
root@t96lp03:~/freude/guest#
root@t96lp03:~/freude/guest# pvsecret create --output secret1.bin --hdr sequesthdr.bin --host-key-document HKD-3931-a96.crt
--no-verify association "SECRET1"
Host-key document verification is disabled. The secret may not be protected.
Successfully generated the request
Successfully wrote association info to 'SECRET1.yaml'
root@t96lp03:~/freude/guest#
root@t96lp03:~/freude/guest# cat SECRET1.yaml
!Association
name: SECRET1
id: 0x03153249db7ce46b0330ffb1a760b59710531af08ec4d7f8424a6870fae49360
root@t96lp03:~/freude/guest# █
```

Figure 5-3 Sample pvsecret create invocation

The Ultravisor uses the secret value to derive login credentials to open up an EP11 session on the APQN associated with this secret. So the secret is one pillar of the security concept for CEX support for Secure Execution.

The created file is encrypted with the public host key and only the Ultravisor on the target host can decrypt and access the secret value. The file needs to be available within the SE guest image to prepare the APQN for crypto load. It may be packaged with the SE guest image or located on a file system mounted during SE guest startup.

The `pvsecret` and `pvextract-hdr` command-line tools are part of the `s390-tools` package as of version 2.29. All recent Linux distributors build and offer the `s390-tools` package also for x86 with limited content which includes these applications.

### 5.4.3 KVM host setup for SE guests with CEX support

KVM SE host support with AP pass-through is available as of Linux kernel version 6.6<sup>1</sup>, and `qemu` version 8.2<sup>2</sup>. IBM is working with its Linux Distribution partners to include these features into the future distribution releases.

There is no special setup needed for the KVM host to run SE guests with AP pass-through other than the KVM host requirements for SE guests without crypto support. A KVM guest with Secure Execution support and mediated device setup to forward the crypto resources into the guest is set up following the combination of both instruction papers mentioned in the introducing paragraph of this chapter.

Note that SE guests with AP pass-through support require a machine level of at least z16 with firmware level.

CEX support for SE guests is restricted to CEX8S (and newer) crypto express cards.

### 5.4.4 KVM guest setup for SE guests with CEX support

KVM SE guest support with AP pass-through is available as of Linux kernel version 6.6<sup>1</sup>, for the command-line tools `s390-tools` version 2.29 is required. IBM is working with its Linux Distribution partners to include these features into the future distribution releases.

<sup>1</sup> check with `uname -a` on the command line

<sup>2</sup> check with `qemu-system-s390x --version` on the command line

Before the crypto resources can be used within an SE guest, a short preparation procedure needs to be followed. The procedure varies based on the mode of the HSM and is described in the following paragraphs.

### Accelerator APQN

An APQN in Accelerator mode needs to be bound to the SE guest by the UV before any crypto load can be addressed to this crypto resource. Figure 5-4 shows the two states of an Accelerator APQN.

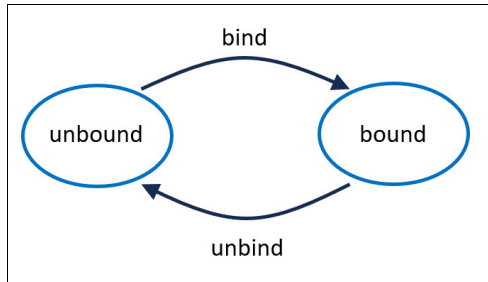


Figure 5-4 Accelerator APQN SE states

Within the SE guest `lszcrypt -V` shows an additional column **SESTAT** that displays the SE state of the APQNs. Figure 5-5 shows an example of this enhanced `lszcrypt` output:

Edit View Bookmarks Plugins Settings Help										
@sequest:~# lszcrypt -V										
.DOM	TYPE	MODE	STATUS	REQUESTS	PENDING	HWTYPE	QDEPTH	FUNCTIONS	DRIVER	SESTAT
	CEX8A	Accelerator	online	0	0	14	08	-MC-A-N-F-	cex4card	-
014	CEX8A	Accelerator	online	0	0	14	08	-MC-A-N-F-	cex4queue	unbound
	CEX8P	EP11-Coproc	online	0	0	14	08	-----XN-F-	cex4card	-
014	CEX8P	EP11-Coproc	online	0	0	14	08	-----XN-F-	cex4queue	unbound

Figure 5-5 `lszcrypt` verbose output with **SESTAT** column

An *unbound* Accelerator APQN is treated as insecure, the Linux crypto device driver will refuse to address this APQN for any crypto requests. To bind the APQN `<aa.dddd>` to this SE guest use the `chzcrypt` command-line application:

