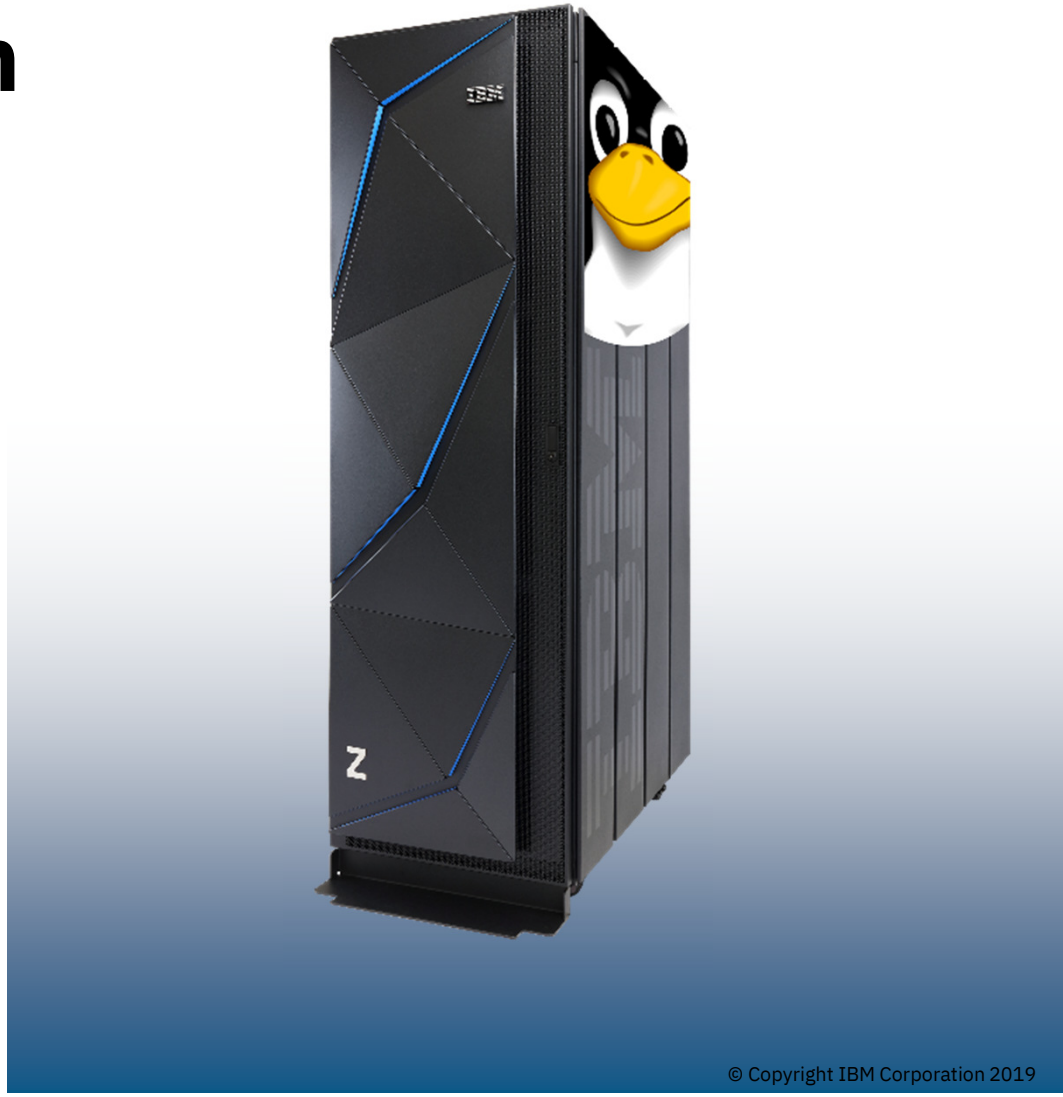
End-to-end Encryption of Data at Rest
for Linux on Z and LinuxONE

Ingo Franzki
ifranzki@de.ibm.com

Reinhard Buendgen
buendgen@de.ibm.com



Data Protection and Security are Business Imperatives

"It's no longer a matter of if, but when ..."

Nearly **6,5 million** records stolen per day,

267,905 per hour

and **4,465** per minute. [1]

24%

Likelihood of an organization having a data breach in the next 24 months [2]

The **greatest security mistake** organizations make is failing to protect their networks and data from **internal threats**. [3]

Of the **14 Billion** records breached since 2013

only **4%** were encrypted [4]

1 http://breachlevelindex.com/
2 2016 Ponemon Cost of Data Breach Study: Global Analysis -- http://www.ibm.com/security/data-breach/
3 Steve Marsh in article: https://digitalguardian.com/blog/expert-guide-securing-sensitive-data-34-experts-reveal-biggest-mistakes-companies-make-data
4 Breach Level Index -- http://breachlevelindex.com/

The Value of Data ...

— **Today data is one of the most valuable assets of many companies**

— **In particular sensitive data must be protected against unauthorized access to avoid**
- Losing customer trust
- Losing competitive advantages
- Being subject to fines and regression claims

— **Data encryption is the most effective way to protect data outside your system be it in flight or at rest**

— **But encrypting data is not easy**
- Requires the introduction of new policies
- Complicates data management
- Requires to securely manage keys
- Costs computing resources

Attacks

Online attack

- Steal or modify data from inside your system
- Requires system access with required privileges
- Attack is observable
  - Takes time proportional to the amount of data that is attacked
  - Has impact on the operation of attacked system (costs resources)
  - It is hard not to leave traces

Offline attack

- Steal or modify data from outside your system
  - The data might even be a dump of your system at a service organization
- Requires access to network, SAN, media
  - May but need not involve stealing media
  - May require access to an decryption key
- Attack is hard to observe if it is observable at all
  - Neither time nor resource constrained

Protecting data against offline attacks

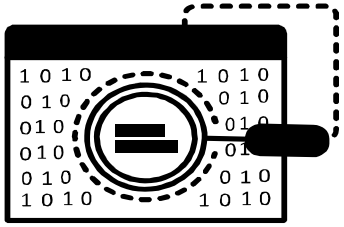
How can you control access outside your system?

