Redbooks
ibm.com/redbooks

Getting Started with Linux on Z Encryption for Data At-Rest

Bill White

Megan Hampton

Benedikt Klotz

Pat Oughton

Guilherme Nogueira

Carlos Henrique Reimer

Security

IBM Z

IBM

Redbooks

IBM Redbooks

Getting Started with Linux on Z Encryption for Data At-Rest

April 2019

SG24-8436-00

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

First Edition (April 2019)

This edition applies to the required and optional hardware and software components needed for Linux on IBM Z encryption for data at-rest.

Contents

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 http://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.

ECKD™	Parallel Sysplex®	z Systems®
IBM®	QRadar®	z/OS®
IBM Z®	RACF®	z/VM®
IBM z Systems®	Redbooks®	z13®
IBM z13®	Redbooks (logo) ®	z13s®
IBM z13s®	System z®	zEnterprise®
IBM z14™	XIV®	zSecure™

The following terms are trademarks of other companies:

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

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

UNIX is a registered trademark of The Open Group in the United States and other countries.

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

Preface

This IBM® Redbooks® publication provides a general explanation of data protection through encryption and IBM Z® pervasive encryption with a focus on Linux on IBM Z encryption for data at-rest. It also describes how the various hardware and software components interact in a Linux on Z encryption environment for *data at-rest*.

In addition, this book concentrates on the planning and preparing of the environment. It offers implementation, configuration, and operational examples that can be used in Linux on Z volume encryption environments.

This publication is intended for IT architects, system administrators, and security administrators who plan for, deploy, and manage security on the Z platform. The reader is expected to have a basic understanding of IBM Z security concepts.

Authors

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

Bill White is an IBM Redbooks Project Leader and Senior IT Infrastructure Specialist at IBM Redbooks, Poughkeepsie Center.

Megan Hampton is a Staff Software Engineer for IBM Z in Poughkeepsie, NY. Megan began her career as a tester on IBM Wave for z/VM®, and later learned development along with Python, Java, and JavaScript coding skills. She then began working with customers and service support, learning why customers liked a product, what could be changed, and whether their expectations were being met. Most recently, Megan moved to Secure Engineering, where she hacks and reverse engineers some of the software products with which she works.

Benedikt Klotz is an IBM Z software and firmware professional in Boeblingen, Germany. He is developing software for the IBM Z Crypto Express Card and for Linux on Z. In his role, he developed a deep understanding of cryptographic topics and is involved in maintaining the IBM CryptoCards website.

Pat Oughton is an IBM Z Client Technical Specialist in New Zealand. He joined IBM in 2015 after working as a z/OS® Systems Programmer for 30 years. His areas of expertise include IBM Z installation (hardware and operating system) and IBM Parallel Sysplex® Implementation. He has written two other IBM Redbooks publications.

Guilherme Nogueira is an IT Specialist for IBM Global Technology and Services in Brazil. He has six years of experience in configuring Linux on IBM System z®. He supported over 1,800 Linux on Z servers as part of his previous role in Linux on Z support for IBM internal account. He is currently working in Auto Server Provisioning, supporting Linux on Z deployments for IBM internal accounts. He holds a degree in Information Security and Computer Networking from Faculdade de Tecnologia de Americana (FATEC Americana). His areas of expertise include Linux, systems automation, security, and cryptography.

Carlos Henrique Reimer is an IT Specialist for IBM Global Technology and Services in Brazil. He has more than 30 years of experience as a z/VM system programmer. He is currently working as an SME for the z/VM Base team in Brazil that supports more than 120 System z servers for an IBM internal account. Holding a post-graduate degree in Web Application Development from Universidade Uniasselvi, his areas of expertise include high availability by using SSI clusters, security, and server management.

A special thanks to the following people for their contributions to this project:

Bob Haimowitz
IBM Redbooks Organization

Tom Ambrosio and William Lamastro
IBM Competitive Project Office

Reinhard Buendgen
Linux on IBM Z Security Product Owner

Brian W. Hugenbruch
IBM Z Security Development for Virtualization & Cloud

Danijel Soldo
IBM Z Performance Analyst

Thanks to the authors of the *Getting Started with z/OS Data Set Encryption*, SG24-8410, for the groundwork that helped shape this IBM Redbooks publication:

Andy Coulson
Jacky Doll
Brad Habbershaw
Cecilia Carranza Lewis
Thomas Liu
Ryan McCarry
Eysha Shirrine Powers
Philippe Richard
Romoaldo Santos

Thanks to those who provided input to this IBM Redbooks publication:

Hendrik Brueckner
Ingo Franzki
Harald Freudenberger
Stephan Hartig
Richard John Moore
Christian Rund
Markus Selve
Frank Uhlig
Klaus Werner
Hans Georg-Zipperer

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

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

- Send your comments in an email to:

  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 Facebook:

  http://www.facebook.com/IBMRedbooks

- Follow us on Twitter:

  http://twitter.com/ibmredbooks

- Look for us on LinkedIn:

  http://www.linkedin.com/groups?home=&gid=2130806

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

  https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm

- Stay current on recent Redbooks publications with RSS Feeds:

  http://www.redbooks.ibm.com/rss.html

1

Protecting sensitive data

Over the course of several years, the term *data breach*, which was once an anomaly, became a normality among the IT industry and its consumers alike.

Statistically, cybersecurity threats are on an all-time high, which leads top industries and organizations to ensure that their customer's data is protected. Nothing that is exposed or in the clear is ever considered safe. It is the obligation of each organization to have a security plan in place. It is also imperative to consider the potential threats in advance and not when impending danger is imminent.

Typically, data protection is regulated as a requirement. However, to ensure the safety of critical and sensitive data, going beyond the minimum threshold is a must. Now is the time to take action and make a difference. The question remains: What are you doing to protect your data?

In this book, we provide guidance, advice, and suggestions about how you can best protect your *data at-rest*[1] in a Linux on Z environment by using encryption.

This chapter includes the following topics:

[1] Data at-rest includes files that are written to storage devices, such as disk and tape. Data at-rest can persist, even when the associated application is no longer running. When an application is restarted, it can retrieve the data at-rest because it is stored on disk or tape.

1.1 Why protect your data beyond compliance

The implication is known that data is one of the most valuable assets of any company and a full-proof security strategy to protect that data is of vital importance.

Also, many organizations are recognizing that protecting only the data that is required to achieve compliance is the bare minimum. It is not uncommon to encounter regulations that were written some time ago that do not implement today's security best practices.

Organizations also realize that a move from selective encryption[2] (protecting specific types of data only) to pervasive encryption (encrypting all data) is needed. Likewise, many barriers that are encountered today with current enterprise data protection policy and strategy can be removed with pervasive encryption, such as the following examples:

- Decoupling encryption from data classification

  This process allows organizations to implement their encryption strategy independent of any challenges they might face while identifying and classifying sensitive data. It also reduces the risk of unidentified or mis-classified data.

- Using encryption without interrupting business applications or affecting service level agreements (SLAs)

  Changes to the application are not required if data is encrypted after it leaves the application and decrypted before it reaches the application.

- Reducing high costs that are associated with processor overhead

  The cost of encryption is minimized by encrypting data in bulk and by using hardware encryption accelerators with high performance and low latency.

In addition, most organizations experience numerous audits per year. Increasing rules from inside and outside of an organization are causing significant security concerns, especially in the short term. Enterprises need security solutions that ensure maximum visibility into activities in their entire infrastructure, along with automated threat analysis and remediation.

The IBM Z platform provides solutions for security teams and auditors to verify up-to-date compliance statistics in near real time. Auditors can also use enhanced tooling to significantly reduce the time and effort that is required to validate compliance requirements and complete audits.

For these reasons, the underlying concept of pervasive encryption suggests encrypting all data rather than encrypting data for compliance only.

[2] Encryption is a technology that is well-versed in the art of hiding sensitive information in plain sight. Encryption operations require a cryptographic key and a cryptographic algorithm. Together, a cryptographic key and algorithm can encrypt and decrypt data.

1.2 IBM Z pervasive encryption

Extensive use of encryption is one of the most effective ways to help reduce these potential risks and financial losses that are caused by a data breach. Encrypting data, whether it be *data in-flight*[3], *in-use*[4], or at-rest, also helps to meet the needs of complex compliance mandates and security best practices.

IBM Z pervasive encryption reduces opportunities for compromise and is enabled through tight platform integration. Pervasive encryption spans the entire Z stack in hardware, software, operating systems, middleware, and even tooling.

The components of the IBM Z platform that play a key role in providing pervasive encryption are shown in Figure 1-1.

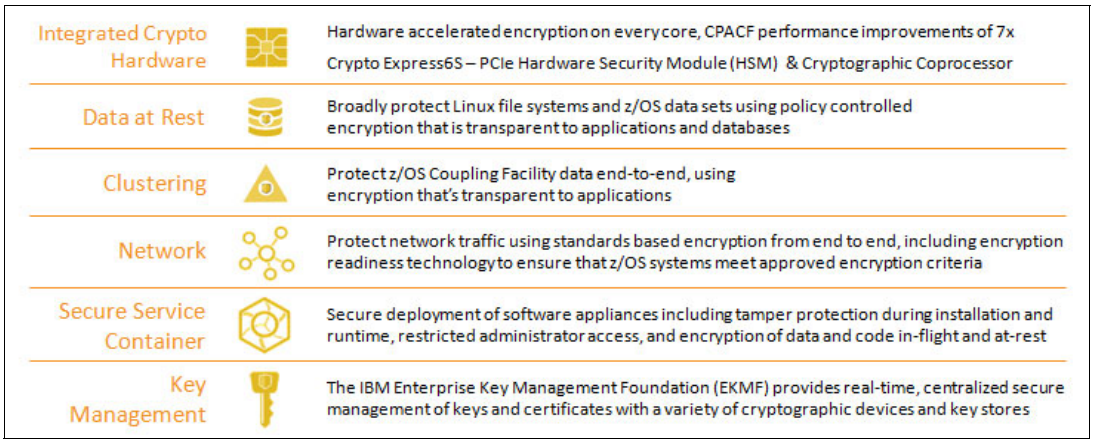

Figure 1-1 IBM Z pervasive encryption components

1.2.1 Linux on Z data at-rest encryption

Pervasive encryption for Linux on Z is a combination of granularity versus complexity and overhead. The bigger the switch, the less freedom that is available to pick and choose different encryption algorithms, keys, and so on.

Similarly, the bigger the switch, the less overhead that is available for analyzing, classifying, encrypting, and maintaining the encryption environment. Ultimately, you gain broader scope of encryption at the cost of granularity the further down the software or hardware stack you go.

Approaches to encryption of data at-rest can be categorized into the following levels:

- SAN Link and Full Disk Encryption
- Virtualization Layer or Host Level Encryption
- File or Dataset Level Encryption (encryption of data at-rest through volume encryption is the focus of this Linux on Z publication.)

[3] Data in-flight can include sensitive data (such as passwords or credit card information) that is sent over a network to a server; for example, to make an online purchase. Data in-flight also includes data that is sent over the storage area network (SAN) from a host system to disk and tape devices. Data in-flight can be stored persistently or it might be in-use in the system temporarily to complete a transaction or operation

[4] Data in-use includes records that can be in memory before it is written to storage devices. Data in-use is not persistent. However, it can be readable in system memory dumps.

- Database Encryption
- Application Encryption

The five pervasive encryption levels for data at-rest are shown in Figure 1-2. Although you have more control over the encryption settings, the higher up you go in the levels of encryption, the more complexity and cost is incurred.

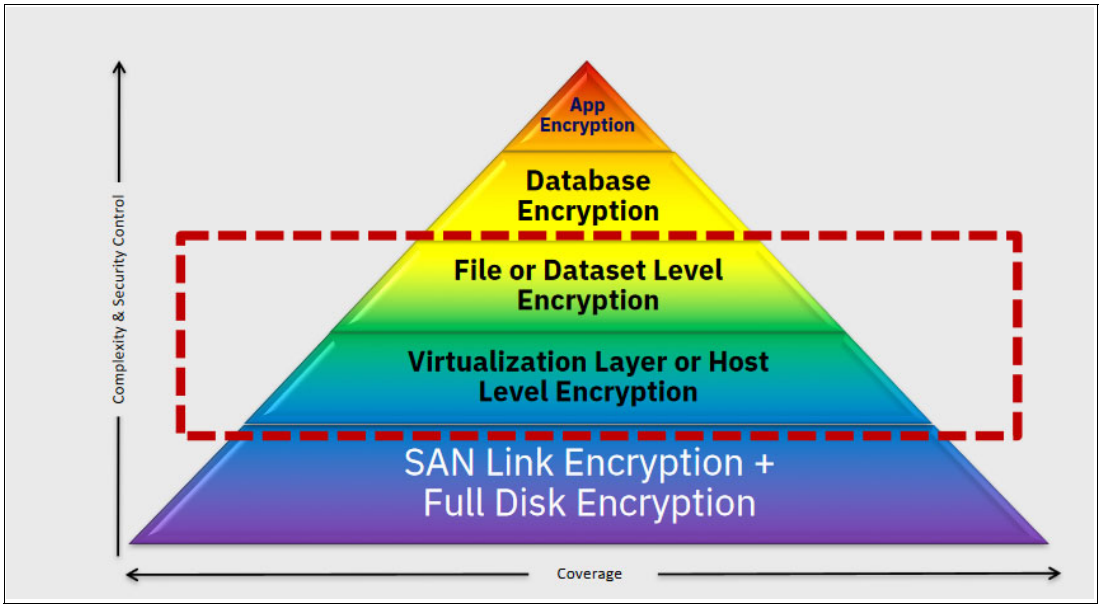

Figure 1-2 Pervasive encryption for data at-rest

SAN link and full disk encryption

Full disk encryption with physical SAN security addresses all major storage administrators' security concerns. This type of encryption allows for minimal disruption of SAN infrastructure deployments and maintains interoperability. It also protects against intrusion, tampering, or removal of physical infrastructure with no application overhead.

This level is considered "all or nothing" encryption and encrypts only data at-rest and at the storage controller subsystem level by using a single key for all encryption. Self-encryption storage (such as IBM DS8800, IBM XIV® Storage Systems, tape, and virtual tape) solves the following security problems:

- Secure disposal of storage at the end of its lifecycle
- Tapes that are lost during shipment
- Data protection after return for repair or in case of theft

Virtualization layer or host level encryption

Virtualization is defined as a mechanism through which the real system memory and resources can be shared among multiple competing execution contexts. However, the virtualization layer is another abstraction layer between network and storage hardware, computing, and the application that is running on it.

All z/VM and Linux on Z encryption falls into virtualization. A hosted architecture means the virtualization layer runs on the operating system. In this case, the operating system works as the host and manages the physical resources and the device support. The operating systems on the virtual machines are the guest environment.

File or data set level encryption

File (also referred to as volume encryption) or data set level encryption provides broad coverage for sensitive data in-flight and at-rest data. It is not apparent to applications and allows for separation of duties within your organization. This broad protection is managed by operating system components and subsystems.

The use of extra compliance controls might not be needed because the data remains encrypted when it is written. Essentially, file (and volume) or data set encryption provides a more flexibility.

Database encryption

Database encryption protects data in-flight, in-use, and at-rest. It is not transparent to the application, but allows for separation of duties and granular access control. Database encryption on IBM Z safeguards the encrypted sensitive data in logs, image copy files, and volume backups, while taking advantage of integrated cryptographic hardware acceleration.

Business policies per database can vary, but typically, implementing a policy-based process at the database encryption level requires more effort compared to a broader scope of encryption, such as at file, volume, or data set level encryption.

Application encryption

Application encryption provides data protection that is managed by the application. It requires changes to applications to implement and maintain encryption and is highly granular in protecting data, right up to the point where it is used by the application.

Applications are responsible for their own key management. This type of encryption is used when other levels of encryption are not available or suitable.

Application-based encryption involves overhead at all stages (analysis, deployment, and maintenance), especially when applications are modified to meet new business needs. Therefore, application programmers must know exactly which data must be encrypted. It is easier to encrypt all data seamlessly at the point that it is written, without having to rely on the programmers to determine exactly what data must be encrypted.

1.2.2 Separation of duties

Although security best practices suggest the duties of administrators be separated to reduce the risk of cybersecurity threats, they are often integrated because of business practices. When laying out a security strategy for encryption of data at-rest, consideration of the different administrator roles and responsibilities is paramount and might require the realignment of your organization's security policies. Separate duties to limit who can access and manage the encryption ecosystem should be assessed as well.

For more information, see 3.4, "Considerations for separation of duties" on page 25.

1.2.3 Managing the pervasive encryption environment

Managing the pervasive encryption environment is supported by several IBM Security solutions. The essential capabilities that are needed are shown in Figure 1-3.

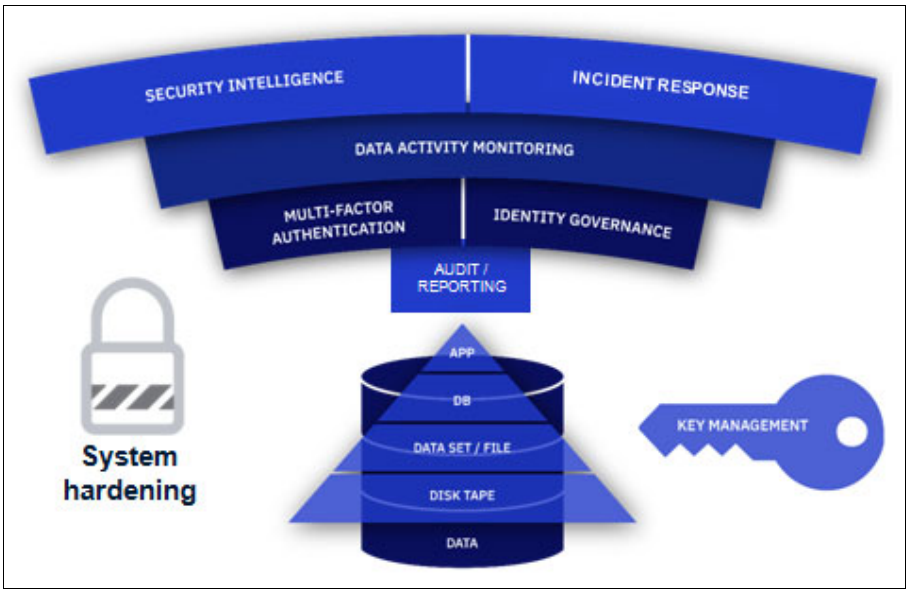

Figure 1-3 IBM Security solutions with essentials capabilities

For more information about the IBM Z security solutions and their capabilities, see Chapter 6, “Auditing and monitoring the data at-rest environment” on page 71, and 2.2.3, “Trusted Key Entry workstation” on page 17.

1.3 Understanding Linux on Z data at-rest encryption

Linux on Z data at-rest encryption provides the following design benefits:

- Offers a higher level of protection along with the high throughput of encryption by using a crypto-coprocessor. Therefore, sensitive key material is not visible in clear form at any time.
- Protects data in a way that is aligned with your current security access control mechanisms, which offers a more straightforward configuration experience.
- Performs efficiently at speed by use of the integrated Z crypto-hardware and software stack.
- Enables encryption without requiring application or database changes.
- Provides cryptographic separation from other environments. Encryption keys can be configured so that they are owned and managed by a local organizational environment (for example, production versus test).

All of these benefits for Linux on Z at-rest encryption rely solely on an efficient key management strategy and solid cryptographic system. The key management strategy is essential to governing and safeguarding the encryption keys that protect your data. You must ensure that the encryption keys are available whenever and wherever encrypted data is used. As such, it is important to understand how Linux on Z data at-rest encryption works with the hardware and software components in the IBM Z cryptographic system.

1.3.1 IBM Z cryptographic system

Linux on Z data at-rest encryption uses the integrated cryptographic system that is available on the IBM Z platform. In a cryptographic system, a cryptographic key and a cryptographic algorithm are required.

The encryption algorithms are public; the encryption keys are secret. The secure management of keys (or key material) is vital to the protection of data in a cryptographic system.

Linux on Z data at-rest encryption uses symmetric encryption, which means, the same cryptographic key is used for encrypting and decrypting data. It can be used to encrypt large amounts of data by using symmetric stream ciphers or by breaking the data into blocks and encrypting each block. Common symmetric encryption algorithms include the Advanced Encryption Standard (AES) and Data Encryption Standard (DES). Linux on Z data at-rest encryption uses the AES algorithm.

Cryptographic key types

Linux on Z data at-rest encryption uses the types of encryption keys that are listed in Table 1-1.

Table 1-1 Encryption key types

Key type	Description
Data	A data-encrypting key that is used to encrypt and decrypt data.
Key-encrypting	A key that encrypts or wraps other keys.
Effective	A type of data-encrypting key; also called a *data key* that is wrapped by a key-encrypting key (KEK).
Master	A special KEK that is in a tamper-responding, Crypto Express adapter only and sits at the top of a KEK hierarchy. Loading and managing the master key can be done by using the Trusted Key Entry (TKE) workstation.
CPACF wrapping	A special key-encrypting key that is generated at LPAR activation and is in the Hardware System Area, which is inaccessible to applications and the operating system. It is used to create protected keys.
Secure	A data-encrypting key that is encrypted by a master key and never appears in clear text that is outside of a secure environment, such as a tamper-responding HSM or IBM Z firmware.
Clear or plain	A data-encrypting key that is not encrypted by any other key. The key material is in plain text.
Protected	A data-encrypting key that is encrypted by a CPACF wrapping key and used within IBM Z platform. This key cannot be shared with another LPAR because the wrapping key is unique. In the case of Linux in a native LPAR, the wrapping key is specific to the LPAR. However, for guests of z/VM or KVM, the wrapping key is specific to the guest.
Operational	A key that is not a master key or KEK, such as a data-encrypting key (which can be clear, secure, or protected).

Key lifecycle management

Key lifecycle management is a critical aspect in any encryption strategy. Cryptographic keys feature a lifecycle that includes tasks, such as key creation, key activation, key deactivation, key archival, and key deletion. Some regulations, such as European Union (EU) General Data Protection Regulation (GDPR), Payment Card Industry Data Security Standard (PCI-DSS), and Health Insurance Portability and Accountability Act (HIPAA), require that key management processes are created and well-documented.

For more information about key management, see 3.6, “Key management considerations” on page 29.

Cryptographic hardware on IBM Z

The IBM Z platform offers cryptographic engines that provide high-speed cryptographic operations. The following cryptographic engines are used with Linux on Z data at-rest encryption:

- Central Processor Assist for Cryptographic Function (CPACF)

  A high-performance, low-latency coprocessor that performs symmetric encryption and calculates message digests (hashes) in hardware. The AES, DES/TDES, SHA-1, SHA-2, and SHA-3 algorithms are supported.

  The cryptographic function is provided through a set of instructions that are available in hardware on every processor unit (CP and IFL). With the z14, a high-quality true random number generator is available.

- Crypto Express adapters

  A tamper-sensing and tamper-responding adapter that provides acceleration for high-performance cryptographic operations. This specialized hardware performs AES, DES/TDES, RSA, Elliptic Curve (ECC), SHA-1, and SHA-2, and other cryptographic operations. It also supports specialized high-level cryptographic APIs and functions. Crypto Express adapters are designed to meet the FIPS 140-2 Level 4 and PCI HSM security requirements for hardware security modules[5] (HSMs).

  For more information about cryptographic hardware, see the publications that are available at this web page.

Linux on Z cryptographic components

The following components for Linux on Z data at-rest encryption are in the Linux *kernel*:

- `dm-crypt`: A device mapper for a crypto-target (such as a storage device). It provides transparent encryption of volumes or block devices.
- `paes`: A module that performs protected key encryption-decryption and implements `paes` cipher. It also provides the cryptographic AES algorithm that is used by the infrastructure for protecting data at-rest (such as volume encryption).
- `pkey`: A module used for secure key and protected key management, including the following tasks:
  - Generates secure key
  - Transforms secure key into protected key

  `pkey` supports protected-key with AES as an encryption cipher for Linux on Z use with `dm-crypt`.

[5] An HSM is a physical computing device that safeguards and manages digital keys for strong authentication and provides cryptographic processing.

The following components for Linux on Z data at-rest encryption are in the Linux *user space*:

- `cryptsetup`: A utility that is used to create secure keys and to manage disk encryption. (It interfaces with `dm-crypt` and `zkey`.)

  By using **`cryptsetup`** commands, you can perform the following actions:

  - Open: Creates a mapping device
  - Close: Removes the mapping device
  - Status: Reports the mapping device status
  - Resize: Resize an active mapping device
  - Format: Format LUKS partitions

- `zkey`: A utility that manages secure keys. You can create secure keys with a length of 128, 192[6], or 256 bits.

  By using the **`zkey`** commands, you can perform the following tasks:

  - Generate secure keys
  - Validate secure keys
  - Reencipher secure key

For more information about examples of the **`zkey`** and **`cryptsetup`** commands and usage, see the following chapters of this publication:

- Chapter 5, “Deploying encrypted volumes for data at-rest” on page 57.
- Chapter 7, “Maintaining encrypted volumes for data at-rest” on page 79
- Chapter 8, “Performing key management for data at-rest encryption” on page 87

1.4 How Linux on Z data at-rest encryption works

When the `paes` cipher is used with Linux on Z data at-rest encryption, the following protected volume options are available:

- The LUKS2 format includes a header on the volume and a one time formatting is required. The LUKS2 header is made up of multiple key slots. Each key slot contains key and cipher information.

  The volume’s secure key is wrapped by a key-encrypting key (which is derived from a passphrase or a keyfile[7]) and stored in a keyslot. The user must supply the correct passphrase to unlock the keyslot. A keyfile allows for the automatic unlocking of the keyslot.

- The plain format does not include a header on the volume and no formatting of the volume is required. However, the key must be stored in a file in the filesystem. The key and cipher information must be supplied with every volume open.

Note: LUKS2 format is the preferred option for Linux on Z data at-rest encryption. For more information, see 3.9.3, “Volume format considerations” on page 43.

[6] Keys that are 192-bit AES keys are not supported by the XTS cipher mode. AES-256 (512-bit) and AES-128 (256-bit) encryption keys are supplied with XTS.