```
chzcrypt --se-bind <aa.dddd>
```

After successful binding an Accelerator APQN, it is usable for any (clear key) crypto load. `lszcrypt` should also show the new state of the APQN as *usable*:

File Edit View Bookmarks Plugins Settings Help										
root@sequest:~# chzcrypt --se-bind 0f.0014										
root@sequest:~# lszcrypt -V										
CARD.DOM	TYPE	MODE	STATUS	REQUESTS	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
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

Figure 5-6 `lszcrypt` showing *usable* Accelerator APQN

To remove the binding issue, use a command like:

```
chzcrypt --se-unbind <aa.dddd>
```

However, during termination of an SE guest all bound crypto resources are reset and the binding is removed.

With the bind step the SE guest claims the crypto resource for exclusive use. *Use* here means to run crypto load: send crypto requests to and receive replies from the HSM. A bound APQN cannot get used by any other SE guest or even the KVM hypervisor. The hypervisor however must still be able to maintain the KVM guests and their assigned resources. So at any time the hypervisor is able to reset the crypto resource to retrieve it for whatever reason. The SE guest will immediately recognize this as all further requests (and pending replies) are rejected.

### EP11 coprocessor APQN

An APQN in EP11 mode needs two steps to become ready for crypto usage. Figure 5-7 shows the state diagram.

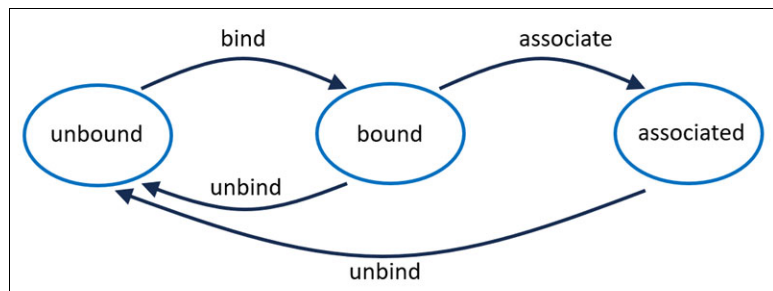


Figure 5-7 EP11 Coprocessor APQN SE states

The bind step is similar to an APQN in Accelerator mode. The SE guest claims the crypto resource `<aa.dddd>` for exclusive use with the `chzcrypt` command:

```
chzcrypt --se-bind <aa.dddd>
```

With a successful bind the SE guest has occupied the crypto resource. The APQN can be used to query any information from the HSM for example the MKVP of the Master Key. Administrative crypto commands like setting up a new Master Key and all crypto load **not** involving any key material is allowed, for example fetching Random Data from the HSM.

Crypto commands containing any secure key material like creation of a secure key, use of a secure key to en- or decrypt data and so on will be rejected. The UV is supervising all the crypto load and will filter out and refuse such requests and replies.

For a full usage the APQN needs to be *associated* with an *AP association secret*. The secret is used by the UV to open an EP11 session to this HSM with the secret value used to derive the EP11 session login authentication data.

Before the association can be done, the *AP association secret(s)* need to be loaded into the Ultravisor. The command-line tool `pvsecret` already known from section 5.4.2, “Secret preparation for SE guests with CEX support” on page 119 does the job:

```
pvsecret add <encrypted-secret-file>
```

With `pvsecret list` the list of secrets known to the UV can be displayed, see the example in Figure 5-8 on page 123. `pvsecret list` shows the secret index - 0 in this example - and the secret id. The secret index is simple the sequence number starting with 0 where the UV stored the secret’s data, and needs to be remembered for the following step. And of course the secret id listed here should match to the secret id at the time of creation as it was listed in 5.4.2, “Secret preparation for SE guests with CEX support” on page 119.