- Data transferred via intranet or internet
- Data transferred via SAN
- Data stored in storage subsystems

Do you trust access control mechanisms of network/SAN/storage subsystems?

Solution:

You better encrypt that data!



Cryptography

Can

- Prove data provenance
- Prove data integrity
- Protect data confidentiality

- Kerkhoff's Principle for secure cryptography:
  - All cryptography methods should be well known
  - Only the cryptographic keys must be secret

When you protect your data using cryptography you must protect your keys!

They are the most critical piece of data!

Cannot

- Protect you from your keys getting stolen

  - by an insider
  - by an intruder
  - via a vulnerability

Here is a Dream ...

What if you could just encrypt all data in-flight and at-rest

- At no cost
- Without changing applications
- Without changing data management
- By pushing a single button

Well, that will remain to be a dream

But with **pervasive encryption** we want to make a large step in that direction

Pervasive Encryption for the Linux on Z and LinuxONE Platform

Technical Foundation

IBM z14 or Emporer II - Designed for Pervasive Encryption

- **CPACF** – Dramatic advance in bulk symmetric encryption performance
- **Crypto Express6S** – Doubling of asymmetric encryption performance for TLS handshakes

Linux on Z and LinuxONE - Full Power of Linux Ecosystem combined with IBM z14 or Emporer II Capabilities

- **dm-crypt** – Transparent volume encryption using industry unique CPACF protected-keys
- **Network Security** – Enterprise scale encryption and handshakes using CPACF and SIMD
- **Secure Service Container** – Automatic protection of data and code for virtual appliances