[7] The key-encrypting key is derived from a passphrase or a keyfile (a large random number), which is used to encrypt and decrypt a secure key.

1.4.1 Creating a secure key

The process that is used to create a secure key for an LUKS2 format volume is shown in Figure 1-4.

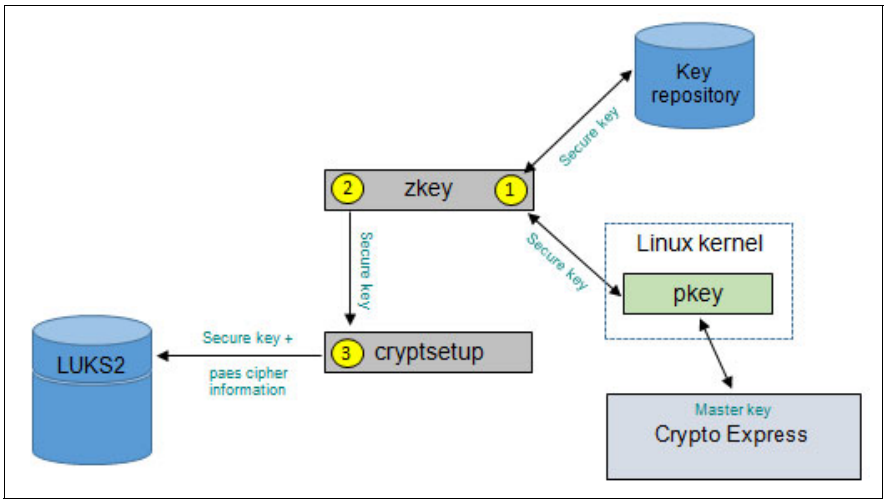

Figure 1-4 Creating a secure key

This process includes the following steps:

1. A secure key is created by using a **`zkey`** command. The `zkey` utility generates the secure key with the help of the `pkey` utility and an assigned Crypto Express adapter (with master key). The secure key is also stored in the key repository.
2. The use of the **`zkey cryptsetup`** command generates output strings that are copied and pasted to the **`cryptsetup`** command to create the encrypted volume with the appropriate secure key.
3. The `cryptsetup` utility formats the physical volume and writes the encrypted secure key and cipher information to the LUKS2 header of the volume.

1.4.2 Opening an LUKS2 formatted volume

The process that is used to open an LUKS2 formatted volume is shown in Figure 1-5.

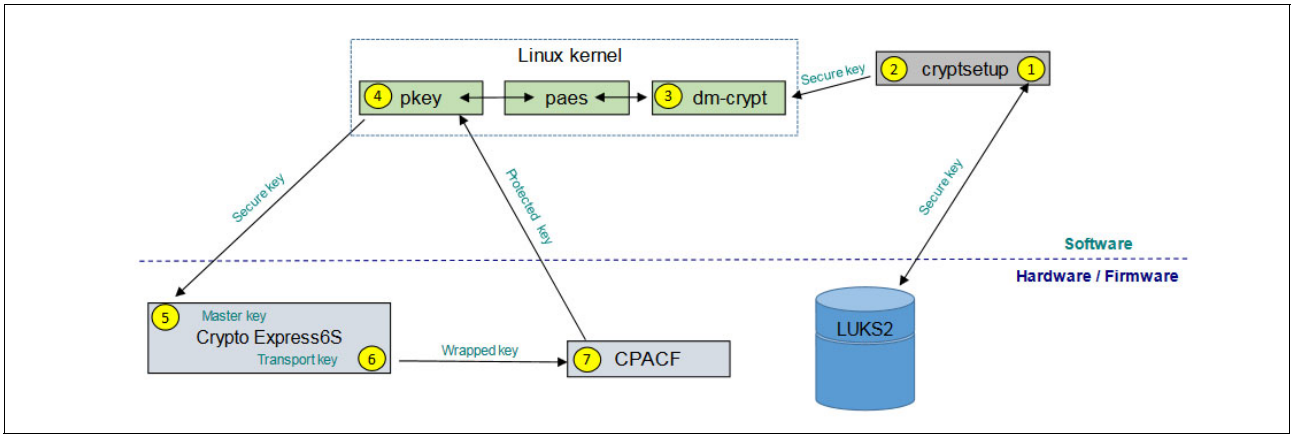

Figure 1-5 Opening an LUKS2 format volume

This process includes the following steps:

1. The `cryptsetup` utility fetches the secure key from the LUKS2 header.
2. The `cryptsetup` utility passes the secure key to `dm-crypt`.
3. The `dm-crypt` passes the secure key to `paes` for conversion into a protected key by using `pkey`.
4. The `pkey` module starts the process for converting the secure key to a protected key.
5. The secure key is unwrapped by the CCA coprocessor in the Crypto Express adapter by using the master key.
6. The unwrapped secure key (effective key) is rewrapped by using a transport key that is specific to the assigned domain ID.
7. By using firmware, CPACF creates a protected key and sends it to the `pkey` module for volume read/write usage with `paes` and `dm-crypt`.

For more information about how the key wrapping process works on the IBM z13® and z14 with Crypto Express adapters, see Appendix A, “Using protected keys for high-speed encryption” on page 99.

2

Identifying components and release levels

This chapter describes the IBM Z hardware and software components that are required or optional for Linux on Z data at-rest encryption. It includes the following topics:

2.1 Starting a Linux on Z data at-rest encryption implementation

Linux on Z encryption is enabled through tight integration that spans the capabilities of the IBM Z platform hardware, firmware, and software.

Implementations of encryption on earlier levels of the IBM Z platform can provide a base for establishing appropriate processes and procedures before implementing a full-scale production environment. However, cryptographic technology with the IBM z14 drastically reduced computational overhead compared to previous generations.

In addition, every new release of Linux on Z and the IBM device drivers continues to provide enhancements for various aspects of data security beyond encryption, such as identity management, access control, auditability, monitoring, and reporting.

The recommended and minimum supported levels of the IBM Z platform components that are needed to implement Linux on Z data at-rest encryption are listed in Table 2-1. The enhancements that are provided by the latest levels of IBM Z hardware, firmware, and software ensure the best scalability, performance, and enhanced system and security management capabilities.

Table 2-1 Supported levels of IBM Z hardware and z/VM for Linux on Z data at-rest encryption

Recommended hardware	z14 (CPACF) and Crypto Express6S	The CPACF in the z14 features up to seven times better performance compared to the IBM z13. The Crypto Express6S is twice as fast as the Crypto Express5S. **Note:** A z14 is a requirement for z/VM encrypted paging support.
Minimum hardware	z196 (CPACF) and Crypto Express3	Test environments only. If running z/VM 6.4; otherwise, z/VM 7.1 required zBC12, zEC12 or later.
Recommended z/VM release	z/VM V7.1 Base	Latest release.
Minimum z/VM release	z/VM V6.4 with APAR VM65993	APAR adds encrypted page support.

For more information about how to create an implementation plan for Linux on Z data at-rest encryption, see 3.1, “Creating an implementation plan” on page 22.

2.1.1 IBM Z platform: Optimized for Linux on Z data at-rest encryption

Considerable performance improvements were made with the cryptographic functions in the z14 and Crypto Express6S feature compared to earlier generations.

Regarding operating system levels for encryption, the most significant difference between levels of z/VM and the Linux on Z kernel is in the toleration capability. Full support to create and access encrypted data (read, write, and so on) is typically offered in the latest releases available. However, older versions with relevant fixes and Linux on Z device drivers might be able to access encrypted file (read, write, and so on).

The preferred and most optimized IBM Z platform for Linux on Z data at-rest encryption is the combination of the z14 (including CPACF enablement), Crypto Express6S, and the latest release of z/VM (V7.1) and Linux on Z distributions.

Note: At the time of this writing, KVM support for Linux on Z data at-rest encryption is *not* available from the Linux distributions.

2.2 Required and optional hardware features

In principle, cryptographic algorithms can run without extra hardware. However, cryptographic algorithms are computationally intense, and the secure handling of keys requires special hardware protection. Therefore, Linux on Z data at-rest encryption takes advantage of the IBM Z platform's cryptographic hardware features to meet the requirements for bulk cryptographic processing.

The following IBM Z platforms and Crypt Express features support Linux on Z data at-rest encryption:

- IBM z14™ with Crypto Express6S (FC 0893) or Crypto Express5S (FC 0890)
- IBM z14 ZR1 with Crypto Express6S (FC 0893)
- IBM z13 or IBM z13s® with Crypto Express5S (FC 0890)
- IBM zEnterprise® EC12 or IBM zEnterprise BC12 with Crypto Express4 (FC 0865) or Crypto Express3 (FC 0864)
- IBM zEnterprise 196 or IBM zEnterprise 114 with Crypto Express3 (FC 0864)

This section introduces the following IBM Z hardware components to consider for use with Linux on Z data at-rest encryption:

- Central Processor Assist for Cryptographic Function
- Crypto Express adapters
- TKE workstation

2.2.1 Central Processor Assist for Cryptographic Function

The no-charge CP Assist for Cryptographic Function enablement (FC 3863) is required on the IBM Z hardware platform to support Linux on Z data at-rest encryption.

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.

2.2.2 Crypto Express adapters

Crypto Express adapters are required to generate the secure keys that are stored in the key repository. They are also required to generate protected keys from secure keys for Linux on Z data at-rest encryption.

Crypto Express adapters are tamper-responding hardware security modules[1] (HSM), which provide high-security, high-throughput cryptographic functions. The Crypto Express adapter adds a layer of protection for the storage and use of a master key.

> **Note:** All installed Crypto Express adapters must be loaded with the same level of code. Otherwise, unpredictable results can occur.

The following Crypto Express configuration options are available:

- Accelerator
- CCA coprocessor
- EP11 coprocessor

> **Note:** For Linux on Z data at-rest encryption, the Crypto Express adapters must be configured as Common Cryptographic Architecture (CCA) coprocessors.

CCA is an architecture and a set of application programming interfaces (APIs) that support cryptographic operations and key management.

To access and use the Crypto Express adapters, applications must use APIs and panel utilities that are provided by the operating system. For Linux on Z, the IBM CCA package (IBM Common Cryptographic Architecture) provides these APIs and manages access to the hardware cryptographic features.

The CCA package can be downloaded from this IBM Security web page.

Determining capacity

To determine the level of capacity that is needed to satisfy the demand on your Crypto Express adapters, the following tasks are performed:

- Assess your workloads and their behavior during peak periods.
- Define thresholds that adhere to your capacity policies and monitor usage.
- Ensure that enough capacity is available for backup situations.

A minimum of two Crypto Express adapters is recommended so that if one adapter must be taken offline (for example, microcode upgrade), the second adapter (loaded with the same master keys as the first) can handle the required workload. In this case, the utilization threshold for each Crypto Express adapter should not exceed 50%.

After initial setup, regularly monitor the use of each Crypto Express adapter while your crypto workloads are running. If the adapter utilization exceeds the wanted threshold, you can increase the number of Crypto Express adapters.

[1] An HSM is a physical computing device that safeguards and manages digital keys for strong authentication and provides cryptographic processing.

Monitoring utilization

The following options are available for viewing or monitoring CryptoExpress adapter usage:

- IBM Z Hardware

  Monitors Dashboard on the Hardware Management Console (HMC). This option shows Crypto Express adapter type and monitors usage in real time.

- z/VM

  MONITOR records (Domain 5, records 9 and 10) are available that have Crypto Express usage at a VM level, and more data is in the CPUMF (Crypto Counters) that shows CPACF usage at an LPAR level.

- Linux

  A few options are available with Linux on Z to monitor usage by using Linux commands; for example, lszcrypt, cpacfstats, perf stat, and the CPU-MF counters of the crypto-activity counter set.

  Note: CPU-MF interfaces are only available to Linux on Z in a native LPAR.

  An open source project that is sponsored by IBM is available that is called Linux Health Checker (LNXHC). The latest version of this project features 10 cryptography health checks out of a total of 70 and is primarily focused on Linux on Z.

For more information, see Chapter 6, “Auditing and monitoring the data at-rest environment” on page 71.

2.2.3 Trusted Key Entry workstation

The Trusted Key Entry (TKE) workstation is an optional hardware feature that can be used for Linux on Z data at-rest encryption.

TKE securely manages multiple cryptographic coprocessors (including master keys) on various generations of IBM Z and other platforms, from a single point of control. Manually managing master keys across a complex installation can require significant systems management effort, introduce audit and secrecy complexity, and can be error prone at critical master key entry stages.

Master key management is not a trivial endeavor in an IBM Z environment. For example, an IBM z14 supports up to 16 Crypto Express6S adapters, each of which can support 85 domains[2]. That means on a single IBM Z platform you can have up to 1360 master keys to manage. For more information about considerations regarding master key management, see 3.6.10, “Choosing key management tools” on page 33.

TKE 9.1 includes a secure hardware-based workstation (FC 0085 or FC 0086) and 4768 Crypto Express adapter, with smartcard-controlled key management. This configuration provides secure, fast, and accurate deployments of new cryptographic material across production, test, and disaster recovery (DR) systems.

[2] The IBM Z platform uses the concept of a cryptographic domain to virtualize the physical coprocessor of the CryptoExpress adapter. A Crypto Express coprocessor can be shared by multiple logical partitions (LPARs) and different operating systems. IBM Z firmware enforces domain usage. The Crypto Express coprocessor manages the assignment of master keys to cryptographic domains. Cryptographic key material for one domain is not usable by another domain with a distinct master key.

For more information about other TKE workstation versions that can be used with earlier generations of the IBM Z platform, see the *TKE Hardware Support and Migration Information* IBM Techdoc.

2.3 Required and optional software features

Depending on the available software and software levels, different features are available for data at-rest encryption. As of this writing, not all current Linux distributions support all of the data at-rest encryption options. Therefore, it is important to review the software requirements before trying something that is not possible on the used distribution.

2.3.1 Required software components for dm-crypt with a plain format setup

The following components are required to use the plain format for data at-rest encryption:

- The Linux kernel version 4.11 with the `paes_s390` and `pkey` modules
- `s390_tools` version 1.39 with the `zkey` utility

The following Linux distributions support data at-rest encryption with the plain format:

- Red Hat RHEL 7.5, 7.6 (Red Hat backported software from upstream kernel back into their 3.10 line)
- SUSE SLES 12 SP4, SLES 15
- Ubuntu 18.04, 18.10

2.3.2 Required software components for dm-crypt with an LUKS2 format setup

The following components are required to use the LUKS2 format for data at-rest encryption:

- The Linux kernel version 4.11 with the `paes_s390` and `pkey` modules
- `s390_tools` version 1.39 with the `zkey` utility
- The `cryptsetup` utility in version 2.0.3

Note: Rolling the master key requires `s390_tools` version 2.6, which includes the `zkey-cryptsetup` utility.

At the time of this writing, only Ubuntu 18.10 supports the LUKS2 format with secure key operations. Contact your Linux on Z distribution to determine whether LUKS2 format with secure key support is available.

2.3.3 Optional software components

Some software features are optional because they enhance some functionality or make the cryptographic operations faster.

Optional components for performance enhancements

Performance can be greatly enhanced by using dm-crypt with 4096-byte sectors and not with the default 512-Byte sectors. However, this change includes the following minimum software requirements:

- Linux kernel version 4.12

- cryptsetup version 2.0.0

The following Linux distributions support 4096-byte sector sizes:

- Red Hat RHEL 7.5, 7.6
- SUSE SLES 15
- Ubuntu 18.04, 18.10

> **Note:** At a minimum, system version 240 is required to use `/etc/crypttabs` with LUKS2 format and 4096-byte sectors. The only distribution that meets this requirement at the time of this writing is Ubuntu 18.10.

Optional components for key repository support

Newer versions of the zkey utility allow it to create a local key repository for managing data keys. This capability requires version 2.4 of the `s390_tools` package and is available on the following distributions:

- Red Hat RHEL 7.6, 7.7
- SUSE SLES 12 SP4, SLES 15
- Ubuntu 18.10

Optional components for monitoring

LNXHC is an open source package that can benefit any Linux on Z environment. It provides 70 health checks for various components of the Linux environment. The checker also includes the ability to add your own custom checks.

2.4 Cost and performance

When choosing which hardware and software components to configure for your Linux on Z data at-rest encryption environment, the following performance considerations are important:

- Operational encryption and decryption performance depends heavily on the capability of the CPACF processor and the data block size.
- Encrypting or decrypting larger blocks of data improves performance.
- With z14 and CPACF, encryption performance improvements compared to z13 and CPACF of up to seven times were measured by IBM.

> **Note:** Be aware that each implementation is unique. Actual performance results can vary, depending on your specific configuration, operating conditions, and workloads.

The cost of widespread data at-rest encryption on a z14 (compared to not encrypting on the z14) might be a low single-digit percentage increase. On a z13, the cost might be significantly greater and should be investigated carefully as part of any z13 encryption project planning. Performance cost can be considerably higher on all generations of processors before z14.

z/VM encrypted paging on IBM z14 still costs less than unencrypted paging on the IBM z13 and the CPU cost of encrypted paging is a function of the paging rate rather than the LPAR size. Therefore, if your z/VM environment does little or no paging, the CPU cost is much lower.

Performance capability of the Crypto Express6S coprocessor (available for the z14 only) also increased significantly, compared to the Crypto Express5S and previous versions. Other functional improvements were made in the Crypto Express6S, in addition to performance.

3

Planning for Linux on Z data at-rest encryption

This chapter describes planning considerations for implementing Linux on Z data at-rest encryption from several perspectives. It includes the following topics:

3.1 Creating an implementation plan

The following approach is suggested for starting a Linux on Z data at-rest encryption implementation plan:

- Understand the scope of the data you want to protect.

  For example, consider what data will be protected. Must the data be protected to satisfy an encryption initiative, such as to satisfy regulatory compliance, or other security requirements? The pervasive encryption approach suggests to encrypt all critical and sensitive data.

  Based on the data that must be protected, create a plan for the volumes that need protection (for more information, see 3.2, “Know where your sensitive data is stored” on page 23).

- Consider a pilot project for an internal proof of technology.

  Develop a use case for the project. Based on the data and volumes that must be protected, create volumes for one of your guests.

- Ensure that the IBM Z platform is ready for Linux on Z data at-rest encryption.

  At a minimum, ensure that the system to be used for the proof of technology satisfies the prerequisite hardware and software levels. Also, consider any other middleware that is required based on the use case to be evaluated.

  In addition, consider the items in the Readiness checklists. Most of the items might not be needed during a proof of technology, but should be evaluated before implementation on production systems. For more information, see 5.2, “Deployment readiness checklist” on page 59.

- Implement the proof of technology and review and assess carefully the expected performance and operational outcomes.

  Prepare the environment by completing the configuration and setup steps; then, complete the required deployment steps to create and access encrypted volumes. Run the application to ensure that it can successfully access the encrypted volumes.

  After the results of the proof of technology are satisfactory, continue with developing a strategy for the broader Linux on Z data at-rest encryption implementation.

- Develop operational processes that protect and maintain the implementation.

  Operational processes might include, but are not limited to, the areas of access control policies, key management, auditing, high availability, disaster recovery, and backup and restore. Consider practicing and refining these operational processes over time.

- Determine how Linux on Z data at-rest encryption should be rolled out to production systems.

  Because the implementation process requires creating volumes or migrating volumes, encrypting volumes individually might be the preferred approach. Therefore, your implementation plan must include multiple phases that are based on the criteria that is used to identify the volumes that must be encrypted.

3.2 Know where your sensitive data is stored

Industry regulations, such as European Union (EU) General Data Protection Regulation (GDPR), Payment Card Industry Data Security Standard (PCI-DSS), and Health Insurance Portability and Accountability Act (HIPAA) require organizations to protect sensitive data. These regulations impose sharp penalties for the disclosure of sensitive data. Which regulations apply to your organization? Is your sensitive data protected?

One of the most effective ways to protect sensitive data is to establish a perimeter around that data. Traditionally, the perimeter was considered the network, which was protected with firewalls and VPNs. Today, we recognize that attackers can breach the network perimeter; therefore, the sensitive data must be protected at the source by using encryption and implementing security policies.

To truly protect sensitive data, you must know what data to protect and where that data is stored and who has access to it. Recognizing sensitive data might not be difficult with formatted data, such as credit card numbers, social security numbers, and passwords. However, identifying all the places where that sensitive data is stored can be a challenge.

Security administrators must consider the following questions:

- Is sensitive data in a database?
- Is sensitive data in a file or volume?
- Is sensitive data in memory?
- Will sensitive data appear in a dump?
- Is sensitive data in the network?
- Is sensitive data on a backup tape?
- Is sensitive data shared with a third party?

3.3 Encryption considerations

The Linux on Z volumes for data at-rest are encrypted by a secure key that can be used only if the correct domain with the correct master key is configured for your guest or LPAR. The secure key can also be wrapped by a passphrase or keyfile when the LUKS2 format is used.

Security note: A passphrase adds security to the system only if the key repository is not accessible because only the secure key in the LUKS2 header is wrapped by the passphrase. Otherwise, the secure key in the key repository might be used to skip the passphrase.

3.3.1 How many secure keys do I need?

A secure key is wrapped under the master key of a domain, which is configured for Linux on Z. Therefore, no one outside the Linux operating system can use the secure key if the master key is unique to this domain.

In general, one secure key can be used for multiple volumes within a Linux on Z guest. However, we suggest to use a secure key for every encrypted volume and not to share one secure key across multiple volumes. This configuration makes it easier to replace compromised keys and re-encrypt volumes, and is in general, a best security practice.

Access to the secure keys or the secure key repository should be audited and only the `root` user or the user in the `zkeyadm` group should have access to the files because only those users must access them.

3.3.2 Should I use a known key?

The **zkey** commands enable importing a secure key from a known clear key. However, importing a clear key is not recommended. Importing clear keys is done only in a secure environment where it is known that only trusted users can access that environment.

It is recommended that a secure key is generated by using **zkey** commands. This process uses the master key that is loaded in the Crypto Express adapter to wrap the clear key. This configuration ensures that no one can access the key material without authorization.

3.3.3 Which cipher mode should be used?

The National Institute for Standardization (NIST) recommends the XTS mode for data at-rest encryption. AES CBC can also be used for data at-rest encryption; however, at the time of this writing, the Linux on Z data at-rest implementation cannot use the Encrypted Sector Salt Initial Vector (ESSIV), which makes the implementation potentially vulnerable.

Other modes of operations are less secure (such as ECB, which is not used in any use case). Therefore, XTS is used for all Linux on Z data at-rest encryption use cases. For more information, see *Recommendation for Block Cipher Modes of Operation: The XTS-AES Mode for Confidentiality on Storage Devices,* NIST Special Publication 800-38E.

3.3.4 Should I use keyfiles or passphrases?

For the LUKS2 format, a passphrase or keyfile can be used to protect the secure key that is integrated in the LUKS2 header. A keyfile is an automatic way to open a LUKS container and contains a random value, which acts as passphrase.

A keyfile does not add any security to the volume encryption. If more security is needed, a passphrase should be used (for more information, see the Security note on page 23). However, the partitions are not unlocked automatically after an IPL; therefore, it is recommended that keyfiles are used.

3.3.5 Should I consider root partition encryption?

At the time of this writing, root partition encryption is not supported by any Linux on Z distribution.

Encrypting the root partition ensures that the partition cannot be mounted when the correct domain with the correct master key is not available. By using the HMC serial console or the z/VM console, it is also possible to provide a passphrase to unlock the root device. If this option is not needed, the passphrase can also be given as a keyfile.

3.4 Considerations for separation of duties

An essential attribute of Linux on Z data at-rest encryption is separation of duties between data owners and administrators. The concept of separation of duties suggests that more than one person is needed to complete a security-related task. This process helps avoid conflicts of interest and can better detect control failures that lead to security breaches, information theft, and violations of corporate security controls and policies.

In addition, the notion of creating a perimeter around the data means that you can limit who can access your sensitive data. In many organizations, the same person might have multiple administrator roles to fulfill. To help protect against non-intentional changes to the security ecosystem, each administrator role has its own user ID (with the appropriate access profile and authority level). By using this approach, an individual with several roles is forced to change to the user ID in which each administrator role is performed with a corresponding audit trail.

At a minimum, the staff that supports a Linux on Z data at-rest encryption environment must possess security, cryptographic, system, and storage administrator roles as described next.

3.4.1 Security administrator role

The security administrator governs and oversees the security policies for an organization's sensitive data. With encryption, this governance can also involve ensuring cryptoperiods are defined and enforced, and the lifecycle of each encryption key (from creation through deletion) is recorded and audited.

The security administrator works closely with data owners to determine which data is encrypted and that the correct level of protection is applied through user access policies.

The security administrator is responsible for identifying data and volumes that must be encrypted, permitting the creation and deletion of keys and encrypted volumes, and for defining the security policies to prevent unauthorized access to the encrypted volumes. This also includes ensuring security policies are in place and active, and the security ecosystem meets compliance and regulation requirements.

3.4.2 Cryptographic administrator role

The cryptographic administrator manages the cryptographic system and secure keys that are used for Linux on Z data at-rest encryption. Responsibilities include generating, updating, rotating, and deleting secure keys. The cryptographic administrator works with the master key-part owners[1] to generate and load the master keys to the CCA of the Crypto Express adapter.

Note: Even if master key-part owners are designated, secure key management and master key management should be carried out by two different staff members.

This role can involve monitoring the use of the Crypto Express coprocessor and CPACF for performance and capacity planning purposes.

[1] Master keys always consist of two or more key parts with a different owner for each master key part. (No single person has the entire master key material.) The master key-part owner adds their part separately and securely to the utility (a TKE workstation or by way of panel.exe) to generate a master key. The cryptographic administrator should not have access to any of the master key material. Having multiple key-part owners ensures the master key's integrity and help avoid it from being compromised.

A cryptographic administrator can also have TKE administrator responsibilities. The TKE administrator manages the TKE workstation that loads master keys. The TKE administrator can enable and disable access control points for callable services that are on cryptographic coprocessors in the TKE workstation.