```

File Edit View Bookmarks Plugins Settings Help
root@sequest:~# chzcrypt --se-bind 28.0014
root@sequest:~# pvsecret add secret1.bin
Successfully added the secret
root@sequest:~# pvsecret list
Total number of secrets: 1

0 Association:
  03153249db7ce46b0330ffb1a760b59710531af08ec4d7f8424a6870fae49360
root@sequest:~# chzcrypt --se-associate 0 28.0014
root@sequest:~# lszcrypt -V
CARD.DOM TYPE MODE STATUS REQUESTS 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
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 usable
root@sequest:~# █

```

Figure 5-8 binding and association of an EP11 APQN

When the AP association secret(s) have been injected into the UV for this SE guest, each EP11 crypto resource needs to get associated with exactly one secret:

```
chzcrypt --se-associate <secret index> <aa.dddd>
```

Under the hood the UV will establish an EP11 session on this HSM. It will use the secret value from the secret with the given index to derive the EP11 login credentials. This procedure may take some time (some milliseconds up to some seconds) and **chzcrypt** will wait up to 30 seconds for completion. Figure 5-8 shows the **lszcrypt** output after a successful association with the SESTATE of the APQN updated to *usable*.

After successful association all the key material related to this APQN is implicitly session-bound. For example a new generated secure key is wrapped by the Master Key at this HSM and session bound to an EP11 session with login credentials derived from the associated secret value. To reuse such a secure key one needs to a) instantiate an SE guest with b) access to an HSM with same Master Key setting and c) associate the APQN for this HSM with the same secret value.

Note that the UV has limited space for handling CEX support for SE guests. The UV on z16 is able to track the bind and association state of up to 12 APQNs per SE guest.

### CCA coprocessor APQN

APQNs in CCA Coprocessor mode are currently not supported within KVM SE guests with AP pass-through. Note that the KVM hypervisor does **not** prevent one to include such APQNs to be forwarded into KVM SE guests. However, the Linux device driver detects this and as a result **lszcrypt** marks such an APQN as *illicit*:

```

File Edit View Bookmarks Plugins Settings Help
root@sequest:~# lszcrypt -V
CARD.DOM TYPE MODE STATUS REQUESTS PENDING HWTYPE QDEPTH FUNCTIONS DRIVER SESTAT
-----
08 CEX8C CCA-Coproc online 1 1 14 08 S--D--NHF- cex4card -
08.0014 CEX8C CCA-Coproc online 1 1 14 08 S--D--NHF- cex4queue illicit
root@sequest:~# █

```

Figure 5-9 illicit state

## 5.4.5 Security Details

Securing the access and usage of the crypto resources is based on two pillars:

- ▶ HSM access with the correct Master Key setup  
All secure keys are wrapped by the MK of the HSM used. This denotes the “something you have” part of a two-factor authentication - *access* to the right resource. Any HSM with the correct (equivalent) MK setup will be sufficient.
- ▶ AP association secrets  
All secure keys used with SE guests with AP pass-through support are session session-bound keys where the session login credentials are derived from the secret value. This can be seen as the “something you know” part of a two-factor authentication, to know some secret.

Fraud use of secure key material requires to fulfill both criteria besides the necessity to catch keys and maybe useful data.

The Secure Execution environment in combination with the Ultravisor firmware ensures the cloud runtime environment is not able to access any memory within the SE guest. However, the hypervisor is still able - and needs this for its job: to maintain the resources of the KVM guests. So the hypervisor can run *denial of service* attacks, for example reset and thus re-claim APQNs assigned to the SE guest. However, the UV firmware guarantees to handle these situations gracefully without the possibility to leak any information like leftover requests within the crypto resources.

## 5.4.6 Redundancy

In the server realm there is often a requirement for High Availability (HA) for the provided services. For example, an application using the pkcs#11 library OpenCryptoki (see the [OpenCryptoki](#) repository on GitHub) may set up two APQNs to gain reliability.

However, using more than one APQN within SE environment requires some considerations:

- ▶ The physical crypto express card plugged into the machine is logically divided into domains. Each domain represents an individual HSM with its own Master Key setup.  
So protection against broken hardware means providing another APQN on another crypto card.
- ▶ When using multiple APQNs for fallback reasons the Master Key setup needs to be the same. Moreover, each HSM is customizable with additional options (often referred to as Access Control Points). These settings may disagree between two HSMs resulting in strange and unexpected behavior during runtime.  
An application should at least check the MKVP of each APQN before use.
- ▶ All usable secure keys within an SE guest with AP pass-through support are session-bound. The session is built up and torn down by the UV based on credentials derived from the secret value associated with the APQN. So a setup of redundant APQNs must be associated with the same secret value.

## 5.4.7 Protecting AP association secrets

As list item “AP Association Secret” on page 118 shows, the generated secrets file is encrypted with the public host key and only the private host key living in the UV on the target machine is able to decrypt it.



However, the secret can be stolen, for example during transfer to the SE guest. With access to the hypervisor, it may then be used to associate the same APQN with another hand-crafted SE guest.

The solution here is to make sure a generated AP association secret is only usable with the related SE guest image. This is done by introducing yet another key, the *customer communication key* (CCK). The CCK key is a symmetric AES 256-bit key generated by the customer, similar to the following:

```
dd if=/dev/random of=<cck_filename> bs=1 count=32
```

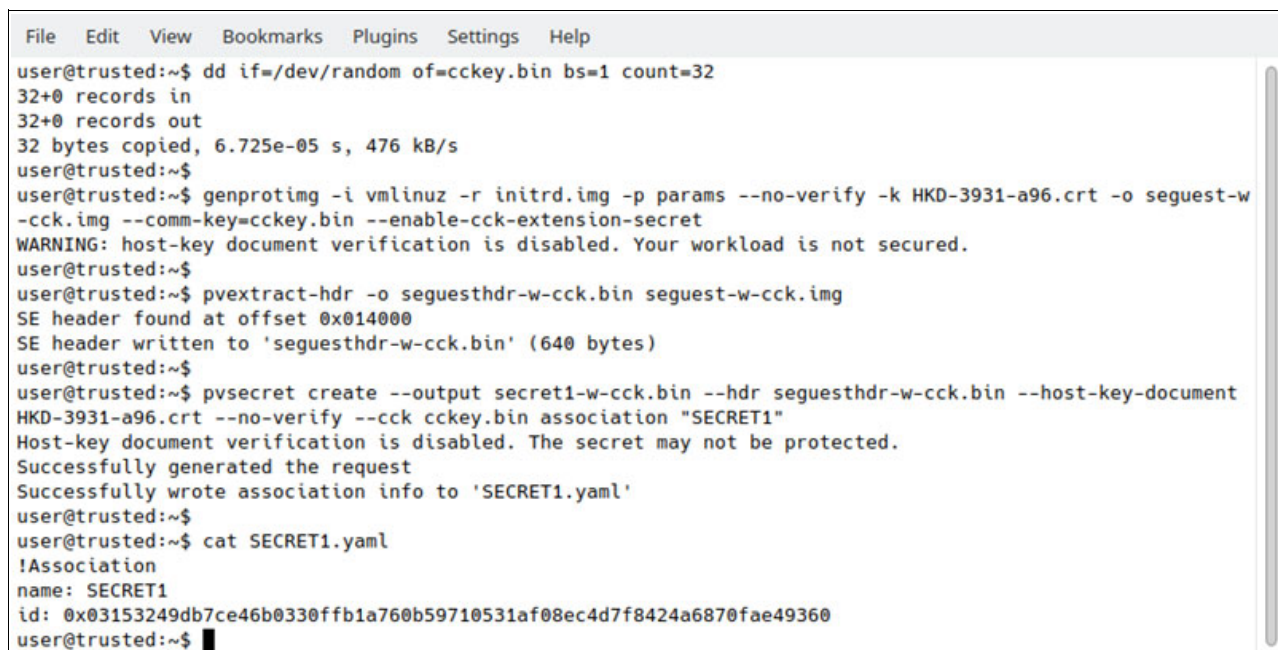
This key is then included in the encrypted SE guest image during the image build process. Additionally the SE guest image enables an option to accept only secrets encrypted with the CCK key, similar to the following:

```
genprotimg [...] --comm-key=<cck_filename> --enable-cck-extension-secret
```

And with the creation of an AP association secret, the option `--cck` tells `pvsecret` to use the customer communication key to additionally encrypt the created secret by using the following command:

```
pvsecret create [...] --cck <cck_filename> association [...]
```

Figure 5-10 shows a session on our trusted host which runs all the steps from CCK key creation to the generation of an CCK key protected secret.



```
File Edit View Bookmarks Plugins Settings Help
user@trusted:~$ dd if=/dev/random of=cckkey.bin bs=1 count=32
32+0 records in
32+0 records out
32 bytes copied, 6.725e-05 s, 476 kB/s
user@trusted:~$
user@trusted:~$ genprotimg -i vmlinuz -r initrd.img -p params --no-verify -k HKD-3931-a96.crt -o sequest-w-cck.img --comm-key=cckkey.bin --enable-cck-extension-secret
WARNING: host-key document verification is disabled. Your workload is not secured.
user@trusted:~$
user@trusted:~$ pvextract-hdr -o sequesthdr-w-cck.bin sequest-w-cck.img
SE header found at offset 0x014000
SE header written to 'sequesthdr-w-cck.bin' (640 bytes)
user@trusted:~$
user@trusted:~$ pvsecret create --output secret1-w-cck.bin --hdr sequesthdr-w-cck.bin --host-key-document HKD-3931-a96.crt --no-verify --cck cckkey.bin association "SECRET1"
Host-key document verification is disabled. The secret may not be protected.
Successfully generated the request
Successfully wrote association info to 'SECRET1.yaml'
user@trusted:~$
user@trusted:~$ cat SECRET1.yaml
!Association
name: SECRET1
id: 0x03153249db7ce46b0330ffb1a760b59710531af08ec4d7f8424a6870fae49360
user@trusted:~$
```

Figure 5-10 Generate and use a CCK to protect a secret

On the receiving side - the SE guest - there is no difference in handling such a protected AP association secret. The running SE guest knows the CCK key as it was injected into the image and CCK secret checking is enabled. So this time the `pvsecret` command tries to add a secret into the UV and the CCK checking is enforced. With a matching secret, the `pvsecret add` command behaves as before. However, the attempt to use a not CCK protected secret would result in failure of the `pvsecret add` command.

The terminal session on the SE guest in Figure 5-11 on page 126 shows a bind and associate example with the generated CCK protected secret.

```

File Edit View Bookmarks Plugins Settings Help
root@sequest:~# chzcrypt --se-bind 28.0014
root@sequest:~#
root@sequest:~# pvsecret add secret1-w-cck.bin
Successfully added the secret
root@sequest:~#
root@sequest:~# pvsecret list
Total number of secrets: 1