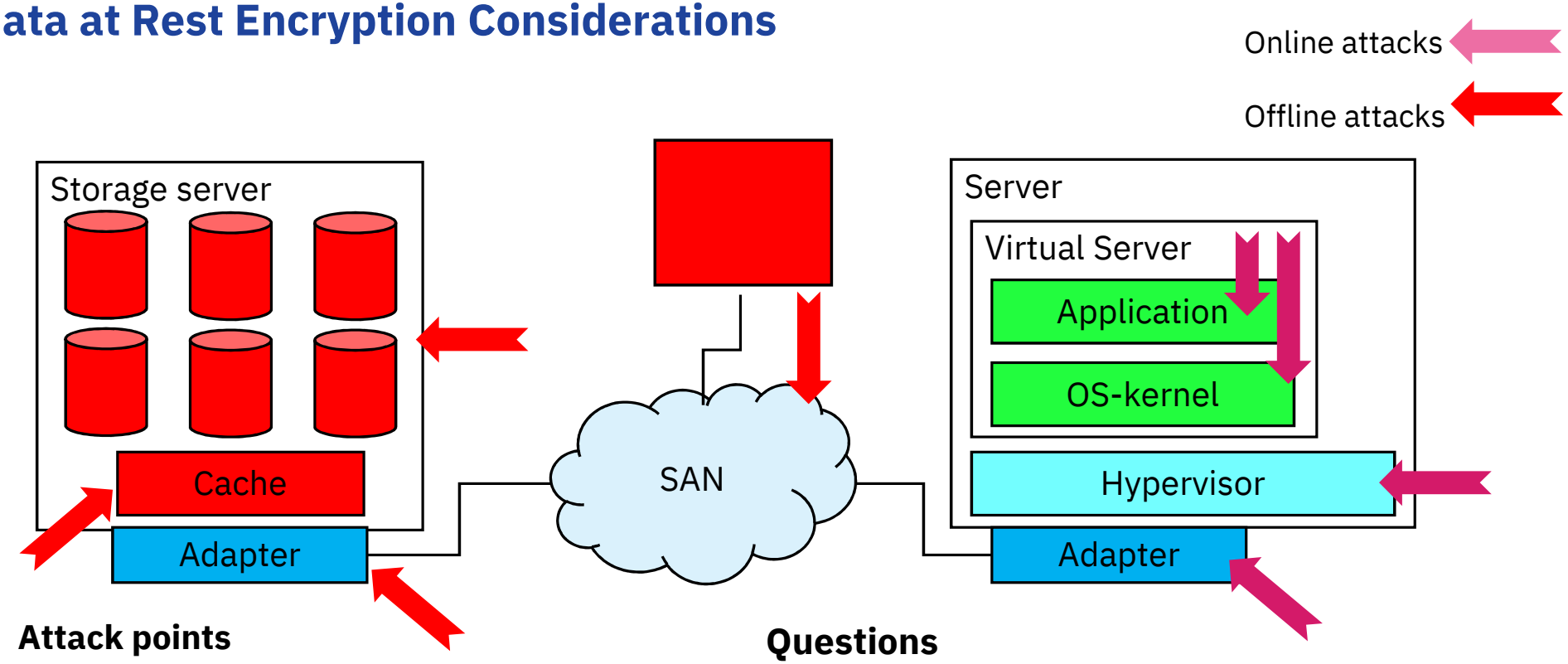
z/VM – New: encrypted paging

- z/VM 6.4 APAR VM65993

Pervasive Encryption for Linux on Z and LinuxONE

Technical Contents: Data at Rest

Data at Rest Encryption Considerations



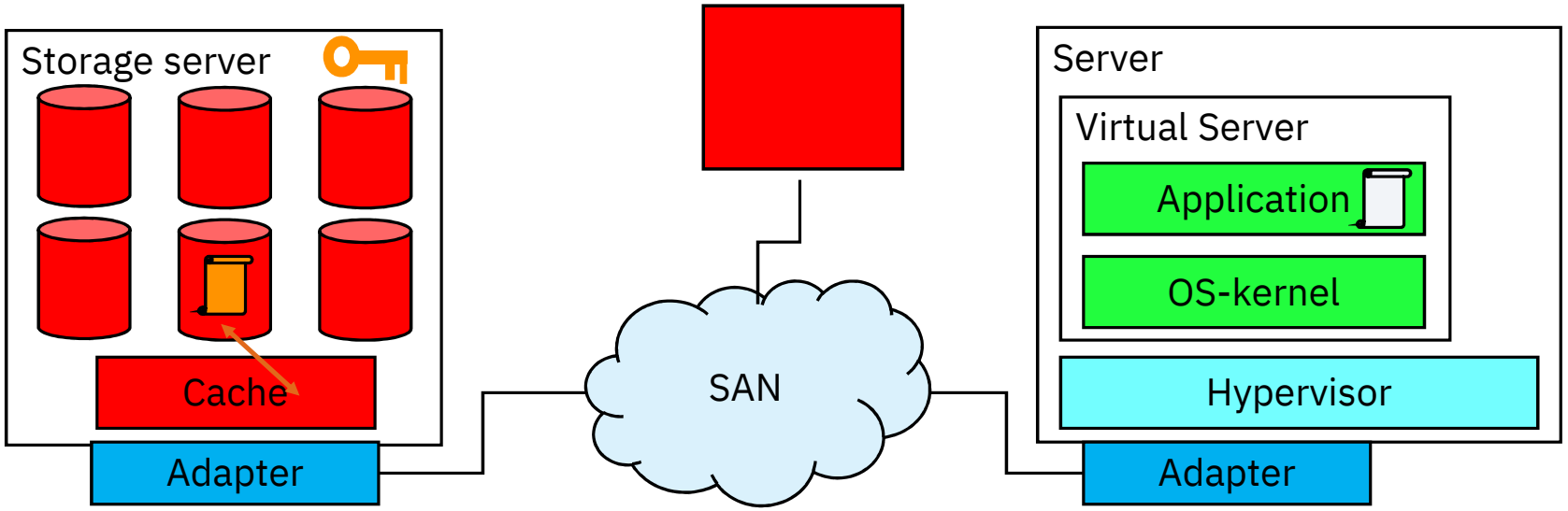
Attack points

- Storage server
- SAN
- Server / Hypervisor / Virtual Server
  - Insider / outsider

Questions

- Where is data decrypted/encrypted?
- Who generates keys?
- Who owns (can access) the keys?
- Backups? Data migration?

Data Encryption on Storage Subsystem



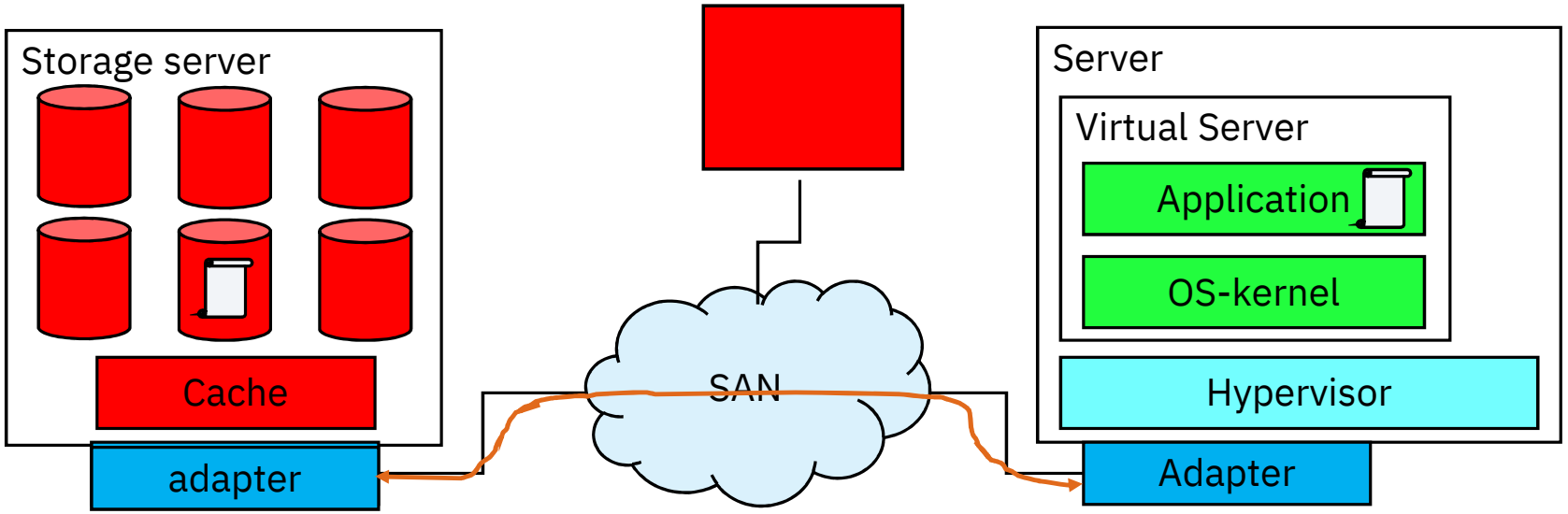
Attack points

- Storage server – (Secured)
- SAN – Not secure
- Server / Hypervisor – Not secure
- Virtual Server insider / outsider – Not secure

Questions

- Where is data decrypted/encrypted?
- Who generates keys? Storage/OS
- Who owns (can access) the keys? Storage/OS admin