3.4.3 System administrator role

A system administrator's responsibilities include managing and maintaining a multiuser computing environment. These responsibilities can vary depending on an organization's requirements and needs. System administrators maintain hardware and software levels to support the encryption system. They also work with the security and storage administrators to determine actions that are related to the creation of encrypted volumes and who can access them, which ensures the correct level of protection is applied through user access policies.

For an IBM Z platform, system administrators include z/VM administrators and Linux on Z administrators.

Typically, a z/VM administrator manages guests across multiple z/VM LPARs. In a z/VM environment, several workloads feature different authorities and access rights. Therefore, a z/VM administrator can be granted different levels of access, depending on their level of authority. A z/VM administrator can have varying access to resources, such as virtual machines and cryptographic hardware and software, which should not include access to encryption keys or sensitive data.

In the case of a Linux on Z administrator, they are responsible for an entire virtual machine. This virtual machine is built for isolation and then it is decommissioned and destroyed at the end of its lifecycle. Therefore, an emphasis is placed on the separation of workloads so that one process or program does not obscure another. A clear distinction should exist between separation of resources, workloads, and duties, and the system commands to implement them. The Linux on Z administrator should not have access to encryption keys or sensitive data.

3.4.4 Storage administrator role

A storage administrator works closely with the security, cryptographic, and system administrators to ensure all sensitive data at-rest is properly protected. The storage administrator is responsible for coordinating the creation of encrypted volumes and for managing backup, migration, and replication of encryption keys and encrypted data based on the organization's guidelines.

3.5 Policy considerations

Encryption is just the start when protecting data at-rest. Only in combination with policies and access control methods can data at-rest be effectively protected. The access control mechanisms include:

Authentication Requires an identity (such as a user ID) and a secret (such as a password) to log in to a system.

Authorization Establishes a set of policies to determine which users can access which data and services.

Linux provides some tools to protect data with access control mechanisms. Because the topic is so broad and business policies vary, this section provides only general guidance regarding access policies. For more information, see your Linux distribution manual.

For more information about access control mechanisms at the z/VM level, see 3.7, “z/VM considerations” on page 36.

3.5.1 Securing the root account

The following description is a starting point for securing the root account and gives a few pointers about which actions can be taken.

Normally, all actions that are related to key and volume management need root access. However, with the help of `sudo` policies and group user management, a separation of duty can be implemented between the key management administrator and the volume administrator. This separation makes sense only when LUKS2 format is used. For plain mode encryption, the user who opens the volume also must have access to the key repository.

The root user is the administrator of the Linux operating system and can override all security policy checks. Normally, access to the root user is controlled only by password authentication. The `sudo` and `su` tools escalate privileges of a normal user to root. `Sudo` policies can be enabled to restrict which commands can be run by a normal user with `sudo`.

Disabling the root account

To make administrative action controllable and auditable, the root user can be disabled and administrative actions can then be done by using **`sudo`** commands, for example. Disabling logging in to the root user is the first step for securing the root account.

In Example 3-1, the two commands are shown that are needed to disable the root account. Running these commands also disables the `su` tool because it needs the root password to work. Instead, `sudo` can then be used for privilege escalation.

> **Attention:** Before disabling the root account, make sure to create a user that can run the **`sudo`** command to gain root privileges. Failing to create this user results in a non-administrable system.

Example 3-1 Commands needed to disable the root account

```
sudo passwd -d root # Remove the root password
sudo passwd -l root # Lock the account
```

Defining access policies

When the root user is disabled, `sudo` can be used to take administrative action. Normally, all users that are in a specific group (often called `wheel` or `adm`) can run `sudo` to escalate their privileges to take administrative actions.

The `/etc/sudoers` file can be edited to allow only some commands be run by using `sudo`. For example, one user can be given the permissions to run **`cryptsetup`**, the **`mount`** command, and the **`lvm`** commands to configure and administer the Linux storage system.

For more information, see the sudo web page and sudoers web page.

> **Important:** Always use the **`visudo`** command to change the `sudoers` file, Use caution when you are changing anything in this file. Incorrect changes can lock you out of your own system.

Access to the `zkey` repository does not require root rights. Users that require access to the repository must be added to the `zkeyadm` group only.

3.5.2 Enable auditing

Even when a separation of duty exists between key management and volume activation, an administrator must exist who can change the `sudoers` file and manage users.

To control administrator activity, auditing must be activated and the logs sent to a separately controlled area. The audit records must be checked regularly. Tools also are available that scan audit records for abnormalities and notify administrators if an issue is found.

For more information see, Chapter 6, “Auditing and monitoring the data at-rest environment” on page 71.

3.5.3 UNIX file permissions and POSIX access control lists

UNIX permissions and access control lists (ACLs) are used to control file and directory permissions after the volume was open and the filesystem was mounted. Encryption does not help against a user that can read the files after the volumes are opened. The correct file permissions must be used to enforce that only the user or application can access the files.

It is also important to run applications with minimal privileges, which helps when the application is hacked because if the applications runs under root, the hacker can perform all of the administrative actions and access all of the files. If the application runs under another user, only administrative actions that are allowed for this user can be used and only files from this user are compromised.

Traditional UNIX file permissions allow it control read, write, and run privileges to a file or directory for the owner and group, and the rest of the users that are not the owner and not part of the group. If this level of protection is not enough, ACLs can be used to define granular control access rights of a file or directory.

For more information about UNIX file permissions, see this web page.

For more information about POSIX ACLs, see this web page.

3.5.4 Mandatory access control systems

Mandatory access control (MAC) systems, such as `apparmor` or `SELinux`, add an access control mechanism to control which objects a process can open and control. Even when a process runs under the user, no one has read access to files that are not needed for the process to run correctly.

If an `apparmor` or `SELinux` policy exists, the MAC systems can filter all access requests and allow only the requests that are crucial for process execution.

For more information about `apparmor`, see this page of the ubuntu wiki.

For more information about SELinux, see this Red Hat web page.

3.6 Key management considerations

Key management is a critical aspect in any encryption strategy. Industry regulations typically require key management processes are created and well-documented.

Terminology: Data and effective keys are *operational keys*. Data keys can be clear, secure, or protected keys.

3.6.1 Understanding key management

Cryptographic keys feature a lifecycle that includes tasks, such as key creation, key activation, key deactivation, and key deletion. Some regulations, such as PCI-DSS, require that key management practices are in place.

Consider the following questions:

- What regulations must be considered?
- What key types and lengths will be used?
- Will the keys be stored in the clear or encrypted?
- How long will keys be active?
- What happens to a key after it is deactivated?
- When should keys be archived?
- How will a key be handled if it is compromised?
- How often will keys be backed up?
- How will keys be distributed to other systems?
- Who will own the key (such as the user or the application owner)?
- What metadata should be associated with a key?
- Will keys be rotated? How often? Which keys?

The various key management areas are briefly explained in the following sections.

3.6.2 Reviewing industry regulations

Before your key management plan is created, identify and review any regulations that might be required for compliance. For regulations that are generic or non-specific, work with your auditors to clarify ambiguities and review your key management plan.

For more information, see 1.1, “Why protect your data beyond compliance” on page 2.

3.6.3 Determining key security

The **`dm-crypt`** command can be used with clear and secure keys for encrypting and decrypting volumes.

We recommend the use of secure keys. A secure key is a data key that is encrypted by using a master key. Therefore, all encrypted data is unreadable without a master key.

The master key should be created from two or more key parts by using a different key owner for each master key part. By using this configuration, reading sensitive key material requires access to the key repository and access to all master key parts.

When clear keys are used and the key repository is dumped, all keys are readable. By using secure keys, dumping the key repository does not yield any sensitive data.

Protected keys are not stored in the key repository. They are created from secure or clear keys and stored in kernel memory. When Linux is restarted, all protected keys are cleared from memory.

For more information, see 3.6.8, “Establishing a process for handling compromised operational keys” on page 32.

3.6.4 Choosing key-part owners

Master key materials should consist of two or more key parts. A different owner for each key part. In the case of disaster recovery or when Crypto Express adapters are added, all key-part owners must be available and present to load their master key part.

Based on this criteria, the following questions must be answered:

- How many key-part owners will you have?
- Who will be those key-part owners?
- How often will master keys need to be changed?
- How will you ensure that the key-part owners will not collude and compromise the master key?

3.6.5 Key lifecycle

After a key is generated, it progresses through multiple states during its lifecycle. The key and its lifecycle must be managed by an authorized administrator.

The following types of keys must be managed in a Linux on Z data at-rest encryption environment:

- Master keys

  These keys are stored in a Crypto Express adapter and used to encrypt operational keys.
- Operational keys

  These keys are stored on the host system in a key repository (for example, the `zkey` repository) or in memory. They are used to perform various cryptography operations.

For more information about the key types that are associated with Linux on Z data at-rest encryption, see 1.3.1, “IBM Z cryptographic system” on page 7.

A simplified key lifecycle from creation (start) to deletion (destroyed) is shown in Figure 3-1 on page 31. Keys can be defined with a valid start date and end date, which is known as the *cryptoperiod*. A cryptoperiod can be used to control when a key is allowed for use in crypto operations.

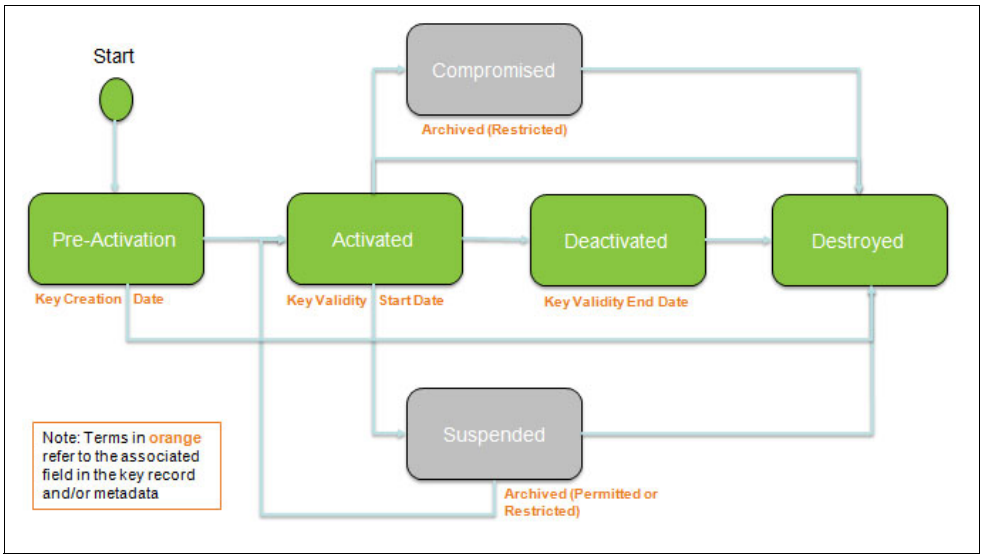

Figure 3-1 Key lifecycle (simple)

Operational key lifecycle

The following terms also are used in referencing the key lifecycle:

- Creating

  Creating is the starting point for generating keys. You must determine who has the authority to generate keys. In this book, **zkey** commands are used to create keys and only root or users in the `zkeyadm` group can use them.

- Updating

  The process of updating or changing keys (rekeying) is performed in instances where a key was determined to be compromised or a security policy states that data must be rekeyed regularly. In both cases, a new key must be generated and the volumes must be reencrypted.

- Deleting

  A secure key is deleted only after all of the volumes that are associated with this key were reencrypted or migrated to another key.

3.6.6 Defining key rotation

The following options are available to rotate keys:

- Rotate the master key

  This option is the simplest method. The new master key must be loaded and then the keys can be re-enciphered by using the **zkey** commands.

- Rotate the operational key

  Rotating the operational key is a complex operation that is done only if the key is compromised or if regulations require rotating operational keys.

Company guidelines often regulate which key must be rotated. Master keys can be rotated with a TKE in a secure manner. Rotating operational keys needs a re-encryption of the entire volumes; rotating a master key needs only the re-enciphering of the operational key, which takes less time.

For more information about setting master keys, see the TKE manual.

For more information about re-enciphering operational keys, see 8.3, “Rotating a master key” on page 92.

For more information about re-encrypting volumes, see 7.3, “Re-encrypting an LUKS2 format volume by using a different secure key” on page 84.

3.6.7 Establishing cryptoperiods

A cryptoperiod defines the time in which a key is active. It is the time between the key activation start date and end date.

Some regulations require keys to have a clearly defined cryptoperiod. When the end date is reached, the key reached its end of life and can be revoked or destroyed. The data that is protected by that key is destroyed or reencrypted with a new key.

Before a cryptoperiod is established, consider the following questions:

- Should cryptoperiods be established for data at-rest operational keys?

  Does a regulatory requirement exist?
- What is an appropriate cryptoperiod?

  Does a regulatory requirement exist?
- What happens to a key at the end of its cryptoperiod?
  - Would the cryptoperiod ever be extended?
  - Will encrypted data be reencrypted with a new key?
- What happens to the volume at the end of the key’s cryptoperiod?
  - Should the volume be destroyed?
  - Should the volume be rekeyed?
- How will administrators identify expired or soon-to-expire keys?

For more information, see the National Institute of Standards and Technology publication *SP 800-57 Part 1 Rev. 4, Recommendation for Key Management, Part 1*.

3.6.8 Establishing a process for handling compromised operational keys

When an operational key that is used in data at-rest encryption is compromised, a plan should be in place to manage the key and the encrypted data.

Volumes that use a compromised key should be set physically offline. Also, the data should be moved as fast as possible to a different volume and the volumes should be securely rewritten.

After all affected volumes are no longer used, the compromised key should be removed to prevent anymore usage of the keys (for example, mistakenly creating a volume by using the compromised key).

For more information about handling compromised keys, see 7.3, “Re-encrypting an LUKS2 format volume by using a different secure key” on page 84.

3.6.9 Establishing a process for handling compromised master keys

In the highly unlikely event that a master key is compromised, it should be immediately changed and the keys must be re-enciphered twice. Crypto Express adapters contain the following master key registers for each master key type:

- New master key (NMK)
- Current master key (CMK)
- Old master key (OMK)

New master keys are loaded into the NMK register. When the master key is set, the NMK register contents are moved to the CMK register. The CMK register contents are then moved to the OMK register, and the NMK register is cleared. In this way, keys that are encrypted by the OMK can still be used on the system.

In the case of a compromise, the OMK should be cleared or overwritten. Therefore, two master key change operations must occur to completely clear the compromised master key from the system.

For more information about loading a master key, see 4.4, “Loading the master key” on page 55.

For more information about re-enciphering the keys in the key repository, see 8.3, “Rotating a master key” on page 92.

Note: Secure keys that use the compromised master keys must be replaced.

3.6.10 Choosing key management tools

Managing cryptographic keys is vital to the overall security of your encrypted data. If the cryptographic keys are compromised, your encrypted data also can be compromised.

Available tools

The tools that are available for key management in Linux on Z and the keys you can manage with them are listed in Table 3-1.

Table 3-1 Tools and what they manage

Tool	Manage master keys?	Manage operational keys?
zkey commands	No	Yes
Trusted Key Entry (TKE)	Yes	No
CCA panel.exe utility	Yes	No

Consider the following points:

- Master keys can be managed by using `panel.exe` or a TKE workstation.
- Operational keys Linux on Z data at-rest encryption can be managed by using the **zkey** command.

Managing master keys

The master key is stored in the Crypto Express hardware security module (HSM) and can be managed by a TKE or by utilities and tools that are provided by the Common Cryptographic Architecture (CCA) Support Program (when the Crypto Express HSM is configured in CCA mode).

Choosing master key owners

Best practice for master key management involves building master keys from multiple key parts where each key part is owned by a different person. Identify who are the key owners and what process they follow for loading master keys.

Loading master keys

The following methods that can be used to load and set a master key are listed in order of strongest security first:

- TKE workstation

  This method is the most secure way to load and set a master key. It involves smart cards and smart card readers. The master key material can be generated directly onto the smart cards and cloned to backup smart cards.

  Each key owner generates their own key part and all owners must be present to complete the key loading process. The key material never needs to be displayed on a computer. For more information, see the TKE video series.

- The CCA `panel.exe` utility

  This method is the least secure way to load and set a master key. It involves entering a 64-character passphrase that generates and loads the master keys onto the Crypto Express adapters and initializes the keys.

  Because the key material is displayed in a window, a process must be in place to ensure that the key material is not disclosed to unauthorized users. Also, the generated key material must be captured and saved for future reentry if disaster recovery is needed or the new adapters are installed.

Managing operational keys

Operational keys are defined as all keys that are not master keys. These keys can be in a keyfile or in memory on the host system. The secure keys that are used with volume encryption are stored in the `zkey` repository and in the LUKS2 header of the volume. Secure keys in the `zkey` repositories are linked to the volumes where the key is used.

Generating operational keys

The **`zkey`** commands can generate random operational keys for use as volume keys or generate them from a known clear key value. The preferred way is to generate the key from random because it is difficult to ensure that the clear key value is secure.

The keys can be stored as a file on disk or in the `zkey` repository. You can use the **`zkey`** commands to associate metadata to the key and get a summary of the key usage and lifecycle. Example 3-2 shows a 256-bit AES key[2] that is used for an LUKS2 format volume.

Example 3-2 zkey list of a 128-bit AES key

```
Key                         : secure_xtskey
--------------------------------------------------------------------------------
```

[2] Two 256-bit keys are saved here because AES XTS needs two keys. For more information, see 8.1, “The zkey repository” on page 88.

```
        Description          : A 256-bit AES key that is used for an LUKS2 format
volume.
        Secure key size      : 128 bytes
        Clear key size       : 512 bits
        XTS type key         : Yes
        Volumes              : /dev/dasdc1:enc-disk1
        APQNs                : (none)
        Key file name        : /etc/zkey/repository/secure_xtskey.skey
        Sector size          : (system default)
        Volume type          : LUKS2
        Verification pattern : 7dc408c6ffbcb861b377bd3b7a48fa6f
                               3781f8972c74c1ff148219c19080a92f
        Created              : 2019-01-24 16:23:22
        Changed              : (never)
        Re-enciphered        : (never)
```

3.6.11 Backing up keys

Volumes that are encrypted by using the LUKS2 format include their own secure key in the LUKS2 header. In plain format, the secure key must be retrieved from the key repository by every **luksOpen** to the volume. In both cases, a backup of the key repository must be made.

Creating backups of keys

The key repository is a critical component of Linux on Z data at-rest encryption. Not only must you protect it from unauthorized users, you must have backup procedures in place as part of your normal housekeeping routines to ensure that regular and valid copies of the repository are available.

> **Note:** Any keys that were created between the time of the backup and the date of recovery are lost. Therefore, it is important that backups are taken regularly.

We recommend manually backing up the key repository *before* a major operation (such as rotating encryption keys or manually transporting a key from one key repository to another). After the operation is completed and the key repository contents are verified, the backup can be deleted. If verification is unsuccessful, the key repository can be recovered from the backup.

The second critical component is the header of LUKS2 volumes. If the header is destroyed, the data on the disk cannot be easily recovered without a full header backup.

Backup and restoration include the following considerations:

- How often will the key repository and the LUKS2 headers be backed up?
  - How often are new keys created?
  - Will keys be backed up before and after major key operations?
- How many backup versions will be kept?
- What tools will be used for backup?

The key repository and the LUKS2 header can be backed up and restored by using a set of different tools.

3.6.12 Planning for disaster recovery

Disaster recovery planning often is regulated by company-wide guidelines. Disaster recover for cryptography is only a small part of this planning process.

To plan for disaster recovery, you must determine whether your remote site meets the following requirements for data at-rest encryption:

- Replicated copies of Linux on Z volumes are also encrypted and access to the key repository is provided.
- Cryptographic coprocessor configurations are replicated across both sites, including the master key. This replication must be done initially and with every master key change. The process can be simplified by using TKE domain groups.

Figure 3-2 shows a disaster recovery solution of cryptographic key material for multi-site environment.

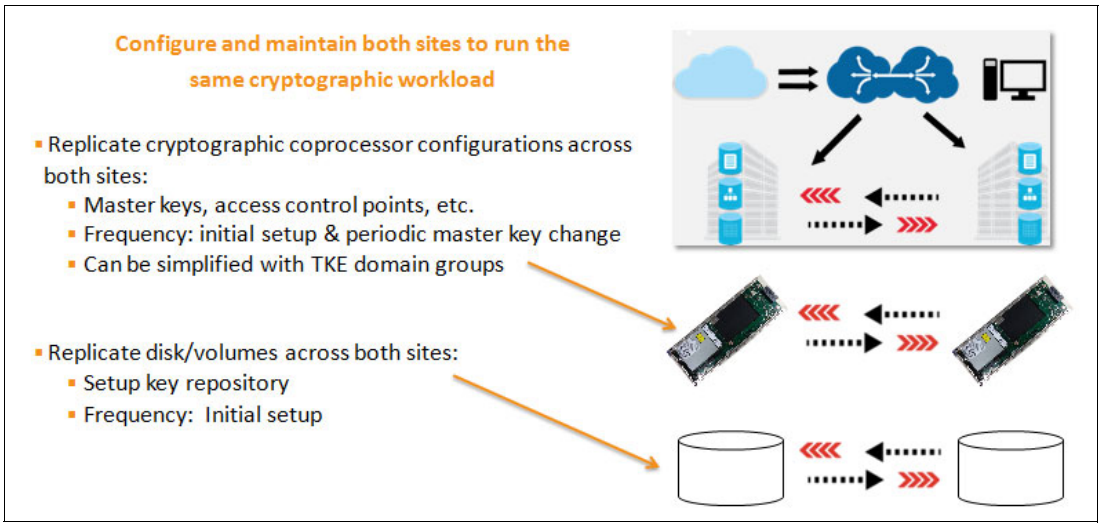

Figure 3-2 Disaster recovery planning

3.7 z/VM considerations

To protect data at-rest with Linux on Z, encryption is only one part of the solution. The virtualization layer, z/VM configuration, is another part.

Consider Linux data at-rest encryption another policy that must be deployed along other business policies. No other policies are mitigated by the introduction of this new policy.

3.7.1 Policy considerations

z/VM provides isolation and protection of virtual machines from each other, and between virtual machines and the system overall. Although, the core capability of security and integrity is provided by z/VM without an External Security Manager (ESM).

Using an ESM

An ESM is an external software product that manages user identities and controls access to system resources. Examples of ESMs are IBM RACF® or CA Top Secret and ACF2.

We strongly suggest to use an ESM to enable separation of duties at the z/VM level.

Organizations that must comply with government and industry regulations on the control and management of customer and client data often require a level of security protection beyond what can be provided by using z/VM internal security mechanisms. Also, the fact that any z/VM administrator without an ESM can easily determine any password on the system can be a significant issue in many regulatory domains.

To satisfy the requirements of a security audit, a necessary step often is to demonstrate that data that is owned and managed by a system cannot be accessed by a system that belongs to a different security profile. This fact can be difficult to demonstrate without an ESM.

An ESM is also needed to restrict access to the Linux on Z console because it is not designed for day-to-day use. It is intended only for interactive use during the installation process or system recovery. The extensive use of it introduces risks to the security and availability of the environment.

Directory management utility

Use a directory management utility, such as `IBM/DIRMAINT`, to manage the definitions of guests and their resources in the z/VM user directory file.

By using a directory management utility, you can track any changes in the dedicate or AP guest configuration that is done on CP directory, for example. If anything regarding the **`CRYPTO`** statement changes incorrectly, it is always possible to identify what was the root cause.

Without a directory management utility, it is much more difficult to track such changes because the CP user directory must be changed manually by a z/VM system administrator by using an editor, such as `XEDIT`.

A directory management utility is necessary when your system becomes more complex with many guests and minidisks because manual directory management can become cumbersome.

3.7.2 Virtual disks

Virtual disks, which are defined as VDisk on CP guest directory, are commonly used for Linux on Z guests to be used for swap area because it is much faster than common minidisks and does not need to be persistent across system recycles. If you use such a facility, consider the use of z/VM page volume encryption because virtual disks are stored in z/VM page volume area in clear text.

3.7.3 System configuration file

Make sure to include a CRYPTO statement in the SYSTEM CONFIG file to select the current crypto resources for share use by the guests. If it is not specified, CP can select by default an incorrect set of crypto resources to create the shared crypto resources pool and a system REIPL is needed to correct the configuration.

Shared crypto resources often are selected by CP from the current total of crypto domains and adapters that are available for acceleration purposes. They also are shared among all guests with the CRYPTO APVIRT specified in their CP directory entries.

When selecting crypto resources for shared purposes, always consider that they are removed from the total amount of crypto resources that are available and can be dedicated to Linux on Z guests. In a z14 for example, the total number of domains per Crypto Express card is 85. Considering that a z14 can include up to 16 Crypto Express cards, a total of 1,360 domains or APs are available for shared and dedicated functions. Check your IBM Z platform specifications for the supported number of Crypto Express cards and domains.

As best practice, at least two domains or APs from different Crypto Express cards must be dedicated to each Linux on Z guest for redundancy purposes. For example, instead of having 1,360 domains or APs available, this number is reduced to 680 and another set of domains or APs must be subtracted from this number to create the crypto resources shared pool if needed.

3.7.4 z/VM SSI cluster considerations

A z/VM single system image (SSI) cluster is a multisystem environment in which the z/VM systems can be managed as a single resource pool and guests can be moved from one system to another while they are running. z/VM SSI cluster consists of several interconnected z/VM systems and each z/VM system is a member of the SSI cluster.

An SSI cluster provides a virtual server mobility function that is called *Live Guest Relocation* (LGR). A running virtual server (guest virtual machine) can be relocated from one member to another. Relocating guests can be useful for load balancing and for moving workload off a physical server or member system that requires maintenance.

LGR is a powerful tool that can be used to manage maintenance windows, balance workloads, or perform other operations that might otherwise disrupt logged-on guests. For example, LGR can be used to allow critical Linux servers to continue to run their applications during planned system outages. LGR can also enable workload balancing across systems in an SSI cluster without scheduling outages for Linux virtual servers.

A z/VM guest must satisfy several conditions to be eligible for LGR, including not using a CRYPTO APDEDICATE statement in the guest CP directory definition. Meeting this condition is a prerequisite to having data at-rest encryption services running on the guest (such guests are not eligible for LGR).