0 Association:
  03153249db7ce46b0330ffb1a760b59710531af08ec4d7f8424a6870fae49360
root@sequest:~#
root@sequest:~# chzcrypt --se-associate 0 28.0014
root@sequest:~#
root@sequest:~# lszcrypt -V
CARD.DOM TYPE MODE STATUS REQUESTS 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
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 usable
root@sequest:~# █

```

Figure 5-11 Using a secret protected by a CCK

## 5.4.8 Important considerations for the secure use of Crypto Express adapters in EP11 mode

In this section, we outline some very important requirements and restrictions for the secure use of Crypto Express adapters in EP11 mode.

### Requirement 1

You must trust the TKE domain admins of all EP11 APQNs that you associate with your SE guest.

In particular, you must trust the TKE domain administrators to tell the SE guest administrators the following:

- ▶ Adapters (SNs) and domains that are configured for you (possibly communicating the certificates installed in the domains).
- ▶ HSM master (wrapping) keys that are installed in the adapter domains by communicating their wrapping key verification patterns.
- ▶ Whenever a master key is changed.
- ▶ Whenever an adapter domain is zeroized.

Further, you must trust that the TKE domain administrators to never configure EP11 HSM master keys used to (over time) protect the same operational key to two different domains of the same adapter. In particular, a TKE domain administrators must never configure the same EP11 HSM master key to two different domains of the same adapter.

### Requirement 2

Your TKE adapter administrators should be trustworthy.

Trustworthy TKE adapter admins must inform your SE guest admin whenever an adapter used by your SE guest is zeroized. If your TKE adapter admins are trustworthy then associating only APQNs whose domain admins are trustworthy is secure.

If your TKE adapter admin is not trustworthy you must check for every secure key generated in the secure guest whether its HSM master key verification pattern is the one communicated by the trusted TKE domain admin. The openCryptoki EP11 token can be configured to only generate secure keys with an expected wrapping key verification pattern, and zkey allows you to inspect the master key verification patterns of keys in the zkey repository.

### **Restriction**

Never use the same association secret with two APQNs of the same adapter. This restriction is to enforce by current IBM and LinuxONE firmware to avoid unexpected side effects of resetting an EP11 AP queue.



# Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this book.

## IBM Redbooks

The following IBM Redbooks publications provide additional information about the topic in this document. Note that some publications referenced in this list might be available in softcopy only.

- ▶ *Securing Your Critical Workloads with IBM Hyper Protect Services*, SG24-8469

You can search for, view, download or order these documents and other Redbooks, Redpapers, Web Docs, draft and additional materials, at the following website:

[ibm.com/redbooks](https://ibm.com/redbooks)

## Other publications

These publications are also relevant as further information sources:

- ▶ [Cryptographic domains](#)
- ▶ *The Virtualization Cookbook for IBM Z Volume 1: IBM z/VM 7.2*, SG24-8147-02
- ▶ [How to Virtualize IBM zSystems Hardware Cryptography with z/VM](#)

## Online resources

These websites are also relevant as further information sources:

- ▶ Cryptographic domains  
<https://www.ibm.com/docs/en/linux-on-systems?topic=wysk-crypto-domains/>
- ▶ Getting started with Podman  
<https://podman.io/docs>
- ▶ Common features of openCryptoki  
<https://www.ibm.com/docs/en/linux-on-systems?topic=322-opencryptoki-features>
- ▶ Protected and secure volume encryption  
<https://www.ibm.com/docs/en/linux-on-systems?topic=2020-protected-secure-volume-encryption>

## Help from IBM

IBM Support and downloads

[ibm.com/support](https://ibm.com/support)

IBM Global Services

[ibm.com/services](https://ibm.com/services)



# Crypto Express for Cloud Workloads

SG24-8547-00

ISBN 0738461660

(1.5" spine)  
1.5" <-> 1.998"  
789 <-> 1051 pages



# Crypto Express for Cloud Workloads

SG24-8547-00

ISBN 0738461660

(1.0" spine)  
0.875" <-> 1.498"  
460 <-> 788 pages



**Redbooks**

## Crypto Express for Cloud Workloads

SG24-8547-00

ISBN 0738461660

(0.5" spine)  
0.475" <-> 0.873"  
250 <-> 459 pages



**Redbooks**

## Crypto Express for Cloud Workloads

(0.2" spine)

0.17" <-> 0.473"

90 <-> 249 pages

(0.1" spine)

0.1" <-> 0.169"

53 <-> 89 pages



# Crypto Express for Cloud Workloads

SG24-8547-00

ISBN 0738461660

(2.5" spine)  
2.5" <-> nnn.n"  
1315 <-> nnnn pages



# Crypto Express for Cloud Workloads

SG24-8547-00

ISBN 0738461660

(2.0" spine)  
2.0" <-> 2.498"  
1052 <-> 1314 pages









SG24-8547-00

ISBN 0738461660

Printed in U.S.A.

Get connected