End-to-End SAN/Network Encryption



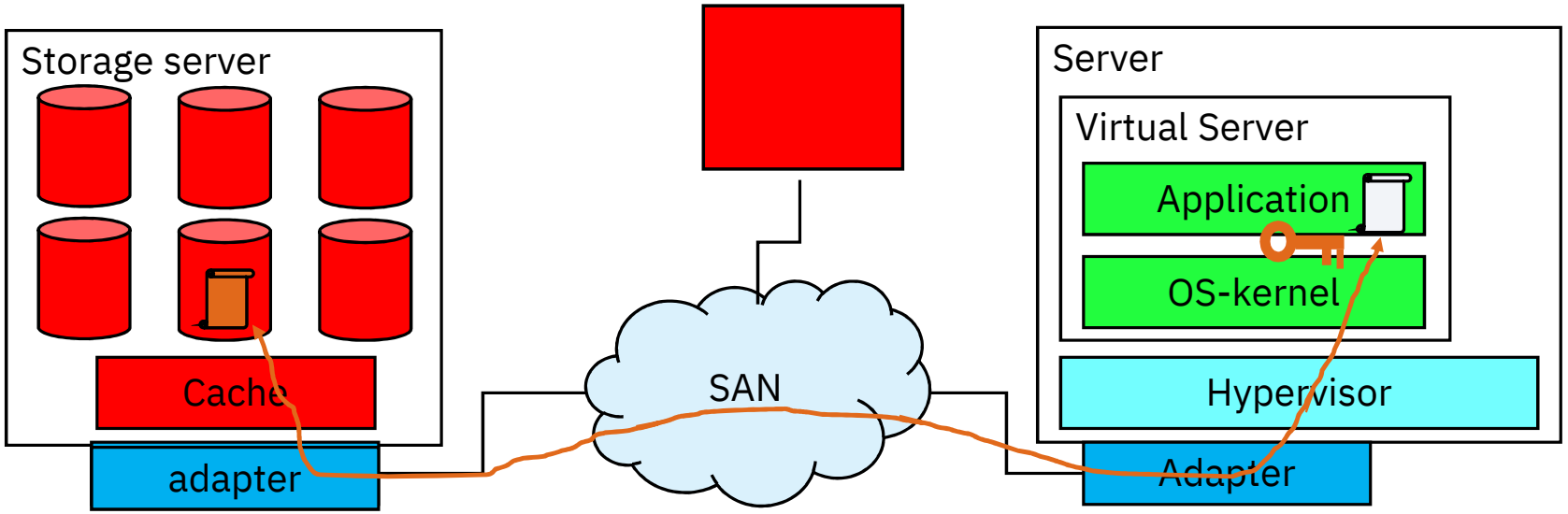
Attack points

- Storage server – Not secure
- SAN – Secured
- Server / Hypervisor – Not secure
- Virtual Server insider / outsider – Not secure

Questions

- Where is data decrypted/encrypted? Adapter
- Who generates keys? System admin
- Who owns (can access) the keys? System admins

End-to-End Data Encryption



Attack points

- Storage server – Secured
- SAN – Secured
- Server / Hypervisor – Secured
- Virtual Server insider / outsider – Not secure

Questions

- Where is data decrypted/encrypted? Application or kernel
- Who generates keys? Application or OS admin
- Who owns (can access) the keys? Application or OS admin

Linux on Z Encryption of Data at Rest

End-to-End Encryption

- **dm-crypt: block device / full volume encryption**
  - Uses kernel crypto
  - Granularity: disk partition / logical volume
- **ext4 with encryption option: file system encryption**
  - Uses kernel crypto
  - Granularity: file, directory, symbolic link
- **Spectrum Scale (GPFS) with encryption option: file encryption**
  - Uses GSKit or Clic crypto libraries
  - Granularity: file
- **DB2 native encryption: data base encryption**
  - Uses GSKit crypto library

Network Encryption

- **NFS v4 with encryption option: encryption of file transport**
  - Uses kernel crypto
- **SMB v3.1: encryption of file transport**
  - Uses kernel crypto

Kernel crypto automatically uses CPACF for AES if the module **aes_s390** is loaded

GSKit and latest versions of Clic use CPACF for AES

dm-crypt & LUKS

dm-crypt Overview

— dm-crypt

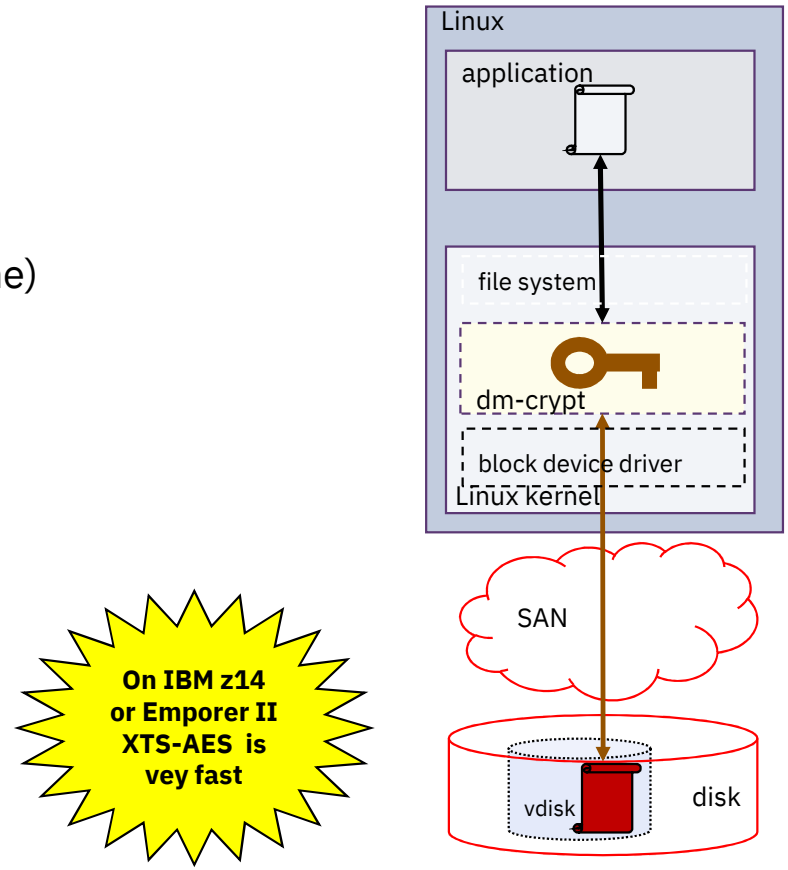
- A mechanism for end-to-end data encryption
- Data only appears in the clear when in program