If the Linux on Z data at-rest encryption guest must be relocated from one z/VM member to another, it must shut down in the member where it is running and then, be rebooted in another member. This process is disruptive from the guest perspective.

For more information about conditions that prevent live guest relocations, see *z/VM: CP Planning and Administration*, SC24-6271.

For more information, see this web page of IBM Knowledge Center.

3.8 Performance considerations

The performance and CPU usage can be greatly enhanced on z13 and z14 by using 4096 bytes instead of the default 512 bytes for the sector size of the encrypted volumes. A significant difference also exists between the use of a z13 versus a z14.

Important: The examples that are described in this section are based on preliminary internal IBM lab measurements on a stand-alone, dedicated system in a controlled environment. Actual performance results can vary, depending on specific configuration, operating conditions, and workloads.

The measurements that are shown in Figure 3-3, Figure 3-4 on page 40, Figure 3-5 on page 40, and Figure 3-6 on page 41 use read/write throughput without filesystem caching effects. Standard filesystem usage with caching often render better results.

In this section, we show the difference in throughput and CPU usage when sequential read with a single thread is used. The z13 is compared with the z14 by using no encryption, clear key encryption, and protected key encryption.

As shown in Figure 3-3, the throughput decreases by 27% when protected key encryption is used with a 512-byte sector size.

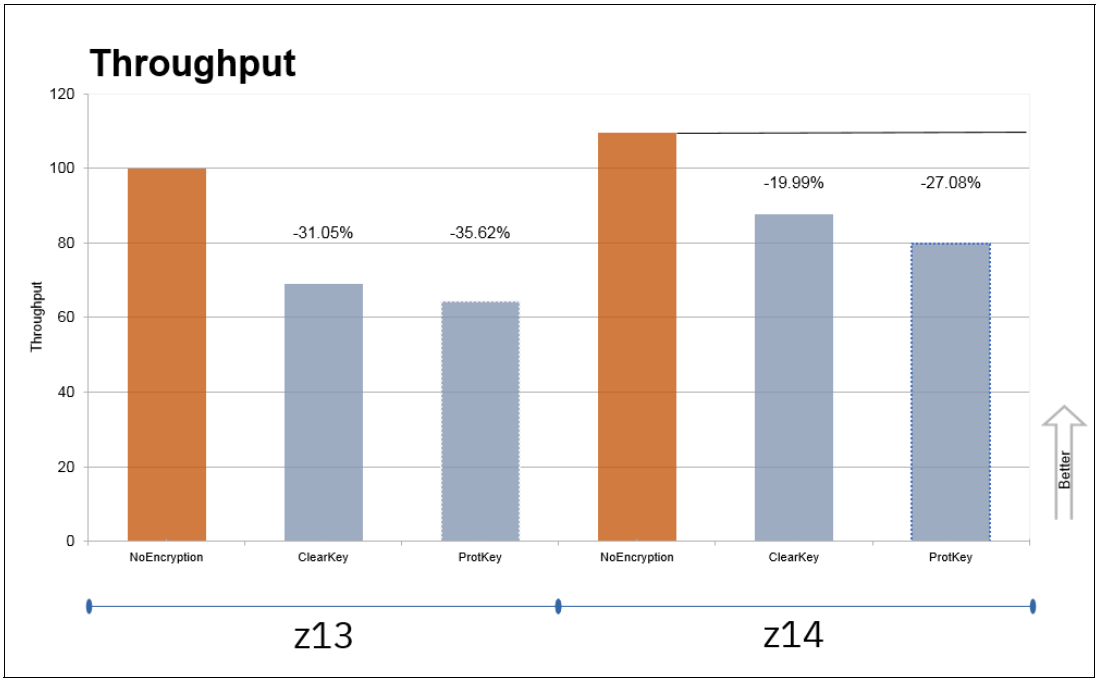

Figure 3-3 Throughput comparison when 512 bytes is used as sector size

With the z13, throughput decreases even further, as shown in Figure 3-4 on page 40. The use of clear key encryption can help somewhat, but not much.

As seen in Figure 3-4 on page 40, the throughput with a 4096-byte sector size decreases only by 9.57% on a z14 and by 20% an z13. Also, no real difference is seen between protected key and clear key operations.

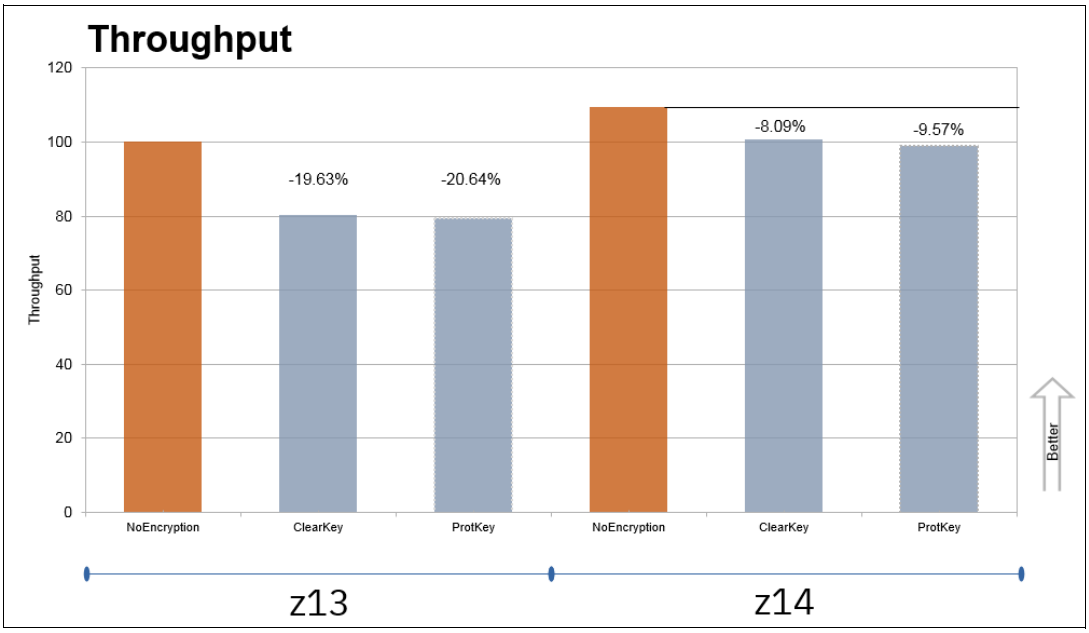

Figure 3-4 Throughput Comparison when 4096 bytes is used as sector size

This difference shows also with CPU usage when 512 bytes is used as a sector size (see Figure 3-5). Between the use of encryption and encryption not used, a difference of 6.3% when protected key is used and 4.6% when clear key is used.

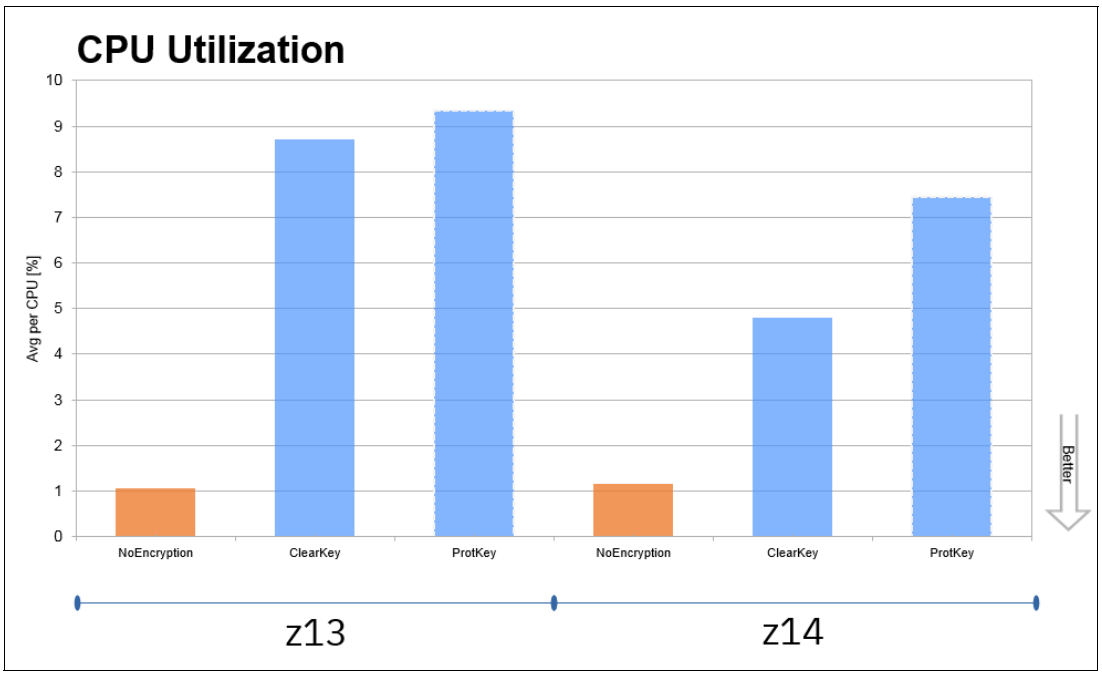

Figure 3-5 CPU utilization when using 512 bytes as sector size

With a 4096-byte sector size, only a delta of 1.9% results when protected key is used, and a delta of 1.7% when clear key is used, as shown in Figure 3-6. The delta between clear and protected key is negligible when a sector size of 4096 bytes is used.

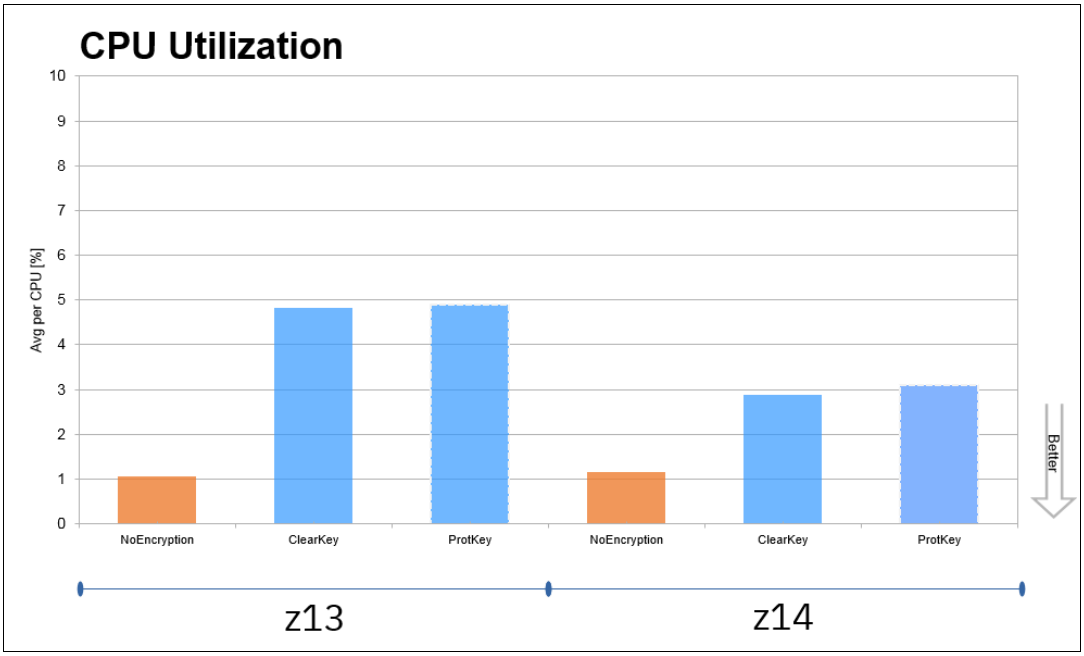

Figure 3-6 CPU utilization when using 4096 bytes as sector size

Therefore, it is recommended to use a z14 for data at-rest encryption. Also, use 4096 bytes as the sector size of the encrypted volumes when the software requirements for 4096-byte sector sizes are fulfilled.

For more information about Linux on Z encryption performance data, see this web page.

For more information, see 2.2.1, “Central Processor Assist for Cryptographic Function” on page 15.

3.9 Disk and logical volume considerations

On IBM Z platforms, ECKD™ and SCSI devices are most commonly used. Linux on Z supports both types of devices. The only real difference between them when encrypting data at-rest is that with ECKD devices, only partitions can be encrypted; with SCSI devices, the entire disk can be encrypted.

3.9.1 Using logical volumes

Although not required with Linux on Z data at-rest encryption, the use of the Logical Volume Manager (LVM) makes it easy to restructure your volume pool later on. With LVM, two views on the volumes are available. The physical view is where the raw disks are visible and the logical view is where all disks are restructured into a set of volumes (also called a *volume group*). It is good practice to have all disks that use the same secure key in the same volume group.

In this volume group, logical volumes can be freely created, resized, and removed. This ability allows a flexible handling when changes to the setup must be made or more space must be made available for an application. Figure 3-7 shows an example of three physical disks that are set up with LUKS2 format and used as physical volumes in the configuration.

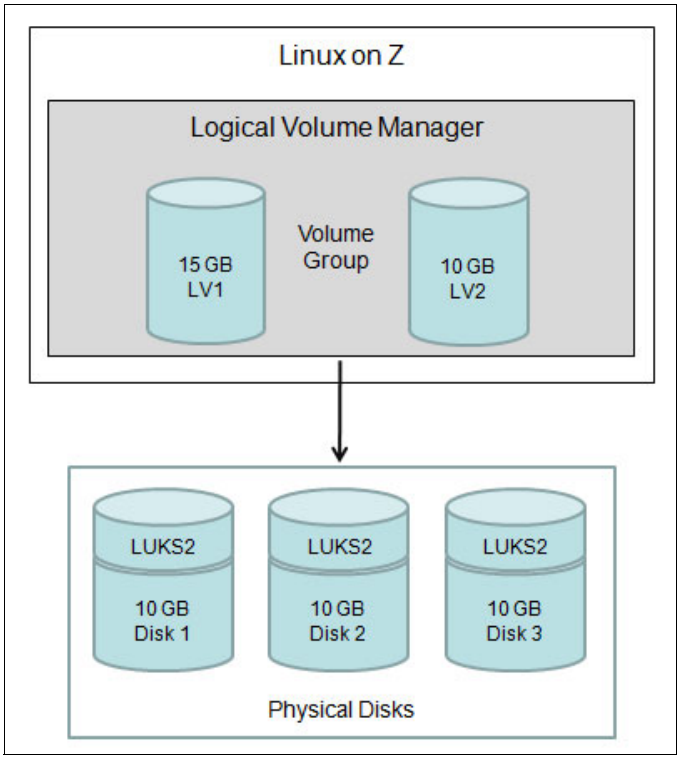

Figure 3-7 Example logical volume setup

For this configuration, the volume group consists of three physical volumes. The two logical volumes consist of blocks from the volume group. The LVM can use all associated physical volume space to create arbitrary logical volumes of arbitrary sizes (capped at the sum of the size of all physical volumes combined) in the volume group.

LVMs can also freely resize the logical volumes to adapt the volume group to higher resource requirements. In our example, 5 GB of storage is not used in any logical volume. When one of the two logical volumes run out of space, the LVM administrator can resize the volume with the unused space.

Important: Moving logical volumes (LVs) between physical volumes (PVs) corrupts filesystems if the physical block size of the target PVs is larger than the physical block size of the source PVs. This issue occurs because filesystems are aligned to the physical block size of the source PVs.

Also, moving LVs from unencrypted PVs to dm-crypt encrypted PVs can corrupt filesystems if the dm-crypt devices use different sector sizes.

Do not extend an LVM volume group with a device that has a larger physical block size than the PVs of the volume group.

To query the physical block size of a device, use the **`blockdev --getpbsz <device>`** command.

For more information, see this website.

3.9.2 Which volumes should be encrypted?

All volumes that host data should be encrypted. As discussed in 3.8, “Performance considerations” on page 38, the effect of encryption is minimal. With LVM and the use of keyfiles, it is also possible to use data at-rest encryption without much administrative overhead.

Linux on Z volume backups

Backing up encrypted data must be done in a secure way. The volumes that are hosting the backup should also be encrypted. Not encrypting the backup volumes defeats the purpose of Linux on Z data at-rest.

3.9.3 Volume format considerations

The important differences between the two volume formats that support Linux on Z data at-rest encryption and the use of `paes` cipher are listed in Table 3-2.

Table 3-2 Important differences between plain and LUKS2 formats

Plain format	LUKS2 format
No header; therefore, no formatting of the volume is required. Opening the volume creates it. Linux cannot detect the volume without attempting to open it.	One-time formatting is required, which creates a flexible JSON header for metadata. Linux can recognize the volume type.
Metadata of the volume must be supplied with every open. Incorrectly supplied metadata can destroy the device.	Metadata is redundantly saved in header and can be auto-repaired with the copy when a corruption is detected.
No passphrase or keyfile support.	Up to 32 passphrases or keyfiles are supported. Secure key is wrapped with a key that is derived from the passphrases or keyfiles. Can use the Argon2 key derivation function for enhanced security.
Volume cannot be reencrypted. Data must be moved out and in again after it was opened with a different secure key (opening it with a different key creates a volume).	The volume can be safely reencrypted.

Note: LUKS1 format does not support the `paes` cipher and therefore cannot perform protected key operations. The use of secure and protected keys is the preferred method for Linux on Z data at-rest encryption.

The plain format is the least complex mode of the two modes because no formatting is required and opening the volume also creates it. However, it does not protect against the use of the incorrect secure key to open it (instead, it creates a volume with the new secure key, which can destroy the old volume). The risk can be minimized by using **`zkey`** commands, which generate the correct cmdline arguments to open the volume.

Therefore, for new projects we suggest you consider the LUKS2 format. It also supports enhanced security that uses the Argon2 key derivation function. LUKS2 format is easier to use because the parameters do not need to be supplied by every open on a volume and can have different passphrases or keyfiles.

> **Note:** Argon2 uses a fixed amount of memory for its computations. This limit can cause out-of-memory errors when more than one volume is unlocking; for example, on start of Linux when several volumes are unlocked at the same time.
>
> For more information, see this page of IBM Knowledge Center.

LUKS2 format volumes are also recognizable by the operating system, whereas plain encrypted volumes resemble random data. LUKS2 format volumes can also be re-encrypted with another secure key. Re-encryption is not possible in plain mode.

3.10 Data compression considerations

Linux on Z can take advantage of the zEDC Express feature to perform data compression by using hardware acceleration instead of software compression. zEDC Express is available for zEC12 and later IBM Z platforms.

If you are considering hardware compression, a Linux Generic Work Queue Engine (GenWQE) driver is required. GenWQE supports hardware-accelerated data compression and decompression that uses the common lib API standards.

Because encrypted data does not compress well, any compression that occurs after encryption is ineffective. Therefore, consider the use of compression when encrypted volumes are created.

For more information about the GenWQE driver, see *Linux on Z and LinuxONE, Device Drivers, Features, and Commands*, SC33-8411.

3.11 General considerations

This section provides information about performing health checks and maintaining your data at-rest encryption environment. It also includes steps for backing out of encryption if it becomes necessary.

3.11.1 Defining a maintenance policy

A robust corrective and preventive maintenance policy is one of the best ways to ensure your operating system and all associated products (including hardware, firmware, and operating system) are as stable and securable as possible. Resolving known defects quickly helps deliver a platform where any new issues can be resolved more quickly.

IBM flags fixes in numerous categories, including High Impact PERvasive (HIPER), Program temporary fix in Error (PE), and Pervasive. Security Vulnerability (SECINT), which are of particular importance, is a classification (SOURCEID) of vulnerability PTFs that are related to Common Vulnerability Scoring System (CVSS). The Linux distributions also use CVSS to score their security vulnerabilities. For more information, see your Linux distribution.

Security and Integrity Vulnerability APARs address problems that are associated with potential unauthorized access or potentially compromised system controls. Because of the highly sensitive nature of any such identified defects, the content is classified as “IBM Confidential” and access is restricted to those APARs. Access is permitted to authorized customers through the IBM z Systems® Security Portal.

Access to the Portal can be requested by using the Systems integrity page of the IBM Z website (terms and conditions apply).

3.11.2 Linux Health Checker considerations

It can be time-consuming in large environments to check dozens of settings on every Linux guest. One way to speed up this process is the use of an open source package that is named the Linux Heath Checker. This package is an IBM sponsored project that focuses on Linux on Z, but is fully extensible with the ability to write your own health checks and create profiles that can match a set of applicable checks for the customer environment.

For more information, see 6.5.4, “Linux Health Checker tool” on page 78.

3.11.3 Backing out of Linux on Z data at-rest encryption

Part of any implementation plan is the preparation for backing out, if required. It is recommended to plan for a simple or gradual implementation of Linux on Z data at-rest encryption so that backout is straightforward and easy.

If the process is followed and you have the basic knowledge of your encryption criteria, the easiest way to backout is to copy the data on your encrypted volumes to non-encrypted volume. If the implementation requires a backout for all encrypted volumes, this process must be done for each encrypted volume.

It is also possible to back out by using an in-place decrypting tool. However, this process also involves backing out all of your data because it is a destructive process that can fail, which can result in losing your volume.

For more information, see this website.

4

Preparing for Linux on Z data at-rest encryption

This chapter provides information about what must be verified and configured on the HMC/SE, in z/VM, and in Linux on Z to support the deployment of Linux on Z data at-rest encryption; that is, guests that are encrypting volumes by using secure and protected keys.

We also cover creating and loading the master key in the guest Crypto Express adapter domain. This step is a prerequisite to any data at-rest encryption deployment tasks.

For more information about hardware and software components, see Chapter 2, “Identifying components and release levels” on page 13.

This chapter includes the following topics:

4.1 Hardware resources availability

Linux on Z data at-rest encryption needs the following hardware resources:

- CPACF
- Crypto Express adapters and domains

In this section, we describe the steps that are necessary to verify whether these resources are available and if they are not, how to install and configure them by using the HMC/SE.

4.1.1 CPACF

CPACF offers a set of cryptographic functions that enhance the encryption and decryption performance of clear and protected key operations and must be installed on the CPC.

Checking CPACF installation

CPACF installation can be verified by completing the following steps on HMC/SE (see Figure 4-1):

1. Expand the **System Management** link in navigation pane.
2. Click **CPC Details**.

   If the message “CP Assist for Crypto functions: Installed” is shown in the Instance Information tab, the CPACF is installed (as highlighted by number 3 in Figure 4-1).

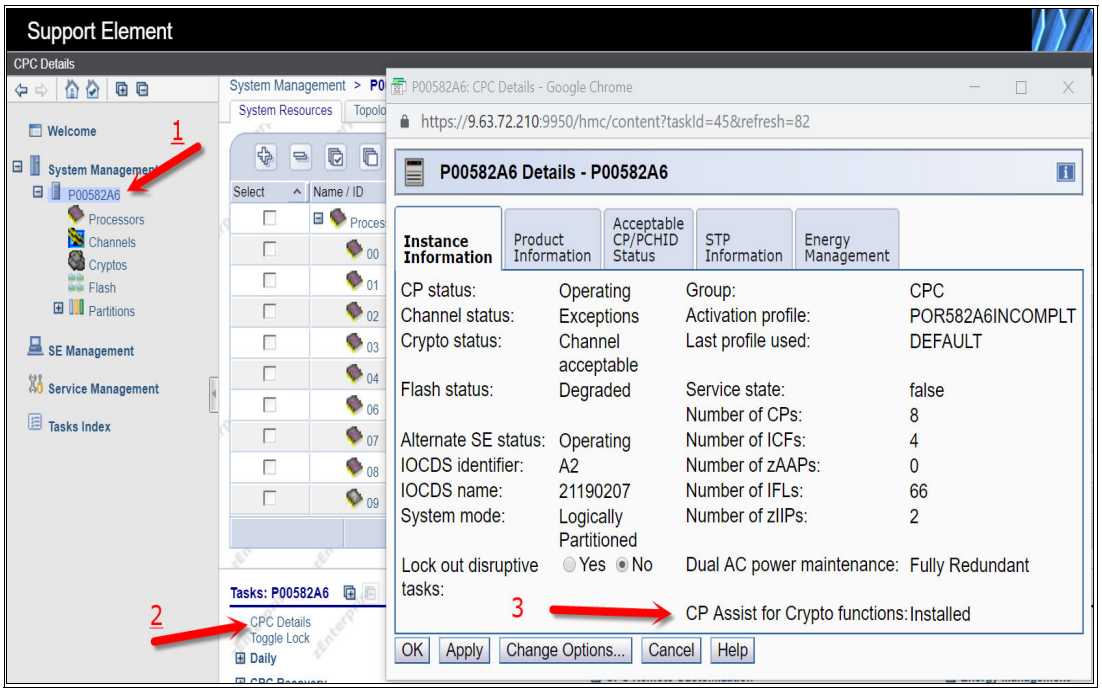

Figure 4-1 Checking CPACF installation on Support Element

Note: If CPACF is not installed, the no-charge CPACF feature (FC 3863) must be ordered and installed in the CPC.

4.1.2 Crypto Express adapters and domains

Linux on Z data at-rest encryption needs Crypto Express adapters to hold the master keys, which are used to create secure and protected keys. The adapters are virtualized in domains that must be configured to use CCA architecture and assigned to the LPAR by using the HMC/SE.

By default, Crypto Express adapters are installed in the IBM Z CPC as CCA coprocessors.

> **Note:** A minimum of two Crypto Express adapters is suggested for high availability purposes. In this case, the adapters must be loaded with the same master keys.

In this section, we describe the steps to verify which Crypto Express adapters and domains are available for the LPAR and how to add new domains, if needed.

Verifying current Crypto Express availability

Complete the following steps to identify which Crypto Express domains and adapters are available for an LPAR that uses HMC windows, as shown in Figure 4-2 on page 49:

1. Log on to the HMC.
2. Select **System Management**.
3. Select the CPC where the LPAR is running.
4. In the Partitions tab, select the LPAR where z/VM is running.
5. Expand **Operational Customization**.
6. Select **Customize/Delete Activation Profiles**.

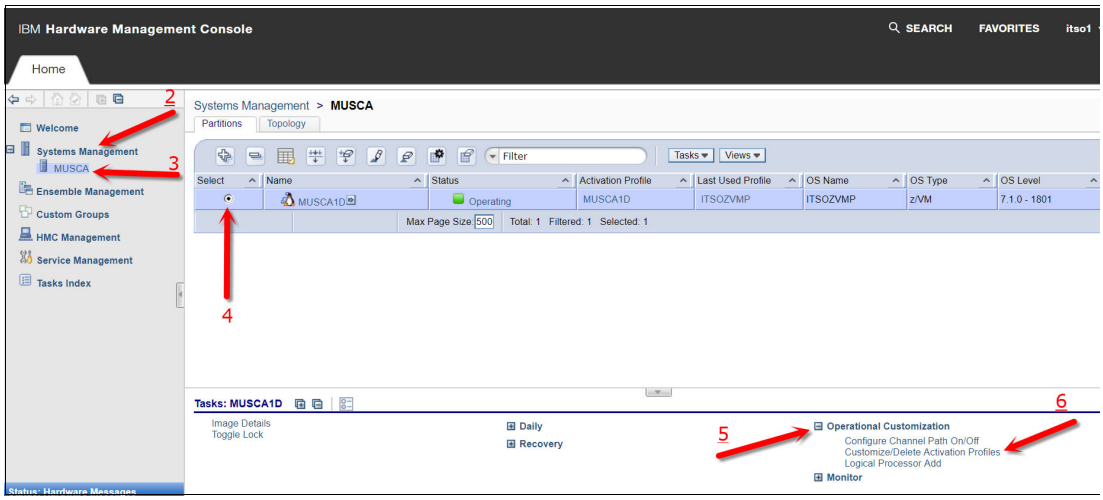

Figure 4-2 Accessing LPAR activation profile

7. Select the correct image profile (see Figure 4-3 on page 50).

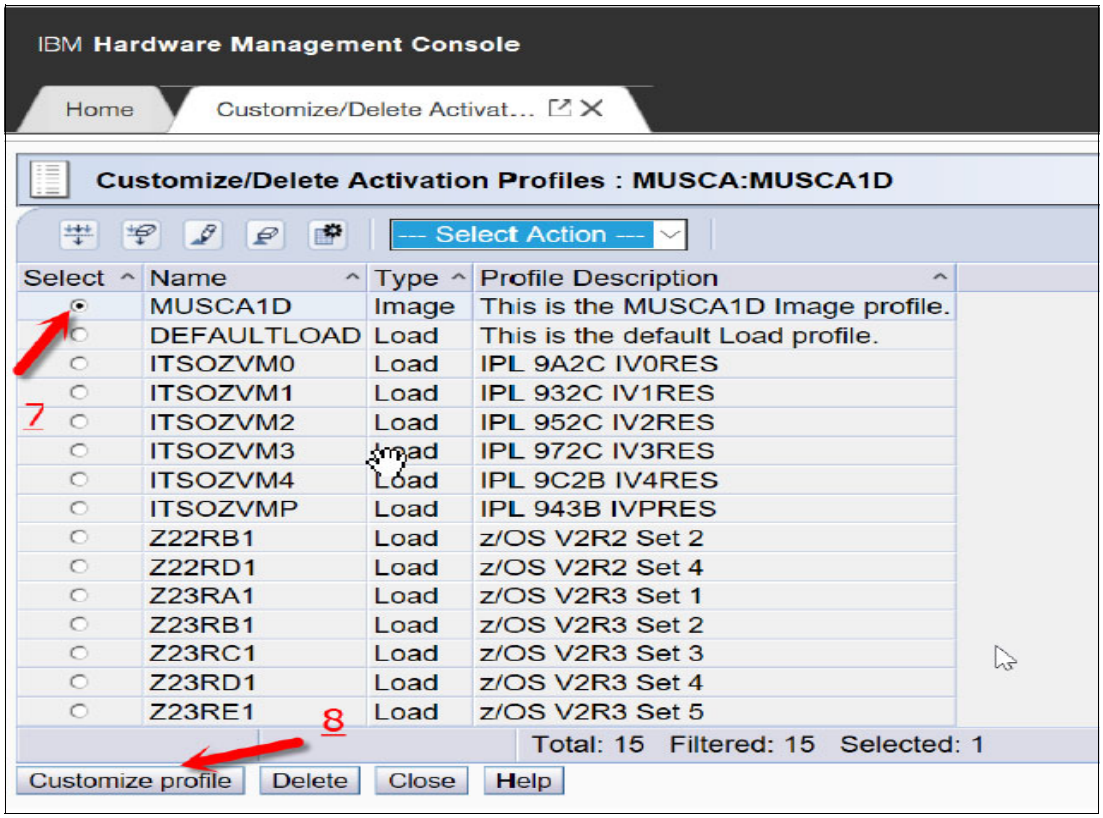

Figure 4-3 Accessing LPAR image profile

8. Click **Customize profile** (see Figure 4-3).
9. Select **Crypto** (see Figure 4-4).

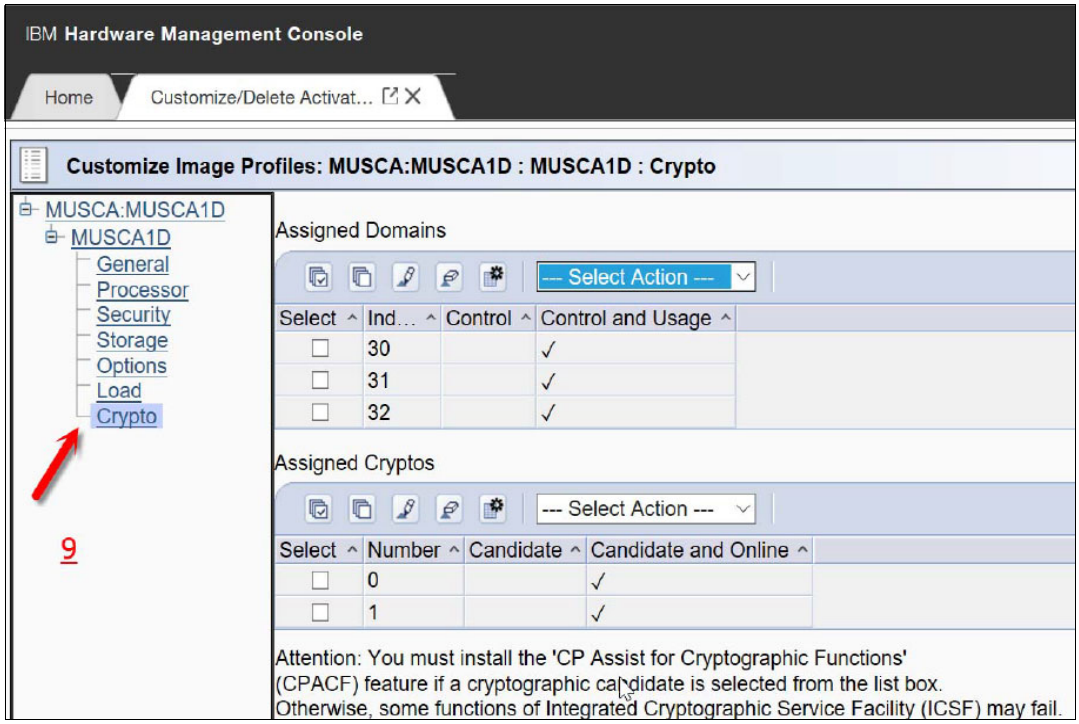

Figure 4-4 Available Crypto Express adapters and domains

The domains and cryptos that are available to the LPAR are displayed in the Assigned Domains area.

Adding Crypto Express adapters and domains to an LPAR

Use HMC/SE to change Crypto Express adapters and domains that are available to the LPAR. For more information about configuring Crypto Express adapters, see *z13 Crypto - Setting up an LPAR to use crypto*. (The instructions that are shown in this document are for the z13, but are similar to those on other IBM Z platforms.)

4.2 z/VM configuration

This section shows the necessary tasks to prepare a z/VM LPAR for use with Linux on Z data at-rest encryption.

Because Linux on Z running under z/VM can have data in memory that is being paged out to z/VM paging volumes in plain text, we also show how to configure z/VM encrypted paging volumes.

Linux on Z data at-rest must have dedicated access to a Crypto Express domain. Therefore, we describe how z/VM is configured to enable this access.

4.2.1 CPACF and z/VM

CPACF is used by z/VM encrypted paging and must be enabled before any encryption is configured.

Checking CPACF availability for z/VM

Use the **CP QUERY CRYPTO** command to verify that your output is similar to the output that is shown in Example 4-1.

Example 4-1 QUERY CRYPTO command with CPACF enabled

```
Q CRYPTO
Crypto Adjunct Processor Instructions are installed
Ready; T=0.01/0.01 22:16:23
```

The output shows the status of the cryptographic hardware and the AP (AdjunctProcessor) of the Crypto Express adapter. The Crypto Adjunct Processor Instructions do not show as “installed” if CPACF is not enabled; therefore, the no-charge CPACF feature (FC 3863) must be ordered.

No explicit z/VM authorization, configuration, or definitions are required for accessing CPACF functions from z/VM.

4.2.2 z/VM paging volume encryption

In this section, we describe how to dynamically enable and disable z/VM encrypted paging and query the status.

Encrypted paging improves z/VM security by using IBM Z hardware to encrypt guest page data. Ciphering occurs as data moves from active memory onto a paging volume, such as ECKD and SCSI devices.

Enabling z/VM encrypted paging

Example 4-2 shows the output of the **SET ENCRYPT** command to dynamically enable z/VM encrypted paging.

Example 4-2 Enabling z/VM encrypted paging

```
set encrypt paging on
Encrypt Paging set on to algorithm AES256
Encrypt Paging Settings:
     Currently: On AES256
     At IPL: Off
Ready; T=0.01/0.01 12:56:22
```

Disabling z/VM encrypted paging

z/VM encrypted paging can dynamically be disabled, as shown in Example 4-3.

Example 4-3 Disabling z/VM encrypted paging

```
set encrypt paging off
Encrypt Paging set off
Encrypt Paging Settings:
     Currently: Off
     At IPL: Off
Ready; T=0.01/0.01 13:00:39
```

Querying z/VM encrypted paging

z/VM encrypted paging status can be queried, as shown in Example 4-4.

Example 4-4 Querying z/VM encrypted paging status

```
query encrypt
Encrypt Paging Settings:
     Currently: Off
     At IPL: Off
Ready; T=0.01/0.01 13:01:56
```

Activating paging volume encryption at IPL time

The encryption can be activated at IPL by including the `ENCRYPT` statement in the system configuration file.

For more information about z/VM encrypted paging, see *z/VM CP Planning and Administration*, SC24-6271.

4.2.3 Crypto Express adapters and domains

Linux on Z that is running as a z/VM guest needs a dedicated Crypto Express domain and adapter to access key handling functions. In this section, we describe how to query z/VM to select a correct domain or adapter to be dedicated to the guest.

Identifying an available CCA domain and adapter

Complete the following steps:

1. Query which crypto domains are available by using the **CP QUERY CRYPTO AP** command, as shown in Example 4-5.

Example 4-5 QUERY CRYPTO AP showing domains available

```
q crypto ap
AP 000 CEX6C  Domain 030 available    shared                unspecified
AP 000 CEX6C  Domain 031 available    dedicated to ITSOUBUP dedication
AP 000 CEX6C  Domain 032 available    dedicated to ITSOREDP dedication
AP 001 CEX6C  Domain 030 available    shared                unspecified
AP 001 CEX6C  Domain 031 available    dedicated to ITSOSLEP dedication
AP 001 CEX6C  Domain 032 available    free                  unspecified
Ready; T=0.01/0.01 17:28:40
```

2. Select a line where the `system usage` column (seventh column from the left) indicates `free` and the `aptype` column (third column from the left) ends with the `C` character, which indicates it is configured to support the CCA architecture.

> **Note:** If you can access the HMC/SE (Support Element), you can add new free domains or adapters to the LPAR. For more information, see *z13 Crypto - Setting up an LPAR to use crypto.*

Dedicating a domain or adapter to a guest

Complete the following steps:

1. Add a `CRYPTO` statement in the z/VM guest user directory with `APDEDICATED` and `DOMAIN` operands. Example 4-6 shows dedicating domain 032 from adapter 001 to guest ITSOSLEP.

Example 4-6 Guest CP directory with dedicated domain/AP in CRYPTO directory statement

```
USER ITSOSLEP LNX4ITSO 16G 16G BG
  INCLUDE IBMDFLT
  IPL CMS
  MACHINE ESA 4
  OPTION CHPIDV ONE
  CPU 00 BASE
  CPU 01
  CRYPTO DOMAIN 032 APDEDICATED 001
  NICDEF 0600 TYPE QDIO LAN SYSTEM VSWITCH1
  NICDEF 0620 TYPE QDIO LAN SYSTEM VSWITCH2
  NICDEF 0640 TYPE QDIO LAN SYSTEM VSWITCH3
  MDISK 191 3390 0021 0010 IVPL01 MR READ WRITE MULTI
  MDISK 9CCB 3390 DEVNO 9CCB MWV ALL ALL ALL 0X9CCB #1
  MDISK 9D4B 3390 DEVNO 9D4B MWV ALL ALL ALL 0X9D4B #2
  MDISK FF01 3390 7059 0500 IVPCOM MR READ WRITE MULTI
  MDISK FF02 3390 7559 0500 IVPCOM MR READ WRITE MULTI
```

2. Update CP online directory with DIRECTXA utility.
3. Recycle z/VM guest to receive the update.

4.3 Preparing Linux on Z

In this section, we describe how to verify whether CPACF and Crypto Express adapter resources are available by using Linux commands.

CPACF availability

Show the features line from `/proc/cpuinfo` contents, see Example 4-7. If `msa`[1] is listed, CPACF is available to the guest.

Example 4-7 CPACF installation verification

```
# cat /proc/cpuinfo | grep features
features        : esan3 zarch stfle msa ldisp eimm dfp edat etf3eh highgprs te vx
#
```

Crypto Express adapter availability

Use the **`VMCP QUERY VIRTUAL CRYPTO`** command to identify which domain and adapter are dedicated to the guest, as shown in Example 4-8.

Example 4-8 Query Crypto Express from Linux on Z perspective

```
# vmcp query virtual crypto
AP 000 CEX6C Domain 032 dedicated
#
```

Verify whether the domain and adapter is online by using the **`lszcrypt`** command, as shown in Example 4-9. Notice that in this case, our Linux on Z guest is connected to a Crypto Express adapter 00 with domain 0020, or 00.0020.

> **Note:** The domain is represented in hexadecimal format with Linux on Z; a decimal format is used in z/VM. In this case, domain 0020 is domain 32 in z/VM.

Example 4-9 Verifying Crypto Express device driver

```
# lszcrypt
CARD.DOMAIN TYPE  MODE        STATUS  REQUEST_CNT
-----------------------------------------------
00          CEX6C CCA-Coproc  online            4
00.0020     CEX6C CCA-Coproc  online            4
#
```

[1] Message Security Assist (MSA) is the CPACF feature.

4.4 Loading the master key

The master key must be loaded into the Crypto Express domain before any data at-rest encryption service deployment. The CCA package can be used to load the keys if the adapter allows such a connection; otherwise, the configuration keys must be loaded by using a TKE workstation.

The `panel.exe` tool can be used by a member of the `cca_admin` group or `root` to load and set master keys for Crypto Express adapters. To enable separation of duty between the master key-part owner, different users can be used to load the master key-parts. For more information about setting up users, see IBM Knowledge Center.

> **Note:** Master key-parts in the clear must be provided to the `panel.exe` tool in a Linux on Z console. Ensure that no single administrator can see all master key-parts. Such visibility can compromise part or the entire master key.

Example 4-10 shows the call to the `panel.exe` and the first menu. It also shows the selection of different master keys that can be loaded. For Linux on Z data at-rest encryption, only the AES master key is used. Therefore, select **3** and confirm your choices by selecting **y** in the confirmation dialog.

Example 4-10 Selecting the correct master key

```
# /opt/IBM/CCA/bin/panel.exe --mk-load-interactive
Preparing to LOAD master key part
Use Which Master Key(MK)?
    Type a number (no spaces; use 'q' to quit) and press Enter:
        1: SYM-MK
            (used to encipher DES (symmetric) key objects)
        2: ASYM-MK
            (used to encipher PKA-RSA (asymmetric) key objects)
        3: AES-MK
            (used to encipher AES,HMAC (symmetric) key objects)
        4: APKA-MK
            (used to encipher PKA-ECC (asymmetric) key objects)
```

The next dialog (see Example 4-11) you are prompted to select which part of the master key part is loaded. For the first part, select **1**; for the middle part, select **2**; and for the last part, select **3**. The user that enters the first part must be in the `cca_lfmkp` group. The user of the middle or last part must be in the `cca_cmkp` group. Confirm your selections in the dialog by selecting **y**.

Example 4-11 Selecting the master key part

```
Create/Enter Which Master Key(MK) Part?
    Type a number (no spaces; use 'q' to quit) and press Enter:
        1: FIRST
            you will be entering the FIRST  MK PART (32 Bytes)
        2: MIDDLE
            you will be entering the MIDDLE MK PART (32 Bytes)
        3: LAST
            you will be entering the LAST   MK PART (32 Bytes)
            next operation should be SET (or CLEAR)
```

After selecting which key part is loaded, the dialog that is shown (see Example 4-12) appears. Enter the 64-character long key part and press Enter. The tool counts the characters and reports if something is incorrect. Confirm the dialog by selecting **y**.

Example 4-12 Entering the master key part

```
Type a 64 character hex value
(2 text chars == 1 binary Byte) using [0-9A-Fa-f]
    (no other characters or spaces; use 'q' to quit)
     and press Enter:
```

Example 4-13 shows the return code for loading the key part. If a nonzero return code is shown, see this Reason code table at IBM Knowledge Center for more information.

Example 4-13 Confirmation message for master key load

```
LOAD for Master key [AES-MK  ] [FIRST   ] with KEY PART:
    [BC2261AC2A55CB6A0A13F40D1B81EB16AD1CF67CF88A1EE98FA3DF0D9D1C723A]
returned:
    Return Code [0] Reason Code [0]
```

After all three master key parts are loaded, secure keys must be reenciphered to the new master key. For more information, see Chapter 8, “Performing key management for data at-rest encryption” on page 87.

Example 4-14 shows the next `panel.exe` command, which must be entered when the new master key is used. Select **3** for the AES master key and confirm your choice by entering **y**.

Example 4-14 Setting the master key

```
# /opt/IBM/CCA/bin/panel.exe --mk-set-interactive
Preparing to SET master key
Use Which Master Key(MK)?
    Type a number (no spaces; use 'q' to quit) and press Enter:
        1: SYM-MK
            (used to encipher DES (symmetric) key objects)
        2: ASYM-MK
            (used to encipher PKA-RSA (asymmetric) key objects)
        3: AES-MK
            (used to encipher AES,HMAC (symmetric) key objects)
        4: APKA-MK
            (used to encipher PKA-ECC (asymmetric) key objects)
```

The new master key is now set and secure keys can be created, which are wrapped under this master key.

Note: If more than one domain is used with a Crypto Express adapter, the `panel.exe` tool uses the default domain that is defined in `/sys/bus/ap/ap_domain`. If another domain is selected, the environment variable `CSU_DEFAULT_DOMAIN` must be set to the correct value before the `panel.exe` tool is used.

5

Deploying encrypted volumes for data at-rest

This chapter provides step-by-step guidance for creating plain format and LUKS2 format volume types with the three supported Linux on Z distributions (Red Hat, SUSE, and Ubuntu).

We used the most currently available software; however, at the time of this writing, some of the tools and capabilities were not yet available for all Linux on Z distributions.

This chapter includes the following topics:

5.1 Our Linux on Z environment

The environment that we used is shown in Figure 5-1. It consisted of the following components:

- z14 ZR1: The latest entry-level z14 is a cloud- and analytics-focused machine with powerful security capabilities. For more information, see this website.
- CPACF: Provides high-speed encrypt and decrypt functions and facilitates the privacy of cryptographic key material when used for data encryption through a key wrapping implementation.
- Crypto Express6S Coprocessors: Crypto Express adapters are tamper-responding hardware security modules that provide cryptographic functions and configured in CCA mode.
- Two ECKD disks that are known to the Linux on Z guests as 0.0.ff01 (for plain format) and 0.0.ff02 (for LUKS2 format).
- z/VM: The z/VM 7.1 release is the latest release and provides enhanced security capabilities, such as page volume encryption.
- Linux on Z running as z/VM guests:
  - ITSOREDP: Red Hat Enterprise Linux Server 7 Service Pack 6 release, supports plain format encryption.
  - ITSOSLEP: SUSE Linux Enterprise Server 15 release, supports plain format encryption.
  - ITSOUBUP: Ubuntu Linux 18.10 non-LTS release, supports plain format and LUKS2 format encryption.

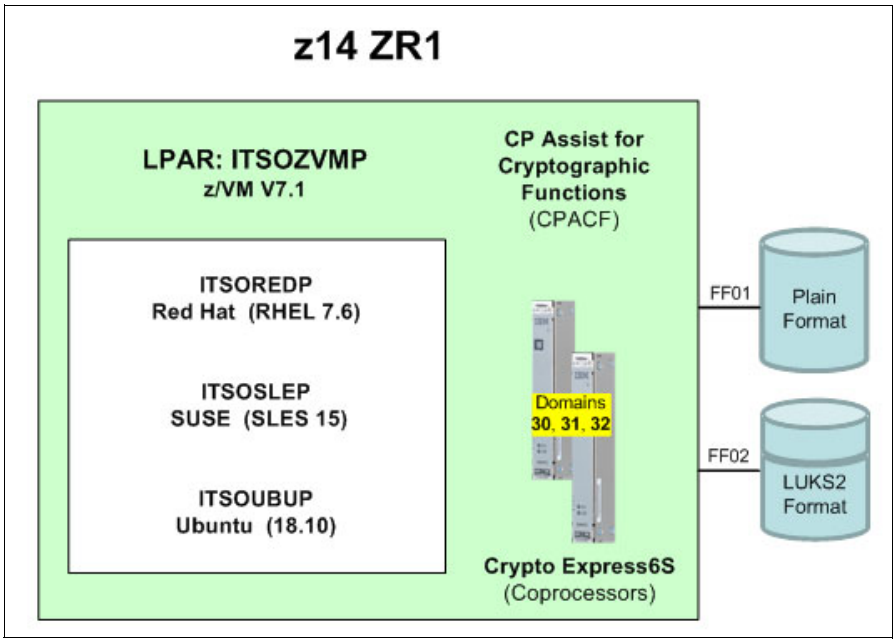

Figure 5-1 Our Linux on Z environment

5.2 Deployment readiness checklist

Table 5-1 provides a set of questions to help you determine whether your Linux on Z environment is ready for data at-rest deployment.

Table 5-1 Checklist for determining deployment for Linux on Z data at-rest encryption

Checklist item	Comments	More information
Have you installed the required hardware and software components and prerequisites?	Linux on Z data at-rest encryption is available on RHEL 7.6, SLES 15, and Ubuntu 18.10. z/VM encrypted paging is available on z/VM 6.4 and z/VM 7.1.	2.2, “Required and optional hardware features” on page 15 2.3, “Required and optional software features” on page 18
Have you determined which volumes are to be encrypted?	What data are you going to encrypt? Scope and scale?	3.1, “Creating an implementation plan” on page 22
Have you reviewed your security, audit, and compliance practices?	In support of Linux on Z data at-rest encryption, identify if any gaps exist in your current practices.	3.3, “Encryption considerations” on page 23
Have you considered the use of security tools for your IBM Z environment?	Ensure that your security policies and keys are managed and monitored.	2.2.3, “Trusted Key Entry workstation” on page 17 Chapter 6, “Auditing and monitoring the data at-rest environment” on page 71
Have you determined the owners of the new tasks for administering and maintaining encrypted volumes and the encryption keys?	Fine or course grained access controls can limit access to data content by personnel that can otherwise pose a possible exposure point.	3.5, “Policy considerations” on page 26 3.7.1, “Policy considerations” on page 36
Have you determined which key management tools you will use?	You can create master keys by using a TKE workstation, which includes smart cards and smart card readers. The `panel.exe` tool is a panel-driven key entry and management tool.	3.6.10, “Choosing key management tools” on page 33
Have you defined your key lifecycle management process?	Track changes to a key’s parameters during its lifecycle.	3.6, “Key management considerations” on page 29
Have you considered your backup and recovery planning scenarios?	The disaster plan includes a mirrored implementation of data at-rest encryption at the backup site with the appropriate master key, crypto domains, and keyfiles.	3.6.12, “Planning for disaster recovery” on page 36 3.11, “General considerations” on page 44
Have you considered what is required to fall back?	Any implementation requires a plan to fall back.	3.11.3, “Backing out of Linux on Z data at-rest encryption” on page 45
Have you planned for test scenarios and education for potential users?	This IBM Redbooks publication can be used as a reference for building test scenarios and education.	Review this entire publication.

5.3 Encryption process overview

Two different encryption processes can be applied. We review the overall steps of each process next. Encrypted volumes compose the Logical Volume Manager 2 (LVM2) physical volumes, which form a volume group and make the space available to logical volumes to use. The data is encrypted transparently when it is written to disk in each LUKS2 format volume separately.

For more information about LUKS2 format, see 3.9.3, “Volume format considerations” on page 43.

5.3.1 Plain format volume overview

For plain format, the **`cryptsetup`** command is used to create a volume with the `plain` mode option.

The process to create a plain format volume includes the following steps:

1. Obtain which domain is used by Linux on Z.
2. Load the `pkey` kernel module[1].
3. Create a secure key by using the **`zkey`** command and storing the key in zkey database.
4. Open a plain format volume by using **`cryptsetup plainOpen`** command.

   The first time the **`plainOpen`** command is used, it creates the volume. When it is used again, it opens the created volume. Ensure that you store the proper open parameters on `/etc/crypttab`.
5. Update `/etc/crypttab` to unlock the volume at system start.
6. Use the volume by creating a filesystem or LVM physical volume.