— Kernel component that transparently

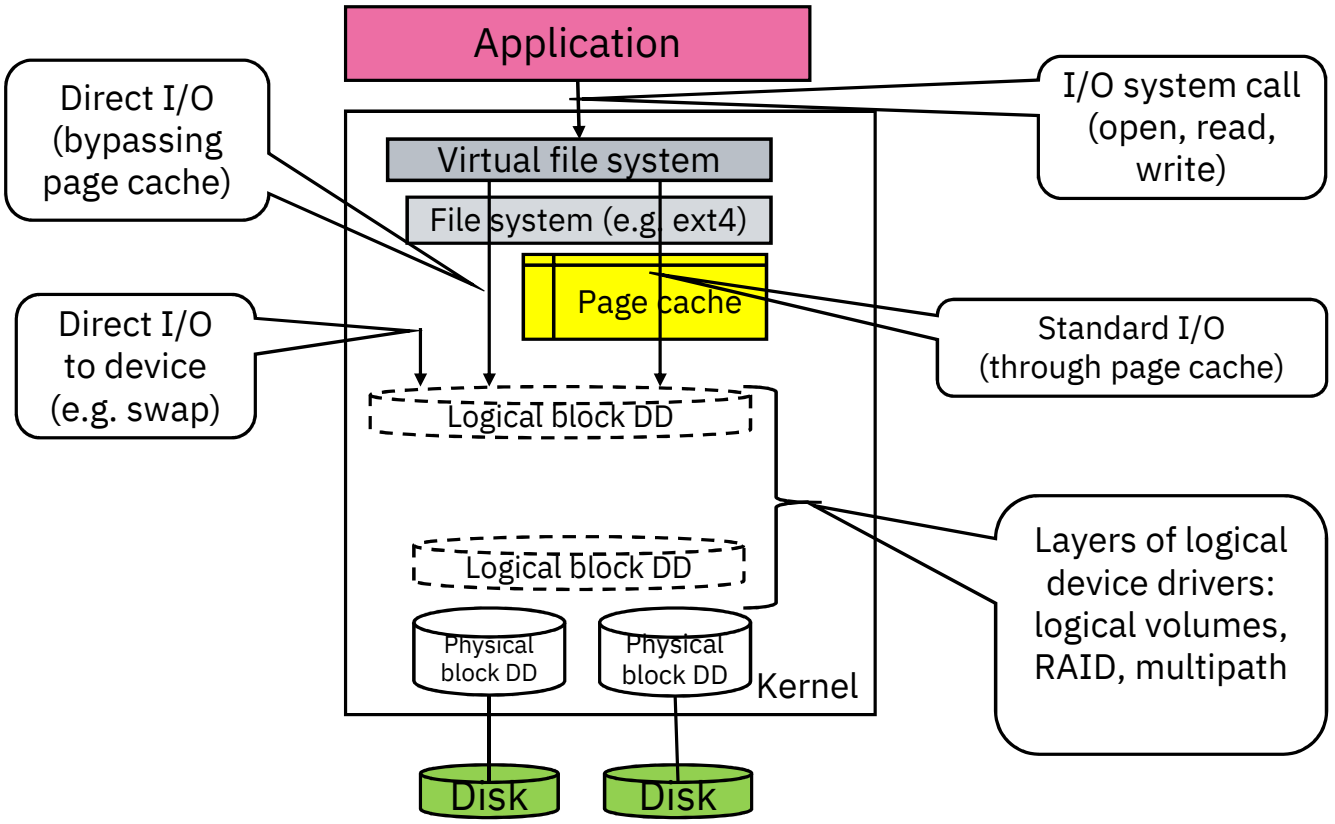
- For a whole block device (partition or LVM Logical Volume)
  - Encrypts all data written to disk
  - Decrypts all data read from disk

— How it works:

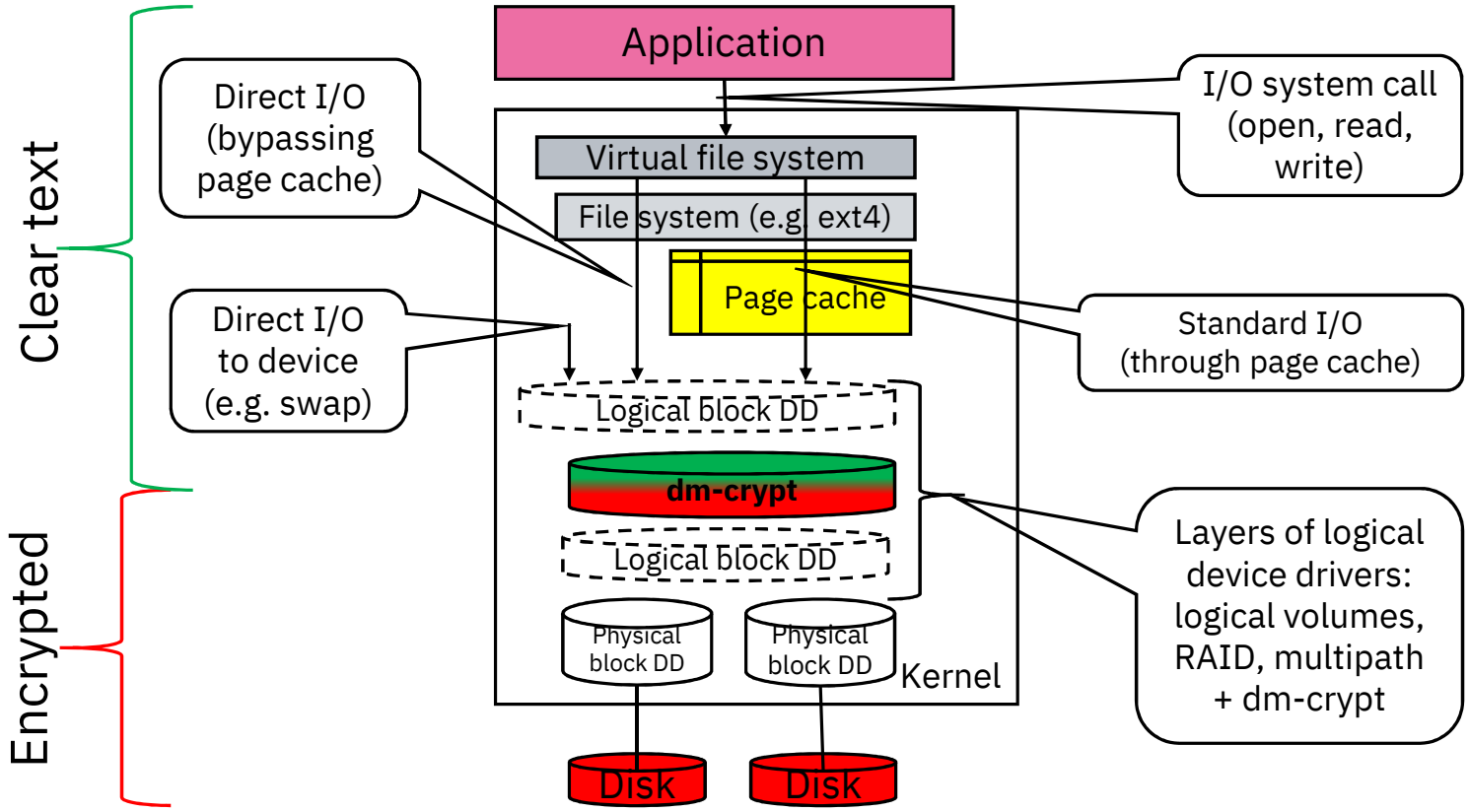
- On opening a volume
  - dm crypt is told which key and cipher to use
- dm-crypt module uses in kernel-crypto
  - can use IBM Z HW if aes_s390 module loaded
    - > AES-CBC
    - > XTS-AES (recommended)



Linux File System Stack



Linux File System Stack with dm-crypt



What Volumes can be Encrypted by dm-crypt?