Figure 5-2 shows the flowchart for the steps to create the plain format volume.

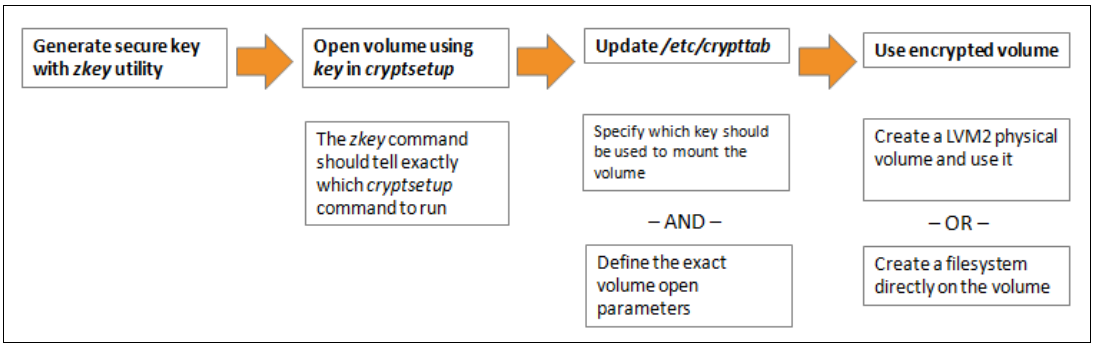

Figure 5-2 Plain format volume flowchart

5.3.2 LUKS2 format volume overview

When generating an LUKS2 format volume, a passphrase or keyfile is needed along with the secure key that is generated by the crypto facilities. The following process is used:

1. Obtain which domain is used by Linux.
2. Load the `pkey` kernel module[1].

[1] Before getting started, you must install the pkey kernel module for protected key management. You cannot use the zkey command until this prerequisite is met.

3. Use `zkey` utility to generate a secure key and store it in the zkey database.
4. Use `zkey` utility to generate the required commands for `cryptsetup`.
5. Run the commands that were generated in step 4 to format the volume and to insert the passphrase.
6. Open the LUKS2 volume by using the passphrase.
7. Generate a random keyfile. It is used as a second passphrase to open the volume.
8. Use `cryptsetup` to allow the keyfile to open the volume.
9. Update `/etc/crypttab` to allow automatic mounting of the volume during start.
10. Use the encrypted volume by creating a filesystem or LVM2 physical volume.

Figure 5-3 shows the flowchart of the steps for creating the LUKS2 format volume.

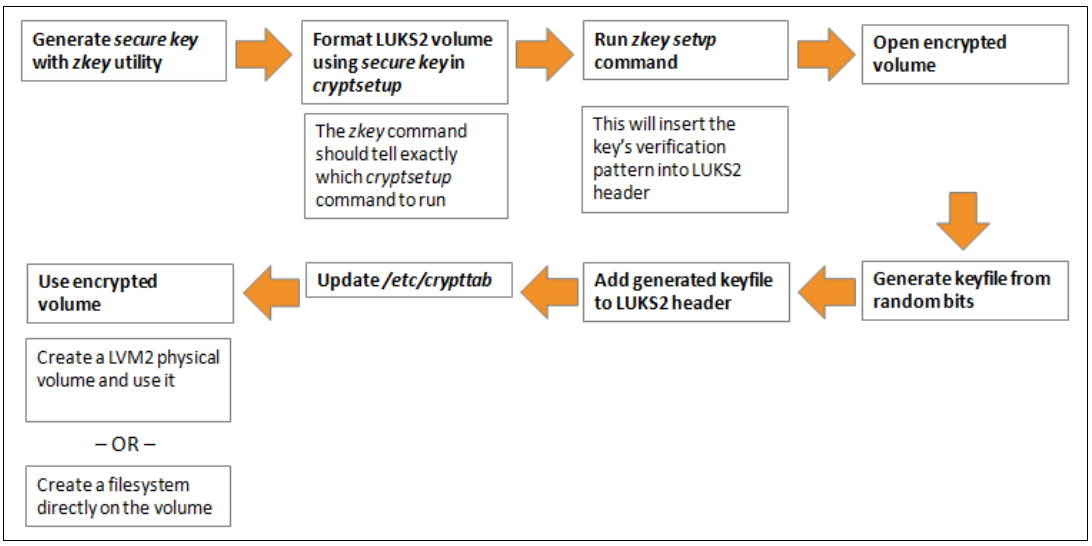

Figure 5-3 LUKS2 format volume flowchart

5.4 Red Hat setup

The current Red Hat supported version is Red Hat Enterprise Linux 7 Service Pack 6. This release includes back ported modules from newer kernel release, such as the `pkey` and `paes_s390` modules.

For the examples in this section, the ECKD disk (0.0.ff01) is used to create the plain format volume on the system.

5.4.1 Creating the plain format volume

Check which crypto domain is connected to Linux on Z by using the **`lszcrypt`** command. Notice that in this case, our Linux on Z guest is connected to a Crypto Express adapter 00 with domain 0020, or 00.0020, as shown in Figure 5-1 on page 62.

> **Note:** The domain is represented in hexadecimal format with Linux on Z; a decimal format is used in the HMC. In this case, domain 0020 is domain 32 in the HMC.

Example 5-1 Displaying Cards and Domains with lszcrypt command

```
# lszcrypt
CARD.DOMAIN TYPE  MODE        STATUS  REQUEST_CNT
-----------------------------------------------
00          CEX6C CCA-Coproc  online            4
00.0020     CEX6C CCA-Coproc  online            4
#
```

Ensure that the necessary kernel module is loaded by using the **modprobe zkey** command.

Now, generate a secure key by using the **zkey** command, passing as parameters the wanted name of the key, the key's bit length, the XTS algorithm to be used to encrypt data, and assigned volumes and APQNs, as shown in Example 5-2.

Example 5-2 Generating a Secure Key with zkey generate command

```
# zkey generate --name secure_xtskey1 --keybits 256 --xts --volumes
/dev/disk/by-path/ccw-0.0.ff01-part1:enc-disk1  --apqns 00.0020
#
```

List the details about the generated key by using the **zkey list -N secure_xtskey1** command, as shown in Example 5-3. It shows the keyfile that was created in Example 5-2, which is associated with the volume and APQN that we provided.

Example 5-3 Listing newly generated key with zkey list -N secure_xtskey1 command

```
# zkey list -N secure_xtskey1
Key                     : secure_xtskey1
--------------------------------------------------------------------------------
        Description     :
        Secure key size : 128 bytes
        Clear key size  : 512 bits
        XTS type key    : Yes
        Volumes         : /dev/disk/by-path/ccw-0.0.ff01-part1:enc-disk1
        APQNs           : 00.0020
        Key file name   : /etc/zkey/repository/secure_xtskey1.skey
        Sector size     : (system default)
        Created         : 2019-01-30 16:36:16
        Changed         : (never)
        Re-enciphered   : (never)

#
```

The **zkey cryptsetup** command is used to retrieve information from the zkey database (such as the key name, target volumes, and APQNs). The output of that command is a **cryptsetup** command to be used to create the volume with the correct parameters, as shown in Example 5-4 on page 63.

Issue the **zkey cryptsetup** to generate output for the **cryptsetup** command for your setup. Then, copy and paste the generated command and run it. If there is a need to automate this process, you can issue the **zkey cryptsetup** command with the **--run** parameter to immediately run the **cryptsetup** command without manual intervention.

Example 5-4 Using zkey cryptsetup command to generate a proper cryptsetup command

```
# zkey cryptsetup
cryptsetup plainOpen --key-file '/etc/zkey/repository/secure_xtskey1.skey'
--key-size 1024 --cipher paes-xts-plain64 /dev/disk/by-path/ccw-0.0.ff01-part1
enc-disk1
#
```

Device nodes are then created under `/dev` pseudo-filesystem, which refers to that volume. The devices that were created for this example are shown in Example 5-5.

Example 5-5 Symbolic links that are created in /dev

```
/dev/disk/by-id/dm-uuid-CRYPT-PLAIN-enc-disk1
/dev/disk/by-id/dm-name-enc-disk1
/dev/mapper/enc-disk1
```

With the volume now created, create an LVM2 physical volume to use as an LVM PV, as shown in Example 5-6.

Example 5-6 LVM physical volume creation on the encrypted volume

```
# pvcreate /dev/mapper/enc-disk1
  Physical volume "/dev/mapper/enc-disk1" successfully created.
#
```

Update `/etc/crypttab` by inserting the zkey-generated file that was used to open the volume during start. You can issue the **`zkey crypttab`** command to print the necessary lines for all zkey-managed volumes in the system. The output for the command is shown in Example 5-7.

Example 5-7 Entry in /etc/crypttab for the newly created volume

```
# zkey crypttab
enc-disk1 /dev/disk/by-path/ccw-0.0.ff01-part1 \
/etc/zkey/repository/secure_xtskey1.skey \
plain,cipher=paes-xts-plain64,size=1024,hash=plain
#
```

5.5 SUSE setup

For the example in this section, we used the following components:

- SUSE SUSE Linux Enterprise Server 15 (SLES 15)[2]
- ECKD disk (0.0.ff01) to create the plain format volume on the system

5.5.1 Creating a plain format volume

Check which crypto domain is connected to Linux on Z by using the **lszcrypt** command. Notice that in this case, our Linux on Z guest is connected to a Crypto Express adapter 00 with domain 0020, or 00.0020, as shown in Example 5-8.

Note: The domain is represented in hexadecimal format with Linux on Z; a decimal format is used in the HMC. In this case, domain 0020 is domain 32 in the HMC.

Example 5-8 Displaying Cards and Domains with lszcrypt command

```
# lszcrypt
CARD.DOMAIN TYPE  MODE        STATUS  REQUEST_CNT
-----------------------------------------------
00          CEX6C CCA-Coproc  online            4
00.0020     CEX6C CCA-Coproc  online            4
#
```

Ensure that the necessary kernel module is loaded by using the **modprobe zkey** command.

Then, create a directory that is named `secure_keys` to hold the keys. Ensure that only root can access that directory, as shown in Example 5-9.

Example 5-9 Creating directory to hold the secure keys

```
# mkdir /etc/secure_keys/
# chmod 700 /etc/secure_keys/
```

Generate a secure key by using the **zkey** command, as shown in Example 5-10.

Example 5-10 Generating a Secure Key with zkey command

```
# zkey generate /etc/secure_keys/xts-secure-key.sk --xts
#
```

Note: At the time of this writing, SUSE SLES 15 does not support a key repository; instead, the key is saved to the `/etc/secure_keys/xts-secure-key.sk` file. The path and file name can be changed, as needed.

After the secure key is created, use the `cryptsetup` program to create the plain format volume that uses the `paes-xts-plain64` cypher, as shown in Example 5-11.

Example 5-11 Creating a plain format volume with cryptsetup command

```
# cryptsetup plainOpen --key-file /etc/secure_keys/xts-secure-key.sk --key-size
1024 --cipher paes-xts-plain64 /dev/disk/by-path/ccw-0.0.ff01-part1 enc-disk1
```

[2] SUSE Linux Enterprise Server 12 SP4 also supports plain format volumes

Create device nodes are created under `/dev` pseudo-filesystem, which refers to that volume. The links that are created for our environment are shown in Example 5-12.

Example 5-12 Symbolic links that are created in /dev

```
/dev/disk/by-id/dm-uuid-CRYPT-PLAIN-enc-disk1
/dev/disk/by-id/dm-name-enc-disk1
/dev/mapper/enc-disk1
```

Use the created symlinks to create an LVM2 physical volume, as shown in Example 5-13.

Example 5-13 LVM physical volume creation on the encrypted volume

```
# pvcreate /dev/mapper/enc-disk1
  Physical volume "/dev/mapper/enc-disk1" successfully created.
#
```

Now you can create a volume group and logical volume by using the physical volume.

Update `/etc/crypttab` by inserting the zkey-generated that is used to open the volume at during start, as shown in Example 5-14.

Example 5-14 Entry in /etc/crypttab for the newly created volume

```
# cat /etc/crypttab
enc-disk1 /dev/disk/by-path/ccw-0.0.ff01-part1 \
/etc/secure_keys/xts-secure-key.sk \
plain,cipher=paes-xts-plain64,size=1024,hash=plain
#
```

5.6 Ubuntu setup

For Ubuntu, the currently supported version is Ubuntu Linux 18.04 LTS. The minimum required levels to work with LUKS2 format are not present in the 18.04 LTS version at the time of this writing; therefore, Ubuntu 18.10 (non-LTS) was used instead.

For the examples in this section, the following disks are used:

- ECKD disk (0.0.ff01) is used to create the plain format volume on the system
- ECKD disk (0.0.ff02) is used to create the LUKS2 format volume on the system

The subsequent sections discuss the steps that are needed to create a plain format volume and an LUKS2 format volume. Depending on the type of encryption format you chose, you must follow only the instructions that are described in “Creating a plain format volume” or “LUKS2 format volume creation” on page 67.

5.6.1 Creating a plain format volume

Check which crypto domain is connected to Linux on Z by using the **`lszcrypt`** command. Notice that in this case, our Linux on Z guest is connected to a Crypto Express adapter 00 with domain 001f, or 00.001f, as it is shown in Example 5-15 on page 66.

Note: The domain is represented in hexadecimal format with Linux on Z; a decimal format is used in the HMC. In this case, domain 001f is domain 31 in the HMC.

Example 5-15 Displaying Cards and Domains with lszcrypt command

```
# lszcrypt
CARD.DOMAIN TYPE  MODE        STATUS  REQUEST_CNT
-----------------------------------------------
00          CEX6C CCA-Coproc  online         8890
00.001f     CEX6C CCA-Coproc  online         8890
#
```

Ensure that the necessary kernel module is loaded by using the **modprobe zkey** command.

Generate a secure key by using the **zkey** command, as shown in Example 5-16.

Example 5-16 Generating a Secure Key with zkey command

```
# zkey generate --name secure_xtskey1 --keybits 256 --xts --volumes
/dev/disk/by-path/ccw-0.0.ff01-part1:enc-disk1 --volume-type PLAIN --apqns 00.001f
#
```

List the details about the generated key by using the **zkey list -N secure_xtskey1** command, as shown in Example 5-17.

Example 5-17 Listing newly generated key with zkey list -N secure_xtskey1 command

```
# zkey list -N secure_xtskey1
Key                         : secure_xtskey1
-------------------------------------------------------------------------------
        Description          :
        Secure key size      : 128 bytes
        Clear key size       : 512 bits
        XTS type key         : Yes
        Volumes              : /dev/disk/by-path/ccw-0.0.ff01-part1:enc-disk1
        APQNs                : 00.001f
        Key file name        : /etc/zkey/repository/secure_xtskey1.skey
        Sector size          : (system default)
        Volume type          : PLAIN
        Verification pattern : bed56ebabe456dcc4e1d0ec8d434ac17
                               3f5256087ffc532fb67c781e6eb66340
        Created              : 2019-01-30 15:59:15
        Changed              : (never)
        Re-enciphered        : (never)
#
```

The **zkey cryptsetup** command is used to retrieve information from the zkey database (such as the key name, target volumes, and APQNs). The output of that command is a **cryptsetup** command to be used to create the volume with the correct parameters, as shown in Example 5-18 on page 67.

Issue the **zkey cryptsetup** to generate output for the **cryptsetup** command for your setup. Then, copy and paste the generated command and run it. If there is a need to automate this process, you can issue the zkey **cryptsetup command** with the **--run** parameter to immediately run the **cryptsetup** command without manual intervention.

Example 5-18 Using zkey cryptsetup command to generate a proper cryptsetup command

```
# zkey cryptsetup --volumes /dev/disk/by-path/ccw-0.0.ff01-part1
cryptsetup plainOpen --key-file '/etc/zkey/repository/secure_xtskey1.skey'
--key-size 1024 --cipher paes-xts-plain64 /dev/disk/by-path/ccw-0.0.ff01-part1
enc-disk1
#
```

Create the physical volume, as shown in Example 5-19.

Example 5-19 LVM physical volume creation on the encrypted volume

```
# pvcreate /dev/mapper/enc-disk1
  Physical volume "/dev/mapper/enc-disk1" successfully created.
#
```

Update `/etc/crypttab` by inserting the zkey-generated that is used to open the volume during start. You can run the **`zkey crypttab`** command to print the necessary lines for all zkey-managed volumes in the system, as shown in Example 5-20.

Example 5-20 Entry in /etc/crypttab for the newly created volume

```
# zkey crypttab
enc-disk1 /dev/disk/by-path/ccw-0.0.ff01-part1 \
/etc/zkey/repository/secure_xtskey1.skey \
plain,cipher=paes-xts-plain64,size=1024,hash=plain
#
```

5.6.2 LUKS2 format volume creation

Generate a secure key by using the **`zkey`** command, as shown in Example 5-21.

Example 5-21 Key generation with zkey command

```
# zkey generate --name secure_xtskey1_luks --keybits 256 --xts --volumes
/dev/disk/by-path/ccw-0.0.ff02-part1:enc--luks2-disk1 --volume-type LUKS2 --apqns
00.001f --sector-size 4096
#
```

Note: Argon2 uses a fixed amount of memory for its computations. This limit can cause out-of-memory errors when more than one volume is unlocking; for example, on start of Linux when several volumes are unlocked at the same time.

For more information, see this page of IBM Knowledge Center.

List the details about the generated key by using the **`zkey list -N secure_xtskey1`** command, as shown in Example 5-22.

Example 5-22 Listing newly generated key with zkey list -N secure_xtskey1 command

```
# zkey list -N secure_xtskey1_luks
Key                           : secure_xtskey1_luks
--------------------------------------------------------------------------------
        Description           :
        Secure key size       : 128 bytes
        Clear key size        : 512 bits
```

```
        XTS type key         : Yes
        Volumes              :
/dev/disk/by-path/ccw-0.0.ff02-part1:enc-luks2-disk1
        APQNs                : 00.001f
        Key file name        : /etc/zkey/repository/secure_xtskey1_luks.skey
        Sector size          : 4096 bytes
        Volume type          : LUKS2
        Verification pattern : 936f463de407dd54b0edd92255c83367
                               dd86cfdd3fdb25c8d4518f99a6d687ee
        Created              : 2019-02-07 15:35:32
        Changed              : (never)
        Re-enciphered        : (never)
#
```

Use the **zkey** command to generate the required output to create the encrypted volume by using the **cryptsetup** command. It uses data from the zkey database to know which volume and what key are used together, as shown in Example 5-23.

Example 5-23 Generating cryptsetup commands with zkey-cryptsetup

```
# zkey cryptsetup --volumes /dev/disk/by-path/ccw-0.0.ff02-part1
cryptsetup luksFormat --type luks2 --master-key-file
'/etc/zkey/repository/secure_xtskey1_luks.skey' --key-size 1024 --cipher
paes-xts-plain64 --sector-size 4096 /dev/disk/by-path/ccw-0.0.ff02-part1
zkey-cryptsetup setvp /dev/disk/by-path/ccw-0.0.ff02-part1
#
```

Run the generated commands and provide a unique password for each volume. In this case, we use the **cryptvolff02** command as the password. This command formats the volume and creates an LUKS2 header that allows it to be mounted later. The process is shown in Example 5-24.

Example 5-24 Creating a volume with suggested cryptsetup command

```
# cryptsetup luksFormat --type luks2 --master-key-file
'/etc/zkey/repository/secure_xtskey1_luks.skey' --key-size 1024 --cipher
paes-xts-plain64 --sector-size 4096 /dev/disk/by-path/ccw-0.0.ff02-part1

WARNING!
========
This will overwrite data on /dev/disk/by-path/ccw-0.0.ff02-part1 irrevocably.

Are you sure? (Type uppercase yes): YES
Enter passphrase for /dev/disk/by-path/ccw-0.0.ff02-part1: cryptvolff02
Verify passphrase: cryptvolff02
#
```

Use the **zkey-cryptsetup setvp** command to set the verification pattern of the secure AES key into the LUKS2 header, as shown in Example 5-25.

Example 5-25 Setting the Verification Pattern with setvp parameter

```
# zkey-cryptsetup setvp /dev/disk/by-path/ccw-0.0.ff02-part1
Enter passphrase for '/dev/disk/by-path/ccw-0.0.ff02-part1': cryptvolff02
#
```

Open the LUKS2 format container by using the password that was created, as shown in Example 5-26.

Example 5-26 Opening the volume by using the created password

```
# cryptsetup luksOpen /dev/disk/by-path/ccw-0.0.ff02-part1 enc-luks2-disk1
Enter passphrase for /dev/disk/by-path/ccw-0.0.ff02-part1: cryptvolff02
#
```

Create a directory to store keys securely and modify its permissions so only root can use it. For more information about restricting access to the keys, see 3.5, “Policy considerations” on page 26.

Then, use the **dd** command to read from `urandom` pseudo device and generate a 4096-byte key, as shown in Example 5-27. This key is used as a secondary LUKS2 key to open the volume at system start.

Example 5-27 Creating directory and key from random source

```
# mkdir /etc/luks_keys/
# chmod 700 /etc/luks_keys/
# dd if=/dev/urandom of=/etc/luks_keys/disk1.key bs=1024 count=4
4+0 records in
4+0 records out
4096 bytes (4.1 kB, 4.0 KiB) copied, 0.000195992 s, 20.9 MB/s
#
```

Ensure that the key is readable only from root by using the **chmod** command, as shown in Example 5-28.

Example 5-28 Setting permissions to protect key

```
# chmod 0400 /etc/luks_keys/disk1.key
```

Use the **cryptsetup luksAddKey** command to add the generated keyfile to LUKS2 header, which allows it to be used to mount the volume automatically during start, as shown in Example 5-29.

Example 5-29 Adding random key to LUKS2 header

```
# cryptsetup luksAddKey /dev/disk/by-path/ccw-0.0.ff02-part1 \
/etc/luks_keys/disk1.key
Enter any existing passphrase: cryptvolff02
#
```

The system creates device node links under `/dev` pseudo-filesystem. Links that are created in our environment are shown in Example 5-30.

Example 5-30 Links created on /dev pseudo-filesystem

```
/dev/disk/by-id/dm-uuid-CRYPT-LUKS2-d67fe5564ed249958c97e1f4a41cdc3d-enc-luks2-dis
k1
/dev/disk/by-id/dm-name-enc-luks2-disk1
/dev/mapper/enc-luks2-disk1
```

Use the created symlinks to create an LVM2 physical volume, as shown in Example 5-31.

Example 5-31 Creating an LVM2 physical volume with the created LUKS2 format volume

```
# pvcreate /dev/mapper/enc-luks2-disk1
  Physical volume "/dev/mapper/enc-luks2-disk1" successfully created.
#
```

Update `/etc/crypttab` by inserting the keyfile that is used to open the volume during start. You can run the **`zkey crypttab`** command to print the necessary lines for all zkey-managed volumes in the system, as shown in Example 5-32.

Example 5-32 Entry in /etc/crypttab for the newly created volume

```
# zkey crypttab
enc-luks2-disk1 /dev/disk/by-path/ccw-0.0.ff02-part1 /etc/luks_keys/disk1.key luks
#
```

Now, you can create a volume group and logical volume normally by using the newly created volume.

6

Auditing and monitoring the data at-rest environment

Linux on Z and z/VM provide different options for creating, storing, and analyzing audit records. Those options must be configured and tuned according to business policies and compliance requirements to help simplify audit efforts.

The Linux default configuration for logging and monitoring might not meet every need; therefore, more configuration work is required to ensure that the proper data is collected.

In this chapter, we describe some of the options that are offered for Linux on Z and z/VM, and how they can best be used for monitoring and auditing purposes.

This chapter includes the following topics:

6.1 Linux file access auditing

Linux on Z data at-rest encryption provides the means to enable separation of duty and access controls, and help enforce that separation within the operating system.

Linux on Z offers options to monitor I/O operations for specific files or file trees in the entire system. Also, depending on your business policies, there can be specific user IDs or actions you must monitor.

6.1.1 Important files to be monitored

File access monitoring can be tailored for each kind of application. We suggest that the following files are monitored for read/write access by any user:

- Secure Linux files, such as shadow and other files that might store secrets, even hashed.
- Secure and random keyfiles, which are stored in a directory, such as `/etc/zkey/repo/` or directories that were created, such as `/etc/secure_keys/` (see Example 5-9 on page 64).
- Encrypted filesystems files, or a subset of them.
- Remote access control files, such as SSH keys.

Note For more information about all of the important files that must be monitored, review your Linux on Z distribution documentation.

6.1.2 The auditd daemon

The auditd daemon is offered by all major Linux on Z distributions. It can intercept file access calls, process forking and general user activities, and so on. It must be configured with a set of rules and its logs are often sent out to a central server by way of the network. This option is important to ensure data persistence when an attacker features root access to the machine that is being audited.

An invader can stop auditing data collection, but every activity up to the stop command be issued is captured and a sufficient audit trail is generated.

Rules for monitoring the protected filesystems for any operation must be created and logs stored in a separate server, if possible. Because it can generate thousands of lines of log in a short period (depending on the application usage of the filesystem), you might need to modify the monitoring rules for each application profile.

For more information about the use of auditd daemon, see this web page.

Data that is collected by auditd daemon can be analyzed by scripts manually (to follow an audit trail of commands), or summarized by using the **`aureport`** command, as shown in Example 6-1 on page 73.

Example 6-1 Output from the aureport command

```
# aureport
Summary Report
======================
Range of time in logs: 01/03/19 02:45:21.290 - 02/27/19 20:14:01.556
Selected time for report: 01/03/19 02:45:21 - 02/27/19 20:14:01.556
Number of changes in configuration: 1
Number of changes to accounts, groups, or roles: 96963
Number of logins: 18866
Number of failed logins: 54
Number of authentications: 151846
Number of failed authentications: 268
Number of users: 11
Number of terminals: 10
Number of host names: 37
Number of executables: 14
Number of files: 0
Number of AVC's: 0
Number of MAC events: 0
Number of failed syscalls: 0
Number of anomaly events: 1
Number of responses to anomaly events: 7
Number of crypto events: 330597
Number of keys: 0
Number of process IDs: 64065
Number of events: 1647496
#
```

6.1.3 Linux remote logging

The Linux default packages for logging management, syslog-ng, rsyslog, and systemd-journal support event log streaming to a central server by way of the network by using TCP or UDP packages. It is recommended that this log streaming is configured in the environment, with a product receives, processes, stores, and analyzes the logs as they are collected.