YES: block devices

- Disks:
  - SCSI disks
- Partitions of
  - ECKD DASDs
  - SCSI disks
- Muti-path devices
- (LVM2) Logical volumes
- Loop back devices
- Other device mapper devices

NO:

- Full ECKD DASDs
- Network file systems like NFS
  - However, you can create a loopback device based on a file in a network file system

dm-crypt & cryptsetup - Volume Formats

dm-crypt

— Kernel module

— Supports multiple volume formats (aka types)
- plain — Focus
- LUKS
- LUKS2 — Focus
- loop-AES
- TrueCrypt / VeraCrypt

— Maintains per volume info on
- Encrypted sectors
- Sector size
- Encryption key, key size
- Nonce(s) used (e.g. IVs)
- Cipher
- Per sector IV generation algorithm

piece of data that dm-crypt encrypts/decrypts in one step

cryptsetup

— Tool to manage encrypted volumes
- Formats volumes
  - LUKS & LUKS2 types
- Opens volumes
  - plain, LUKS, LUKS2, loop-AES & TrueCrypt types
- Closes volumes
- Status
- Resize
- Format specific functions

— Communicates with dm-crypt / device mapper

dm-crypt: Linux Unified Key Setup (LUKS)

Plain

- **No formatting required**
- **No header**
- **Key & cipher name must be supplied with every open**
  - Key must be stored in a file in the file system
- **512 B, 1, 2 and 4 kB sector support**
- **Support for protected key (PAES)**

LUKS1

- **One time formatting required**
- **Header on disk**
  - Fixed binary header format
- **Key & cipher name is contained in the header**
  - Key is wrapped by a key derived from a passphrase
  - Up to 8 key-slots
- **Passphrase must be supplied with every open**
- **512 Byte sector support**
- **No support for PAES**

LUKS2

- **One time formatting required**
- **Header on disk**
  - Flexible header format (JSON)
  - Redundancy of metadata
  - Detection of metadata corruption
  - Automatic repair from metadata copy
- **Key & cipher name is contained in the header**
  - Key is wrapped by a key derived from a passphrase
  - Up to 32 key-slots
- **Passphrase must be supplied with every open**
- **512 B, 1, 2 and 4 kB sector support**
- **Support for protected key (PAES)**

How to Set-up a dm-crypt LUKS Volume

Once

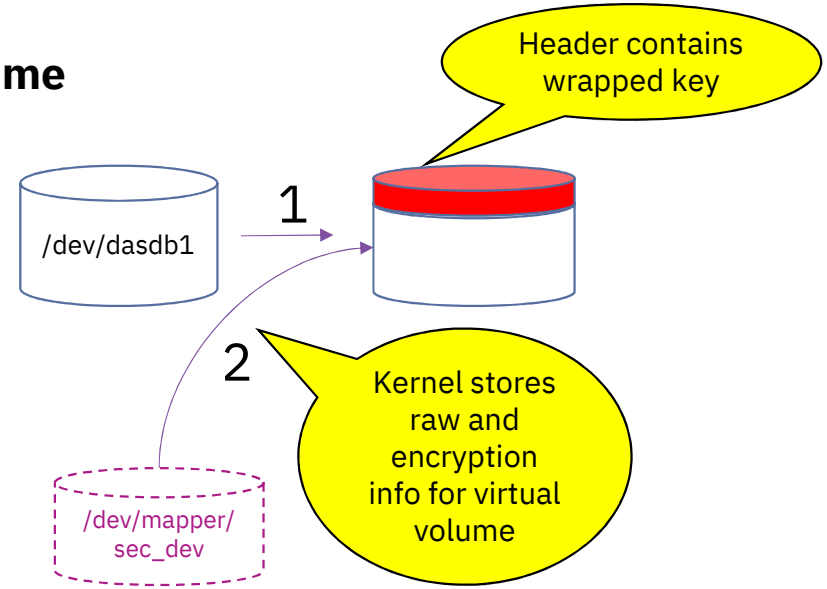
1. **Format "raw" volume as dm-crypt volume**
   - `# cryptsetup luksFormat …`
     - cipher, key length, hash, passphrase
   - Writes LUKS header to disk

With every boot

2. **Open dm-crypt volume and assign it a virtual volume name**
   - `# cryptsetup luksOpen …`
     - dm-crypt volume, virtual volume, passphrase
   - Creates virtual volume in **/dev/mapper**
   - Can be automated with **/etc/crypttab**

Business as usual

3. **Use virtual volume**
   - mkfs (or mkswap)
   - mount (or swapon)
   - Any kind of standard I/O
   - Do **not** use raw volume directly



On CPACF Performance

- CPACF performance depends on the size of buffers being en-/decrypted
- The larger the buffer the better the performance
  - Best performance with ≥ 4kB buffers
- dm-crypt always used 512 byte buffers
- Starting with kernel 4.12 and cryptsetup 2.0:
  - dm-crypt can use 4kB buffers
    - Plain mode
    - LUKS2
    - NOT: LUKS1 !

	512b	4kB
z13 or Emporer	1	1.5
z14 or Emporer II	4	13

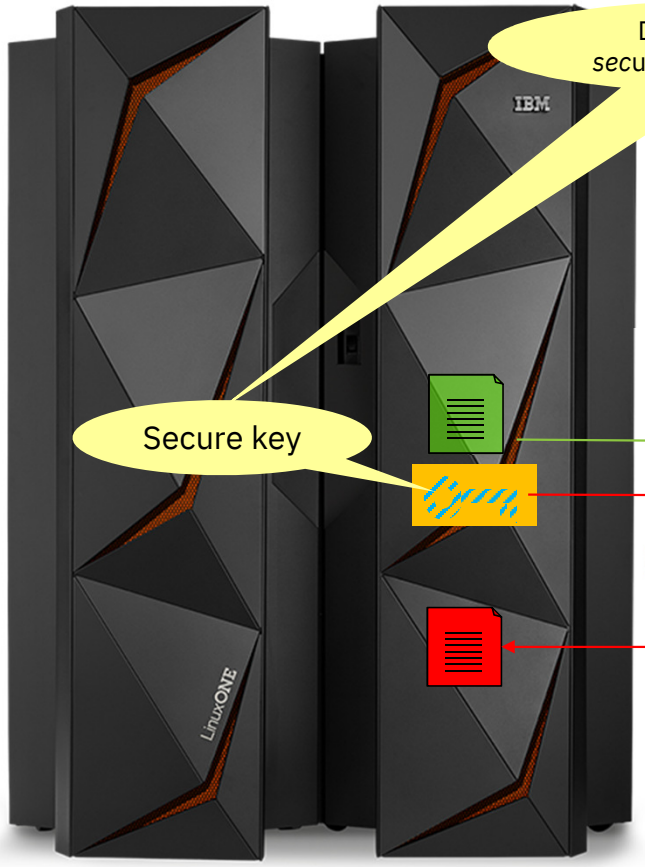
Relative AES GCM/XTS in-memory performance measured with openSSL speed test

Pervasive Encryption for Linux on Z and LinuxONE

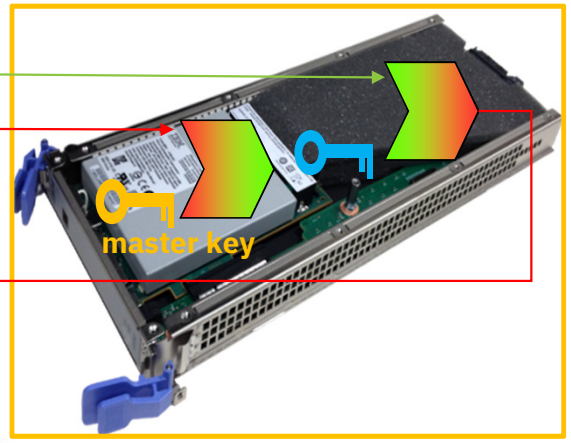
Supporting CPACF Protected Key Crypto

Clear Key vs. Secure Key Cryptography

Clear key crypto



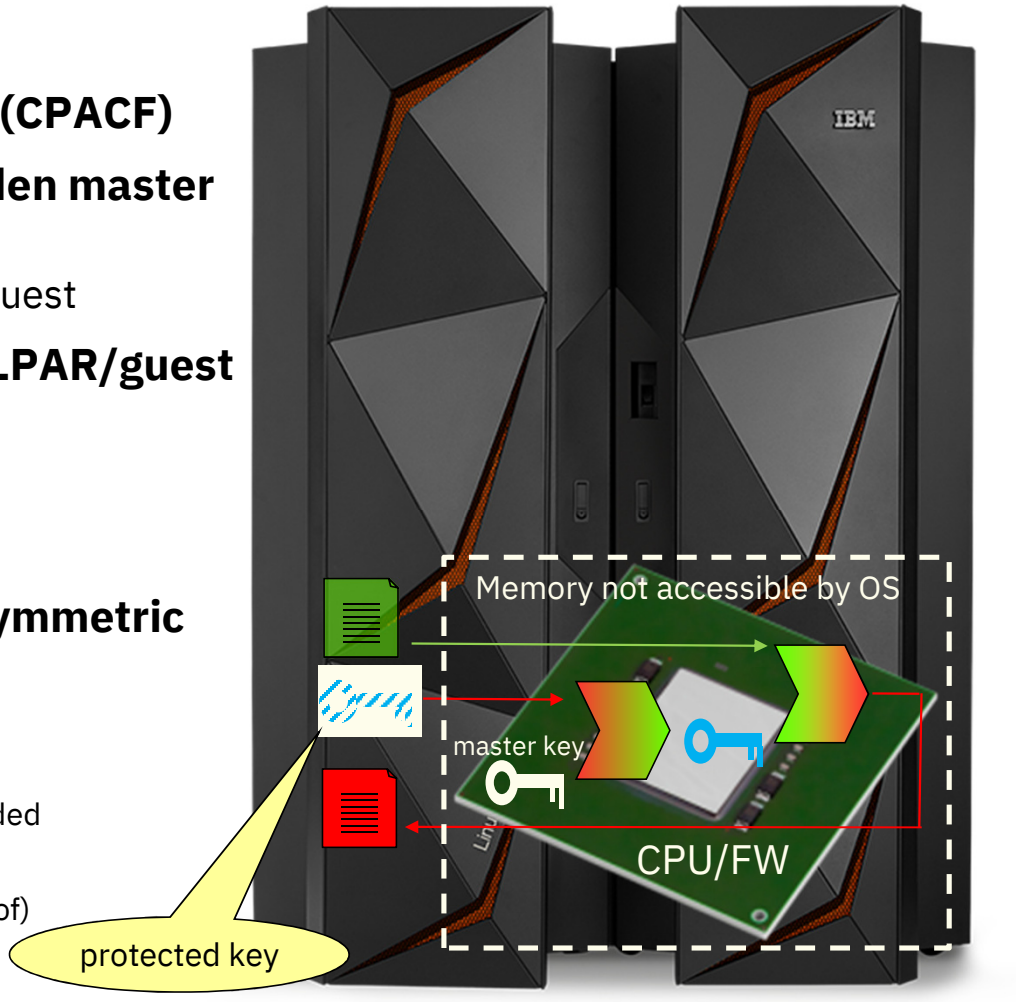
Hardware security module (HSM) → Tamper proof
E.g. Crypto Express Adapter