IBM QRadar® is an option to centralize this control. For more information, see 6.4, “IBM QRadar” on page 74.

6.2 IBM Resource Access Control Facility for z/VM

The IBM Resource Access Control Facility (RACF) for z/VM is an External Security Manager (ESM). It is an optional component of z/VM that controls access to all z/VM resources.

IBM RACF for z/VM greatly increases the security and robustness of your environment.

RACF uses profiles to check whether access can be granted to a user or guest system based on the access rights that are defined and stored in a RACF database.

By using RACF, you can perform the following tasks:

- Track who uses privileged accounts; that is, MAINT, MAINT630, and MAINT710.
- Prevent technical support user IDs and z/VM guests from being revoked by a password revocation policy. To do so, you define these IDs as Protected user IDs. Together with the RACF class SURROGAT logonby policy, you can get full information about who used the z/VM guest.
- Provide logging mechanisms (SMF records) to show the following information:
  - Who accessed what resources.
  - Which access violations occurred.
- Meet separation of duty requirements by having defined security administrators separately from system administrators.

Currently, RACF does not control or produce audit records about Crypto Express activities. No RACF profile must be created to allow a z/VM Linux guest access to a crypto express adapter or domain. Only the CRYPTO statement is needed in the guest CP directory.

For more information about the use of IBM RACF for z/VM, see IBM Knowledge Center.

6.3 IBM Security zSecure Manager for RACF z/VM

IBM Security zSecure™ Manager for RACF z/VM offers an extended compliance framework for automation and coverage for compliance verification. Audit results can be improved by using a comprehensive, automated audit that references a built-in knowledge base. Manual processes for gathering data to support activities for compliance also can be reduced. IBM RACF for z/VM is a prerequisite for this offering.

For more information about the use of IBM Security zSecure Manager for RACF z/VM, see this web page.

6.4 IBM QRadar

IBM QRadar is an optional software feature that provides Security Information and Event Management (SIEM) capabilities for security activities for Linux on Z data at-rest encryption.

This solution supports consolidating event data from thousands of devices and applications across the infrastructure, including z/VM and Linux on Z, and uncovering suspected security incidents in near real time to support compliance and threat management. It uses the advanced IBM Sense Analytics Engine to baseline normal behavior, detect anomalies, uncover advance threats, and remove false positives.

Managing security events

Many enterprises include a requirement to manage security information and event notifications. SIEM software and hardware were developed to collate and manage these events.

Many sources of such information are available, such as Syslog, middleware logs, and hardware event notifications.

Typically, many records are written to these repositories. Managing these records (such as alert, response, and archiving) can be difficult because the consumers of such information can include a technical, management, security, planning, compliance, and audit audience.

Collating and aggregating such information might not be sufficient. A SIEM must efficiently provide threat and urgency capabilities. The IBM QRadar SIEM offering adds analytics and intelligence to IBM Z-sourced event notifications. Also, layering IBM Security zSecure adds considerably to the capability of QRadar to manage security events.

For more information about the IBM QRadar, see this web page.

6.5 Monitoring

Monitoring can occur at various levels of the environment, from the physical hardware level and all the way up to the Linux guests that are running under z/VM.

6.5.1 IBM Z monitoring

At a physical hardware level, the only monitoring that can be performed is Crypto Express adapter usage.

Although the Crypto Express cards are not directly used for data at-rest encryption (other than secure and protected key processes), it is still important to monitor the adapter usage in the unlikely event of an adapter failure (or scheduled outage for microcode update), so that sufficient capacity is available to continue to run without effect.

For example, if you have two adapters that are over 50% busy, the overall capacity that is required if an outage occurs when workload failovers from one adapter to the other is greater than 100%. This issue results in queuing and delays for the workload, which expects to use Crypto Express adapters.

6.5.2 z/VM monitoring

The main method of monitoring Crypto Express and CPACF usage from within z/VM is the use of the z/VM CP (control program) MONITOR.

z/VM CP writes MONITOR records that are based on your requirements. These records were enhanced to include information about z/VM encrypted paging support. You can enable MONITOR records for System, Processor, Storage, Network, and so on.

The following records are the most useful to monitor and understand Crypto Express usage:

- Domain 5 - Records 9 and 10

  Crypto Express Adapter usage at a z/VM or LPAR Level. These records show physical card usage.

- Domain 3 - Record 2. Encrypted Paging usage

  These records help you to understand how many pages are being encrypted and decrypted per second and the percentage of CPU that is used for each task.

Use the **CP QUERY MONITOR** z/VM command to verify whether z/VM is collecting Crypto Express usage data.

> **Note:** Your z/VM ID must include CP CLASS E to be authorized to use the **CP QUERY MONITOR** command. You can check which CP classes are available for your z/VM ID by using the **CP QUERY PRIVCLASS** command.

Ensure that PROCESSOR DOMAIN is enabled in the EVENT and SAMPLE block by reviewing the output of the **CP QUERY MONITOR** command (see Example 6-2). Crypto Express statistics data records 9 and 10 belong to this domain.

Example 6-2 Verifying whether MONITOR PROCESSOR domain is enabled

```
q monitor
MONITOR EVENT ACTIVE    BLOCK    4     PARTITION    8192
MONITOR DCSS NAME - MONDCSS
CONFIGURATION SIZE       68 LIMIT         1 MINUTES
CONFIGURATION AREA IS FREE
USERS CONNECTED TO *MONITOR - PERFSVM
MONITOR   DOMAIN ENABLED
PROCESSOR DOMAIN ENABLED
STORAGE   DOMAIN ENABLED
SCHEDULER DOMAIN DISABLED
SEEKS     DOMAIN DISABLED
USER      DOMAIN DISABLED
I/O       DOMAIN ENABLED
   PCIF CLASS ENABLED
   ALL DEVICES ENABLED
NETWORK   DOMAIN ENABLED
ISFC      DOMAIN DISABLED
APPLDATA  DOMAIN DISABLED
SSI       DOMAIN DISABLED
COMMAND   DOMAIN DISABLED
MONITOR SAMPLE ACTIVE
               INTERVAL    1 MINUTES
               RATE     2.00 SECONDS
MONITOR DCSS NAME - MONDCSS
CONFIGURATION SIZE     4096 LIMIT         1 MINUTES
CONFIGURATION AREA IS FREE
USERS CONNECTED TO *MONITOR - PERFSVM
MONITOR   DOMAIN ENABLED
SYSTEM    DOMAIN ENABLED
PROCESSOR DOMAIN ENABLED CPUMFC
STORAGE   DOMAIN ENABLED
USER      DOMAIN ENABLED
   ALL USERS ENABLED
I/O       DOMAIN ENABLED
   PCIF CLASS ENABLED
   ALL DEVICES ENABLED
NETWORK   DOMAIN ENABLED
ISFC      DOMAIN DISABLED
APPLDATA  DOMAIN ENABLED
   ALL USERS ENABLED
SSI       DOMAIN DISABLED
```

```
Ready; T=0.01/0.01 14:19:10
```

After the domain is enabled, you can use your preferred z/VM performance tool to visualize the collected monitor data.

For more information about which monitor data is available for crypto express activity, see monitor record 9 (Crypto performance counters) at this web page.

For more information about 10 (Crypto performance measurement data), see this web page.

For more information about how to set up MONITOR to capture your required information, see this web page. Included on this page are various tools to make the information from MONITOR more readable.

Note: The IBM z/VM Performance Toolkit does not report on Crypto Express usage, but it does display which MONITOR events are being captured.

Linux distributions provide the **`perf`** command, which can be used to collect and store performance data for further analysis. When Linux on Z is running native in LPAR, statistics can be queried from the Crypto Express cards by using the following commands that are provided with s390-tools package:

- **`lscpumf`**: Lists the installed CPU-MF facilities for the current hardware
- **`lscpumf -i`**: Shows the authorized counter sets

For more information about CPU-MF related commands, see IBM Knowledge Center.

6.5.3 Linux on Z monitoring

The `libica` package includes the following tools to review the capabilities of your cryptographic hardware and usage counts for these capabilities as used by the guest:

- `icainfo`: Obtains an overview of the supported algorithms with modes of operations and how they are implemented on your Linux system (hardware, software, or both).
- `icastats`: Obtains a statistics table with all crypto operations that are used by the user's processes. For the root user, `icastats` provides statistics for all users, or processes, on the system.

Note: `dm-crypt` cannot be monitored with `icainfo` or `icastats`.

The `s390tools` package also provides the `cpacfstatsd` tool, which is a daemon that can be run to collect the CPACF usage statistics for data at-rest encryption.

Note: At the time of this writing, the `cpacfstatsd` daemon works in a native Linux on Z LPAR only. It does *not* work when Linux on Z is running as a guest under z/VM.

6.5.4 Linux Health Checker tool

The Linux Health Checker is an open source command line tool for Linux on Z. It identifies potential problems before they affect your system's availability or cause outages.

It also collects and compares the active Linux settings and system status for a system with the values that are provided by health-check authors or defined by you. It produces output in the form of detailed messages that provide information about potential problems and the suggested actions to take.

The Linux Health Checker runs on any Linux platform that meets the software requirements. It can be easily extended by writing new health check plug-ins. Currently available health check plug-ins focus on Linux for Z.

As of this writing, 70 heath checks are available, of which 10 are cryptography-related. For more information about a complete list of current checks, see this web page.

The output from the basic and verbose `crypto_cpacf` health check is shown in Figure 6-1.

```
lnxadmin@itsoubup:~$ lnxhc run crypto_cpacf
Collecting system information
Running checks (1 checks)
CHECK NAME                              HOST                                                    RESULT
==========================================================================================================
crypto_cpacf ........................... itsoubup                                               SUCCESS

1 checks run, 0 exceptions found (use 'lnxhc run --replay -V' for details)
lnxadmin@itsoubup:~$


lnxadmin@itsoubup:~$ lnxhc run crypto_cpacf -V
Collecting system information
  Host 'itsoubup'
    file::/proc/cpuinfo                                                                        [SUCCESS]
    file::/proc/sysinfo                                                                        [SUCCESS]
Running checks (1 checks)
CHECK NAME                              HOST                                                    RESULT
==========================================================================================================
crypto_cpacf ........................... itsoubup                                               SUCCESS

Check results:             Exceptions:              Run-time:
  SUCCESS........: 1         High.........: 0         Min per check.: 0.014s
  EXCEPTION......: 0         Medium.......: 0         Max per check.: 0.014s
  NOT APPLICABLE.: 0         Low..........: 0         Avg per check.: 0.014s
  FAILED SYSINFO.: 0         Total........: 0         Total.........: 0.053s
  FAILED CHKPROG.: 0
  PARAM ERROR....: 0
  Total..........: 1
lnxadmin@itsoubup:~$
```

Figure 6-1 Output from the crypto_cpacf health check

The latest version of the Linux Health Checker can be downloaded from the Linux Health Checker page of the SourceForge website.

The supporting documentation is available from IBM Knowledge Center.

7

Maintaining encrypted volumes for data at-rest

In this chapter, we describe options that can be used for maintaining encrypted volumes. The options provide a base for moving data into encrypted volumes, re-enciphering keys, and changing keys.

We also provide examples of how to move data from unencrypted volumes into newly created encrypted volumes for Linux on Z data at-rest.

Some of the tools that are described in this chapter are not yet available with all Linux on Z distributions for LUKS2 format; therefore, our examples are based on Ubuntu 18.10 only.

This chapter includes the following topics:

7.1 Migrating data from existing volumes

In this section, we describe the following ways of migrating data into newly created encrypted volumes:

- The rsync utility, which is a versatile file copying tool
- The pvmove utility, which allows data to be moved from one volume to another

7.1.1 Using rsync

The rsync utility copies data from a file perspective. All metadata, such as access, modification, and change times, are replicated when the **-a** command-line switch for **rsync** is used. Any symbolic link that might exist is replicated to the new target filesystem and its target contents are not mirrored.

If you have other filesystems that are mounted over the source filesystem, dismount and clone them manually to their target partitions.

Any application that is writing to the filesystem must be stopped before the data copy begins.

Creating an encrypted volume and filesystem

Use the process that is described in 5.6.2, “LUKS2 format volume creation” on page 67 to create an LUKS2 format volume in a dedicated LVM2 volume group.

Use the space in the volume group to create a logical volume and a filesystem of your choice.

Mounting in parallel

Mount the newly created encrypted volume and the current data volume in parallel. As shown in Example 7-1, we used the `/mnt` mount point for temporarily mounting the new volume.

Example 7-1 LVM2 logical volume filesystems mounted in parallel

```
# df -Ph /appdata/ /mnt/
File system                                Size  Used Avail Use% Mounted on
/dev/mapper/datavg_clear-appdata           20G  9.9G  8.8G  53% /appdata
/dev/mapper/datavg_encrypted-appdata_enc   20G   45M   19G   1% /mnt
#
```

Copying data

Use the **rsync** command to duplicate data from one filesystem to another, as shown in Example 7-2. Ensure that any applications that use the filesystem are stopped before issuing the command.

Example 7-2 Command to synchronize files from the unencrypted filesystem to the new encrypted one

```
# rsync -av /appdata/ /mnt/
sending incremental file list
./
app_data_file_1
app_data_file_2
app_data_file_3
app_data_file_4
app_data_file_5
app_data_file_6
```

```
app_data_file_7

sent 637,690,309 bytes  received 152 bytes  182,197,274.57 bytes/sec
total size is 637,534,208  speedup is 1.00
#
```

Updating fstab

Update the `/etc/fstab` configuration file to point to the new volume on the mount point. Remove or comment the reference to the old volume mount point to avoid issues during boot-up.

Mounting encrypted device in place

Unmount the old volume and mount the encrypted volume in its place.

7.1.2 Using Logical Volume Manager 2 pvmove

Linux Logical Volume Manager 2 enables moving data between physical devices seamlessly and without any interruption to the normal functions of the system. A physical volume can be moved into one or more physical volumes automatically, if you do not specify a target device and allow LVM to manage the target decision.

Note: Moving logical volumes between physical volumes corrupts filesystems if the physical block size of the target physical volume is larger than the physical block size of the source physical volume. This result occurs because filesystems are aligned to the physical block size of the source physical volume.

For more information about possible data corruption, see the following IBM Knowledge Center web pages:

- Migrating to an encrypted LVM physical volume
- Valid physical block size combinations of LVM physical volumes

To query the physical block size of a device, use the **`blockdev --getpbsz <device>`** command.

In this section, we show how data is moved from an unencrypted physical volume to an encrypted physical volume in the same volume group without any service interruption. The data is encrypted as it is written to the encrypted volume without any added processing.

Figure 7-1 shows how the process of moving data from regular unencrypted to encrypted volumes in the same volume group works.

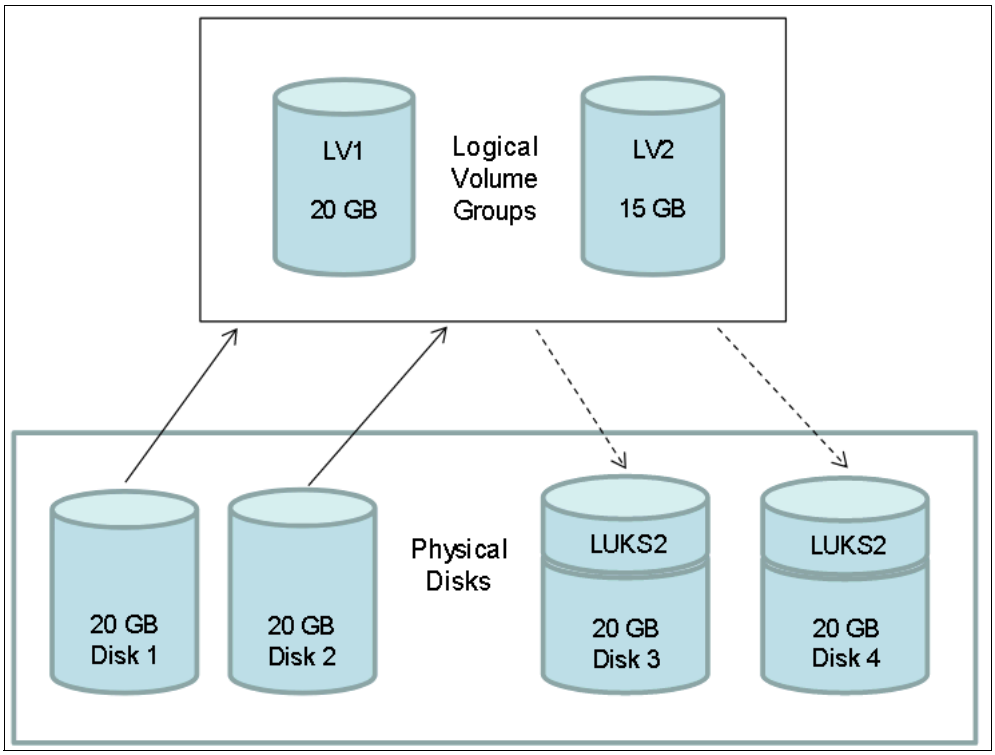

Figure 7-1 Moving LVM data from unencrypted volumes to LUKS2 format encrypted volumes

Creating physical volumes

Create an LUKS2 format volume by using the process that is described in 5.6.2, “LUKS2 format volume creation” on page 67.

Adding physical volume to volume group

Extend the volume group that you want to move by using the new physical volumes.

Moving data from an old physical volume to a new physical volume

Select one of the old physical volumes and use the **pvmove** command to transparently move data to the new volume, as shown in Example 7-3.

Example 7-3 Using pvmove command to move data

```
# pvmove /dev/dasdg1
  /dev/dasdg1: Moved: 0.16%
  /dev/dasdg1: Moved: 37.58%
  /dev/dasdg1: Moved: 72.73%
  /dev/dasdg1: Moved: 100.00%
#
```

This process moves any data that is in `dasdg1` to another disk in the same volume group. You do not need to specify which targets are used.

Removing plain physical volume from group

Remove the plain, now empty volume from the Volume Group by using the **`vgreduce vgname /dev/dasdg1`** command. Repeat the steps that are described in this section until only encrypted volumes are on the Volume Group.

> **Note:** By using the **`cryptsetup-reencrypt`** command, a regular, unencrypted volume can be converted to an LUKS2 format volume in-place, on the same disk, if double the space that is used for the data is available. However, this process is not recommended without first making a complete backup of your data.

7.1.3 Considerations for moving data out of encrypted volumes

When you are considering moving data out of encrypted volumes, make sure that a backup of the data is available before the encrypted volumes are destroyed or the hardware that supports its removed. A volume with no associated Crypto-Express hardware or domain (master key) is an inaccessible volume.

These methods also work for moving data out of encrypted volumes by using the reversed targets instead.

7.2 Re-encrypt clear key to protected key LUKS2 format volume

This process changes an LUKS2 format container from clear key to secure key mode. This change is done by using the `cryptsetup-reencrypt` tool, which reciphers the entire volume.

> **Important:** The volume's content must be re-encrypted by using the new key. The process can take some time to complete, depending on the disk throughput and system load.
>
> Always have a backup of the volume that you want to resize. If the operation fails, a device can be rendered inoperable and all of its data is lost.

In this scenario, the LUKS2 format container in clear key mode and stored in `/dev/mapper/disk9` and named `enc-disk9`. This container is the subject for the in-place process of converting from clear key to secure key.

> **Note:** A clear key is exposed in the memory of the operating system and is susceptible to attacks, such as memory dumping. With the use of secure and protected keys, the effective keys are never exposed in the memory of the operating system.

Generate a secure key for the volume to be converted, as shown in Example 7-4.

Example 7-4 Use of the zkey generate command to create a key

```
# zkey generate --name secure_xtskey9 --keybits 256 --xts --volumes
/dev/mapper/disk9:enc-disk9 --volume-type LUKS2 --apqns 03.0039,04.0039
#
```

Use the **cryptsetup-reencrypt** command create a secure key-based volume, as shown in Example 7-5.

Example 7-5 Use of cryptsetup-reencrypt command to reencrypt by using the new secure key

```
# cryptsetup-reencrypt /dev/mapper/disk9 --cipher paes-xts-plain64
--master-key-file /etc/zkey/repository/secure_xtskey9.skey --key-size 1024
#
```

Set the Verification Pattern for the secured key into the LUKS2 header, as shown in Example 7-6.

Example 7-6 Setting the Verification Pattern into the LUKS2 header with setvp parameter

```
# zkey-cryptsetup setvp /dev/mapper/disk9
#
```

Create an LVM PV with the encrypted volume and use that PV to add as space onto a Volume Group.

7.3 Re-encrypting an LUKS2 format volume by using a different secure key

In situations where you must rotate secure keys, re-enciphering them with a new master key might not be enough.

Note: We suggest that a full backup of all encrypted data is available before this process is started because it must reencrypt all sectors that contain active data, which can lead to data loss if unsuccessful. It can take some time to complete depending on amount of data, disk throughput, and system load.

Generate a new secure key by using only the parameters that are shown in Example 7-7.

Example 7-7 Generating a new secure key by using the zkey command

```
# zkey generate --name new_secure_xtskey_luks2 --keybits 256 --xts
#
```

Use the **cryptsetup-reencrypt** command to reencrypt the entire device with a new secure key. This process reencrypts all disk sectors that contain data and takes some time to complete (see Example 7-8).

Example 7-8 Use of the cryptsetup-reencrypt command to reencrypt the volume

```
# cryptsetup-reencrypt /dev/disk/by-path/ccw-0.0.ff02-part1 --cipher
paes-xts-plain64 --master-key-file
/etc/zkey/repository/new_secure_xtskey_luks2.skey --key-size 1024 --key-slot 0
--key-file /etc/luks_keys/disk1.key
Finished, time 00:01.438,  319 MiB written, speed 221.8 MiB/s
#
```

Set the new Verification Pattern into the LUKS2 header by using the **zkey setvp** command, as shown in Example 7-9.

Example 7-9 Setting the Verification Pattern into the LUKS2 header by using the setvp parameter

```
# zkey-cryptsetup setvp /dev/disk/by-path/ccw-0.0.ff02-part1 --key-file
/etc/luks_keys/disk1.key
#
```

Ensure that the old key is not used for any other purposes; then, remove it as shown in Example 7-10.

Example 7-10 Removing the old key by using the zkey command

```
# zkey remove --name secure_xtskey1_luks
#
```

Assign the new key for the volume, as shown in Example 7-11.

Example 7-11 Updating zkey database by using the new key for the volume

```
# zkey change --name new_secure_xtskey_luks2 --volumes
/dev/disk/by-path/ccw-0.0.ff02-part1:enc-luks2-disk1 --volume-type LUKS2
--sector-size 4096
#
```

8

Performing key management for data at-rest encryption

This chapter covers the key management operations that are needed for a Linux on Z for data at-rest. The chapter features a Linux-specific view and does not include master key-specific topics.

For more information about loading and setting master keys, see 4.4, “Loading the master key” on page 55. Consider a TKE workstation for larger environments.

This chapter includes the following topics:

8.1 The zkey repository

The administrator can use the `zkey` utility to manage secure keys for Linux on Z data at-rest encryption. The `zkey` utility can manage secure keys that are in files or in its own key repository. Keys that are in the key repository are annotated with attributes, which gives the administrator information about the key and how it is used.

Two keys in the key repository are shown in Example 8-1. Both keys are AES 256-bit keys, but only the second key can be used for AES XTS, which is a block cipher that is used for full disk encryption. Most of the listed items can be changed by using **zkey** change command. Even the name can be changed by using **zkey** rename command. Attributes that cannot be changed are the verification pattern, dates (created, changed, and reenciphered) and whether it is an XTS key.

Example 8-1 Listing of keys in a zkey repository

```
Key                          : default_key
--------------------------------------------------------------------------------
        Description          :
        Secure key size      : 64 bytes
        Clear key size       : 256 bits
        XTS type key         : No
        Volumes              : (none)
        APQNs                : (none)
        Key file name        : /etc/zkey/repository/default_key.skey
        Sector size          : (system default)
        Volume type          : luks2
        Verification pattern : 78dfbfd1df164c40837b8a5a3db30e17
                               53c0f108600cbdbd2da918f19a3466f0
        Created              : 2019-02-06 15:43:58
        Changed              : (never)
        Reenciphered        : (never)

Key                          : xtsplain_key
--------------------------------------------------------------------------------
        Description          :
        Secure key size      : 128 bytes
        Clear key size       : 512 bits
        XTS type key         : Yes
        Volumes              : (none)
        APQNs                : (none)
        Key file name        : /etc/zkey/repository/xtsplain_key.skey
        Sector size          : 4096 bytes
        Volume type          : plain
        Verification pattern : 6592436f38618f053c8c7c73a1cd2b2e
                               2f48dd7a6b3455c750d261f0efdfd708
        Created              : 2019-02-06 15:47:31
        Changed              : (never)
        Reenciphered        : (never)
```

Consider the following important items:

`Secure key size`	Size of the secure keyfile. The secure key size includes only a limited correlation to the clear key size.
`Clear key size`	Size in bits of the AES key. Can be 128, 192, or 256. With AES XTS keys, only 128 and 256-bit sizes are supported. The default is 256 bit.
`Volumes`	Defines the volumes that use the secure key and the dm-crypt name that the device uses when it is opened by using the **luksOpen** command. The format is `<path to volume>:<dm-crypt name>`. For example, `/dev/dasdc1:enc-disk1` has the disk name `dasdc1` under `/dev/dasdc1` and after **luksOpen**, the name is `enc-disk1` under `/dev/mapper`.
`APQNs`	Specifies the Crypto Express card adapter and domain numbers that are used with this secure key. For example, `01.0001` defines adapter number one with domain one.
`Sector size`	Defines the sector size of the volumes that are associated with this key. Can be the default (512 bytes) or 4096 bytes when defined.
`Volume type`	The volume type can be plain or LUKS2 format.

8.2 Working with the repository

Keys that are in the key repository can be exported, imported, copied (duplicated), and renamed. Attributes of the key also can be added, changed, and removed. For example, duplicating a key is used when that same key is needed for a different volume type (one key for plain format and one key for the LUKS2 format).

When a key is exported, this key is copied from the repository to a file. The key loses most of its attributes (except for attributes that are maintained within the secure key object) when exported. An import copies a key from a file into the key repository. Attributes can be associated with the key when a key is imported. The export process for key `secure_xtskey2` and the raw content of the key are shown in Example 8-2.

Example 8-2 Exporting a secure key as a file

```
$# zkey list --name secure_xtskey2
Key                        : secure_xtskey2
--------------------------------------------------------------------------------
        Description          :
        Secure key size      : 128 bytes
        Clear key size       : 512 bits
        XTS type key         : Yes
        Volumes              : /dev/dasdd1:enc-disk2
        APQNs                : (none)
        Key file name        : /etc/zkey/repository/secure_xtskey2.skey
        Sector size          : (system default)
        Volume type          : LUKS2
        Verification pattern : ae350bef5693ad73eaea396beb44608a
                               3e9c4861111f5698b6ecb586e79855ba
        Created              : 2019-01-24 16:49:43
        Changed              : (never)
        Reenciphered        : (never)

$# zkey export exported-key.skey  --name secure_xtskey2
```

```
$# hexdump exported-key.skey
0000000 0100 0000 0400 c0eb b264 2d0c 3bfa 93ef
0000010 d872 527f 6dc7 1f1e 0c18 49f3 6b90 1af2
0000020 913c 7aba 5e85 970d e638 e8fb 4dfc b33d
0000030 0000 0000 0000 0000 0100 0020 d639 0687
0000040 0100 0000 0400 c0e9 b264 2d0c 3bfa 93ef
0000050 cb4b 7991 1e16 62f6 5b69 6110 c112 6831
0000060 f6b0 ea63 dcad 6ea3 6ef6 ddd9 182c 4745
0000070 0000 0000 0000 0000 0100 0020 54be a5f0
0000080
```

As shown in Example 8-3, this key is imported as `imported-xts-key` in the key repository. The use of the **zkey** command generates the verification pattern and because the key is the same; that is, it matches the verification pattern of `secure_xtskey2`.

Example 8-3 Importing a secure key from a file

```
$# zkey import exported-key.skey --name imported-xts-key -d "Imported key - Copy
of secure_xtskey2"
$# zkey list --name imported-x*
Key                         : imported-xts-key
--------------------------------------------------------------------------------
        Description          : Imported key - Copy of secure_xtskey2
        Secure key size      : 128 bytes
        Clear key size       : 512 bits
        XTS type key         : Yes
        Volumes              : (none)
        APQNs                : (none)
        Key file name        : /etc/zkey/repository/imported-xts-key.skey
        Sector size          : (system default)
        Volume type          : luks2
        Verification pattern : ae350bef5693ad73eaea396beb44608a
                               3e9c4861111f5698b6ecb586e79855ba
        Created              : 2019-02-11 22:35:56
        Changed              : 2019-02-12 18:37:23
        Re-enciphered        : (never)

$# zkey remove --name imported-xts-key
zkey: Remove key 'imported-xts-key'? y
```

As shown at the end of Example 8-3, we remove the key again. The `zkey` utility needs a confirmation for this step, but it can be skipped by using the **`--force`** option, if needed (for example, in scripts).

Attention: `dm-crypt` needs the key for plain setup volumes in the repository to unlock it. Removing such a key prompts `zkey` to display a message that indicates which volumes are associated with this key. Remove the key only if you no longer use these volumes or if you have a backup key. Otherwise, the associated volumes become unusable.

LUKS2 format volumes do not need the key in the repository. However, the key is saved to a different location for backup reasons.

The copy process for a key is shown in Example 8-4. All attributes are copied except for the volumes, which cannot be copied. This process is done because volumes normally do not have two keys to unlock it. The verification pattern indicates that both keys (`secure_xtskey`) and (`copy_of_secure_xtskey`) are the same.

Example 8-4 Copying a secure key in the zkey repository

```
$# zkey list -N secure_xtskey
Key                          : secure_xtskey
--------------------------------------------------------------------------------
        Description          :
        Secure key size      : 128 bytes
        Clear key size       : 512 bits
        XTS type key         : Yes
        Volumes              : /dev/dasdc1:enc-disk1
        APQNs                : (none)
        Key file name        : /etc/zkey/repository/secure_xtskey.skey
        Sector size          : (system default)
        Volume type          : LUKS2
        Verification pattern : 7dc408c6ffbcb861b377bd3b7a48fa6f
                               3781f8972c74c1ff148219c19080a92f
        Created              : 2019-01-24 16:23:22
        Changed              : (never)
        Re-enciphered        : (never)
$# zkey copy -N secure_xtskey -w copy_of_secure_xtskey
$# zkey list -N copy_of_secure_xtskey
Key                          : copy_of_secure_xtskey
--------------------------------------------------------------------------------
        Description          :
        Secure key size      : 128 bytes
        Clear key size       : 512 bits
        XTS type key         : Yes
        Volumes              : (none)
        APQNs                : (none)
        Key file name        : /etc/zkey/repository/copy_of_secure_xtskey.skey
        Sector size          : (system default)
        Volume type          : LUKS2
        Verification pattern : 7dc408c6ffbcb861b377bd3b7a48fa6f
                               3781f8972c74c1ff148219c19080a92f
        Created              : 2019-02-12 18:49:01
        Changed              : (never)
        Re-enciphered        : (never)
```

As shown in Example 8-5, we now assign a new volume to the secure key. The `zkey` utility checks if this volume is an available block device and if no other key uses the same volume. Use the + and - prefix to inform `zkey` if it must add or remove an associated volume. The same rule applies to adding and removing cards or domains.

Example 8-5 Changing the associated volume and the name

```
root@s96lp05:~# zkey change --name copy_of_secure_xtskey --volumes
+/dev/dasda1:enc-disk2
$# zkey rename --name copy_of_secure_xtskey -w securekey_for_dasda1
$# zkey list --name securekey_for_dasda1
```

```
Key                          : securekey_for_dasda1
--------------------------------------------------------------------------------
        Description          :
        Secure key size      : 128 bytes
        Clear key size       : 512 bits
        XTS type key         : Yes
        Volumes              : /dev/dasda1:enc-disk3
        APQNs                : (none)
        Key file name        : /etc/zkey/repository/securekey_for_dasda1.skey
        Sector size          : (system default)
        Volume type          : LUKS2
        Verification pattern : 7dc408c6ffbcb861b377bd3b7a48fa6f
                               3781f8972c74c1ff148219c19080a92f
        Created              : 2019-02-12 18:49:01
        Changed              : 2019-02-12 18:58:34
        Re-enciphered        : (never)
```

After assigning the volume, the key is renamed to `securekey_for_dasda1`.

8.3 Rotating a master key

When the master key of a domain must be rotated, all secure keys must be reenciphered to the new master key. Key repository administrators can use the `zkey` utility to reencipher all keys that are associated with one specific APQN or select specific keys to be reenciphered.

8.3.1 Reenciphering keys in the key repository

Reenciphering is done by using the `zkey` utility. A valid secure key that is wrapped with the old master key is shown in Example 8-6. If no master key rotation is done, the key is usable. If the master key is rotated again, the secure key becomes invalid and cannot be used.

Therefore, the secure key must be reenciphered to the current master key before the next master key rotation by using the **`zkey reencipher`** command.

Example 8-6 Validating a secure key that is wrapped with the old master key

```
$# zkey validate
Key                          : secure_xtskey1_luks
--------------------------------------------------------------------------------
        Status               : Valid
        Description          :
        Secure key size      : 128 bytes
        Clear key size       : 512 bits
        XTS type key         : Yes
        Enciphered with      : OLD CCA master key
        Volumes              : /dev/disk/by-path/ccw-0.0.ff02-part1:enc--luks2-disk1
        APQNs                : 00.001f
        Key file name        : /etc/zkey/repository/secure_xtskey1_luks.skey
        Sector size          : 4096 bytes
        Volume type          : LUKS2
        Verification pattern : 277419afcf2e816bf053ffa0b4b5e143
                               7127fc355a69ceac9305b1dbff024a07
        Created              : 2019-02-12 14:37:38
        Changed              : (never)
```

```
        Re-enciphered        : (never)
WARNING: The secure key is currently enciphered with the OLD CCA master key. To
mitigate the danger of data loss re-encipher it with the CURRENT CCA master key
1 keys are valid, 0 keys are invalid, 1 warnings
```

Example 8-7 shows the reenciphering process of all keys that are associated with APQN 00.001f. The only key that is associated with this APQN is `secure_xtskey1_luks`. It also shows if the keys are reenciphered to the current or the new master key. If a key has a defined, associated LUKS2 format volume, it also shows the command that must be used to reencipher the key in the LUKS2 header.

Example 8-7 Reenciphering process for keys that are associated with APQN 00.001f

```
$# zkey reencipher --apqns 00.001f
Re-enciphering key 'secure_xtskey1_luks'
The secure key is currently enciphered with the OLD CCA master key and is being
re-enciphered with the CURRENT CCA master key
The following LUKS2 volumes are encrypted with key 'secure_xtskey1_luks'. You
should also re-encipher the volume key of those volumes using command
'zkey-cryptsetup reencipher <device>':
  /dev/disk/by-path/ccw-0.0.ff02-part1:enc--luks2-disk1
1 keys re-enciphered, 0 keys skipped, 0 keys failed to re-encipher
```

Note: When a key uses an old master key, the key is reenciphered in-place by default so that it can be used immediately. For reenciphering a secure key from current to the new master key, a staged approach is used by default.

In this process, the secure key that is encrypted with the current master key is duplicated before reenciphering the secure key with the new master key. This process makes it possible to use the secure key with the current master key until the new master key is set. After the new key is set, the process must be completed by using the **--complete** option. The in-place and staged process can be forced by using the **--staged** or **--in-place** options.

The **zkey validate** command (see Example 8-8) shows the secure key that is reenciphered with the current master key and the date of the reenciphering process.

Example 8-8 Validating secure key after it was reenciphered

```
$# zkey validate
Key                          : secure_xtskey1_luks
--------------------------------------------------------------------------------
        Status               : Valid
        Description          :
        Secure key size      : 128 bytes
        Clear key size       : 512 bits
        XTS type key         : Yes
        Enciphered with      : CURRENT CCA master key
        Volumes              : /dev/disk/by-path/ccw-0.0.ff02-part1:enc--luks2-disk1
        APQNs                : 00.001f
        Key file name        : /etc/zkey/repository/secure_xtskey1_luks.skey
        Sector size          : 4096 bytes
        Volume type          : LUKS2
        Verification pattern : 277419afcf2e816bf053ffa0b4b5e143
                               7127fc355a69ceac9305b1dbff024a07
        Created              : 2019-02-12 14:37:38
```

```
        Changed              : (never)
        Re-enciphered        : 2019-02-12 14:46:02
1 keys are valid, 0 keys are invalid, 0 warnings
```

8.3.2 Reenciphering volume keys

The administrator uses the **zkey-cryptsetup** command for reenciphering volume master keys. For reencrypting a volume under a different secure key (for example, if the old key is compromised), the **cryptsetup-reencrypt** command is used. For more information, see 7.3, “Re-encrypting an LUKS2 format volume by using a different secure key” on page 84.

Reenciphering a volume

In most cases, the commands for **zkey-cryptsetup** are similar to the commands that are used with `zkey`.

Example 8-9 shows a memory dump of the LUKS header of the volume `/dev/disk/by-path/ccw-0.0.ff02-part1`, which is associated with the `secure_xtskey1_luks` key that was reenciphered as described in Chapter 7, “Maintaining encrypted volumes for data at-rest” on page 79.

Example 8-9 Dump of an LUKS2 header that uses a secure key as volume master key (paes cipher)

```
$# cryptsetup luksDump /dev/disk/by-path/ccw-0.0.ff02-part1
LUKS header information
Version:        2
Epoch:          5
Metadata area:  12288 bytes
UUID:           36d3ef52-c7cd-4199-b76f-5c4af6597b1d
Label:          (no label)
Subsystem:      (no subsystem)
Flags:          (no flags)
Data segments:
  0: crypt
    offset: 4194304 [bytes]
    length: (whole device)
    cipher: paes-xts-plain64
    sector: 4096 [bytes]
Keyslots:
  0: luks2
    Key:        1024 bits
    Priority:   normal
    Cipher:     aes-xts-plain64
    PBKDF:      argon2i
    Time cost:  4
    Memory:     448517
    Threads:    2
    Salt:       80 f3 a8 01 d6 48 15 73 f8 6f 11 39 11 57 ac 2c
                9f 21 30 60 38 b8 13 02 33 1b aa a9 a4 75 40 53
    AF stripes: 4000
    Area offset:32768 [bytes]
    Area length:512000 [bytes]
    Digest ID:  0
  1: luks2
    Key:        1024 bits
    Priority:   normal
```

```
    Cipher:     aes-xts-plain64
    PBKDF:      argon2i
    Time cost:  4
    Memory:     491980
    Threads:    2
    Salt:       b6 be a1 e2 84 c7 44 82 0f da 08 e8 13 6f fa 94
                22 96 0a 73 8d f6 e4 bc c3 2c 5e eb a2 d0 a3 0c
    AF stripes: 4000
    Area offset:544768 [bytes]
    Area length:512000 [bytes]
    Digest ID:  0
Tokens:
  0: paes-verification-pattern
Digests:
  0: pbkdf2
    Hash:       sha256
    Iterations: 33470
    Salt:       8a 05 8a a4 42 d9 4f 32 b9 0e ec a7 f6 5f 79 3e
                a2 88 20 d5 b2 f9 76 b7 48 8c ce 77 13 c0 50 5e
    Digest:     59 03 7f f6 08 02 7d a0 27 c8 3a 34 93 0b b5 5b
                bb c0 8f 1a cf 5c 4b cd 47 d5 de a2 67 3d 59 d9
```

The header shows the key slots, salts, and digest that we used for the key derivation process to unlock the master key of the volume (the secure key). Next, we reencipher the secure key and re-create the LUKS2 header. The first part of a staged approach is shown in Example 8-10. The master key (which is the secure key) of volume `/dev/disk/by-path/ccw-0.0.ff02-part1` is reenciphered by using the new master key.

Example 8-10 Reenciphering a volume with a staged approach

```
$# zkey-cryptsetup reencipher /dev/disk/by-path/ccw-0.0.ff02-part1
Enter passphrase for '/dev/disk/by-path/ccw-0.0.ff02-part1':
The secure volume key of device '/dev/disk/by-path/ccw-0.0.ff02-part1' is
enciphered with the CURRENT CCA master key and is being re-enciphered with the
NEW CCA master key.
Staged re-enciphering is initiated for device
'/dev/disk/by-path/ccw-0.0.ff02-part1'. After the NEW CCA master key has been
set to become the CURRENT master key, run 'zkey-cryptsetup reencipher' with
option '--complete' to complete the re-enciphering process.
```

In Example 8-11, this process is completed after the new master key was set as the current key.

Example 8-11 Completing the reenciphering process

```
$# zkey-cryptsetup reencipher /dev/disk/by-path/ccw-0.0.ff02-part1 --complete
Enter passphrase for key slot 0 of '/dev/disk/by-path/ccw-0.0.ff02-part1':
Re-enciphering has completed successfully for device
'/dev/disk/by-path/ccw-0.0.ff02-part1'.
All key slots containing the old volume key are now in unbound state. Do you
want to remove these key slots?
y
WARNING:  Before re-enciphering, the volume's LUKS header had multiple active
key slots with the same key, but different passwords. Use 'cryptsetup
```

```
luksAddKey' if you need more than one key slot.
```

Note: Completing the reenciphering of LUKS2 format volumes makes the key slots unusable, which were not used for reenciphering. If more than one key slot is used (for example, for a keyfile), they must be added again by using the **cryptsetup luksAddKey** command.

As shown in Example 8-12, the key slots and the digest object of the LUKS header are changed.

Example 8-12 Dump of the LUKS header after the secure key was reenciphered

```
$# cryptsetup luksDump /dev/disk/by-path/ccw-0.0.ff02-part1
LUKS header information
Version:        2
Epoch:          10
Metadata area:  12288 bytes
UUID:           36d3ef52-c7cd-4199-b76f-5c4af6597b1d
Label:          (no label)
Subsystem:      (no subsystem)
Flags:          (no flags)
Data segments:
  0: crypt
    offset: 4194304 [bytes]
    length: (whole device)
    cipher: paes-xts-plain64
    sector: 4096 [bytes]
Keyslots:
  2: luks2
    Key:        1024 bits
    Priority:   normal
    Cipher:     aes-xts-plain64
    PBKDF:      argon2i
    Time cost:  4
    Memory:     348099
    Threads:    2
    Salt:       1d 3b 9d 29 6f 80 cd b9 4a 2a 98 92 cc ed 3e 62
                8a aa fb 47 c6 39 bd 2f b7 fd ca 08 48 62 a8 ea
    AF stripes: 4000
    Area offset:1056768 [bytes]
    Area length:512000 [bytes]
    Digest ID:  1
Tokens:
  0: paes-verification-pattern
Digests:
  1: pbkdf2
    Hash:       sha256
    Iterations: 31207
    Salt:       4e 0f ca 7e 51 e4 4e e2 d6 21 40 bb 77 a8 37 11
                ad 85 99 63 ac 02 7b da 2e c2 70 d2 b1 2b 31 03
    Digest:     86 2e c0 7c 15 b8 26 c3 9b 0b 78 d4 fc 08 88 c7
                d1 9a 06 fb 98 9d 4d ab 7b 2b e9 bf d3 8e 88 c4
```

8.3.3 Recovering invalid volume master keys

If the corresponding master key of a secure key is lost, the secure key can no longer be used to unlock a volume. Therefore, the associated secure keys are unusable. This issue can occur if a secure key was not reenciphered.

If only the volume master key was forgotten and the same key in the key repository was reenciphered to the current or old master key, the key of the volume can be replaced with the key of the repository by using the **`cryptsetup setkey`** command.

For more information, see IBM Knowledge Center.

> **Warning:** Setting a new secure key for a volume can make the volume unusable if the wrong secure key is selected.

A

Using protected keys for high-speed encryption

The use of secure keys and protected keys in the Linux on Z data at-rest encryption process ensures that key material is not available or visible to unauthorized users at any time.

The Central Processor Assist for Cryptographic Functions (CPACF) wrapping key is used to rewrap (encrypt) a secure key after it is decrypted. The CPACF wrapping key is in a protected area of the hardware system area (HSA), which is not visible to the operating system or applications.

In this appendix, we show the key wrapping process for the z13 and z14 with Crypto Express adapters.

Rewrapping a secure key into a protected key with Crypto Express6S

The key wrapping process on a z14 with a Crypto Express6S adapter is shown in Figure A-1. Notice that the data key material is not in the clear at any point in the process.

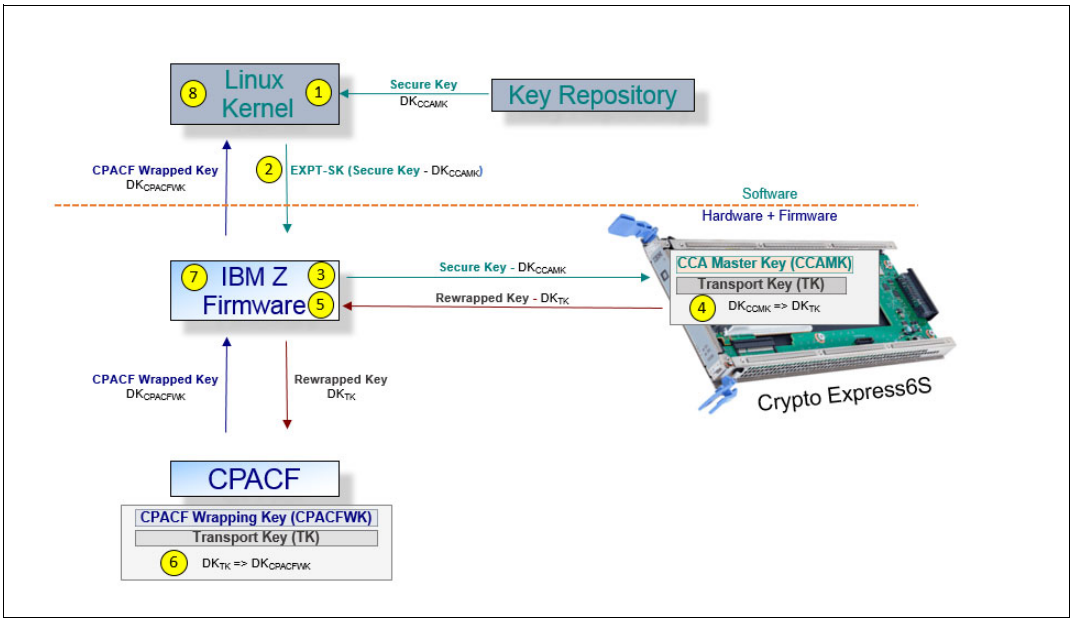

Figure A-1 Key wrapping process with the Crypto Express6S

The following process is used to rewrap a secure key into a protected key (as shown in Figure A-1):

1. The Linux kernel retrieves the data key, which is stored as a secure key (encrypted by using a master key [CCAMK]) from the LUKS2 header of the volume that should be opened or (as shown in Figure A-1) retrieves it from a key repository if plain mode is used for the volume.
2. The Linux kernel starts the process by sending a command with the secure key to IBM Z firmware.
3. IBM Z firmware sends the secure key with transport key[1] information to Crypto Express6S.
4. Crypto Express6S decrypts the secure key by using the master key and rewraps the data key by using a transport key.
5. The rewrapped data key (encrypted by using the transport key) is sent back to IBM Z firmware.
6. IBM Z firmware starts CPACF to unwrap and rewrap the data key by using a CPACF wrapping-key[2] to create a protected key.
7. IBM Z firmware returns the CPACF wrapped-key (protected key) to the Linux kernel.
8. The Linux kernel caches the protected key in the kernel memory and removes it again when the volume is closed.

[1] Transport keys are derived for each cryptographic domain by way of a key agreement protocol between IBM Z firmware and Crypto Express firmware.

[2] CPACF wrapping key and transport key are in a protected area of HSA that is not visible to the operating system or application.

Rewrapping a secure key into a protected key with Crypto Express5S

The key wrapping process works differently with a Crypto Express5S adapter compared to the Crypto Express6S adapter. The process of rewrapping a secure key to protected key on a z13 with a Crypto Express5S is shown in Figure A-2. The process is similar to earlier generations of the IBM Z platform and Crypto Express adapters.

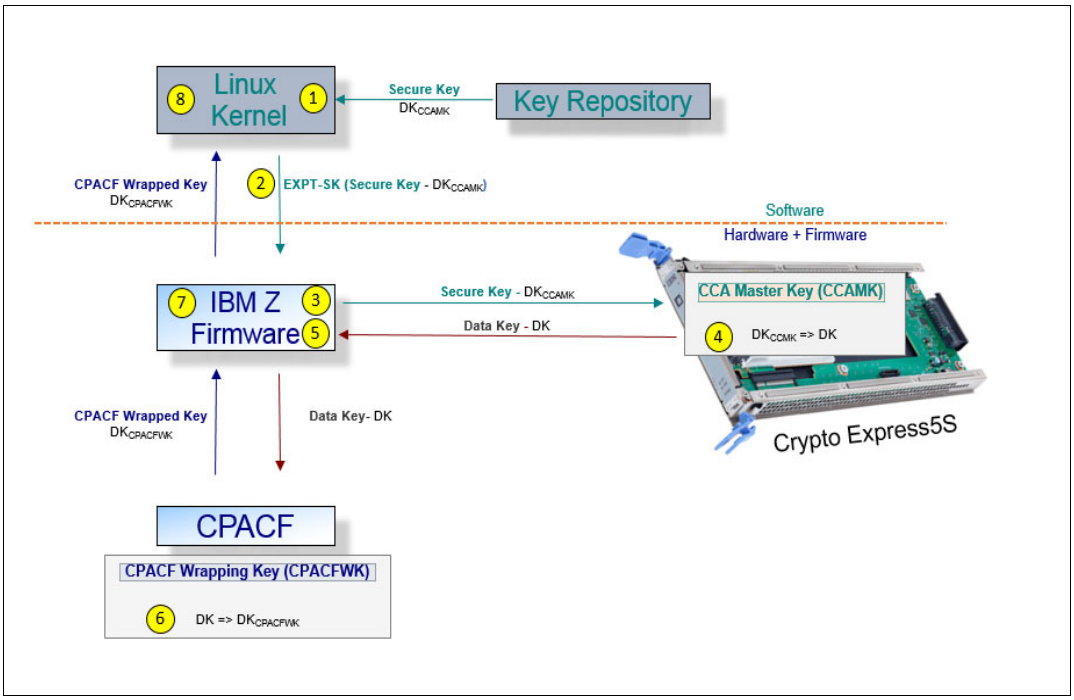

Figure A-2 Key wrapping process with the Crypto Expess5S

The following process is shown in Figure A-2:

1. The Linux kernel retrieves the data key, which is stored as a secure key (encrypted by using a master key [CCAMK]) from the LUKS2 header of the volume that should be opened or (as shown in Figure A-2) retrieves it from a key repository if plain mode is used for the volume.
2. The Linux kernel starts the process by sending a command with the secure key to IBM Z firmware.
3. IBM Z firmware sends the secure key to the Crypto Express5S.
4. The Crypto Express5S decrypts the secure key by using the master key.
5. The data key is sent to Z firmware.
6. IBM Z firmware starts CPACF to wrap the data key by using a CPACF wrapping key to create a protected key.
7. IBM Z firmware returns the CPACF wrapped key (protected key) to the Linux kernel.
8. The Linux kernel caches the protected key in the kernel memory and removes it again when the volume is closed.

Redbooks

Getting Started with Linux on Z Encryption for Data At-Rest

(0.2”spine)
0.17”<->0.473”
90<->249 pages

SG24-8436-00

ISBN 0738457469

Printed in U.S.A.