Secure key crypto

CPACF Protected Keys

- **Linux on Z and LinuxONE function of the CPU (CPACF)**
- **Each virtual server (LPAR or guest) has a hidden master key**
  - Not accessible from operating system in LPAR or guest
- **Protected key: a key wrapped by the hidden LPAR/guest master**
- **Protected key tokens can be generated**
  - From secure keys using CEX Adapter (secure)
- **Linux on Z and LinuxONE CPU can compute symmetric encryption for protected keys**
  - **Pro**
    - No clear keys in operating system memory
    - Fast, encryption/decryption at CPU/CPACF speed, no I/O needed
  - **Cons**
    - „Hiding“ of master key not as good as in HSM (not tamper proof)
    - Not certified as HSM
    - The LPAR/guest master key is not persistent
      - > need to store (secure) key to derive protected key from

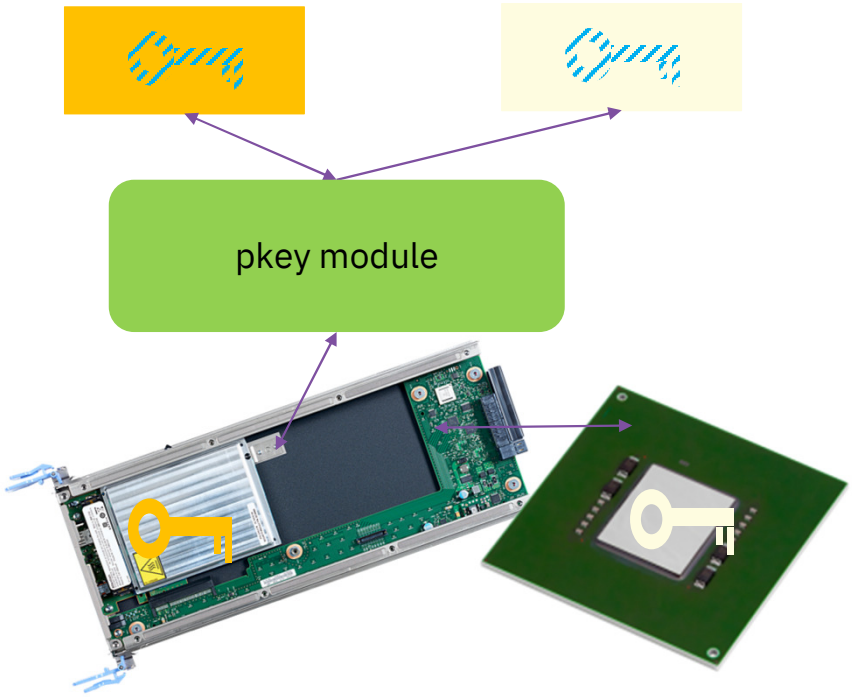


Kernel Support for Protected Keys

— **pkey kernel module**
- uses zcrypt device driver (ap)
- generates secure keys
- transforms secure keys into protected keys

— **Cipher “PAES” implements protected key AES**
- paes_s390 kernel module
- takes secure key as argument
- generates and caches equivalent protected key

— **On opening the dm-crypt volume with PAES cipher:**
- transform secure key into equivalent protected key
- protected key encryption/decryption at CPACF speed



Pervasive Encryption for Linux on Z and LinuxONE

Supporting CPACF Protected Key Crypto for dm-crypt

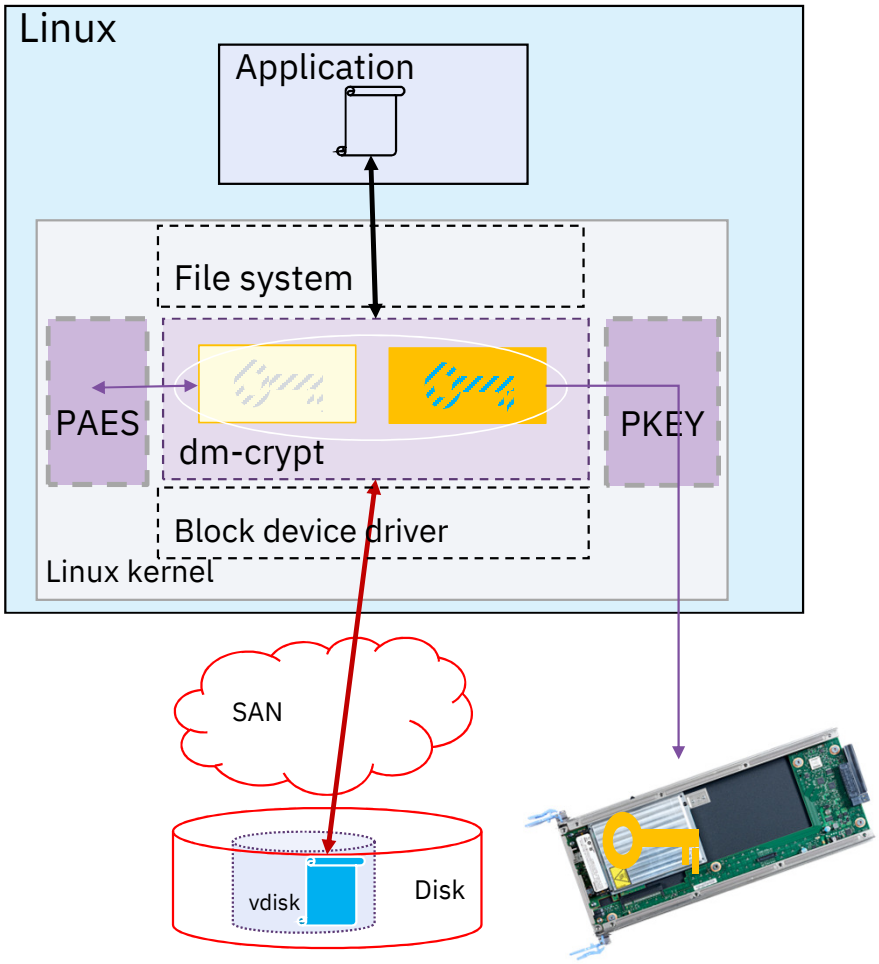
End-to-End Data at Rest Encryption with Protected Key dm-crypt

— **End-to-End data encryption**
- The complete I/O path outside the kernel is encrypted:
  - Hypervisor, adapters, links, switches, disks

— **dm-crypt**
- A mechanism for end-to-end data encryption
- Data only appears in the clear in application

— **Linux kernel component that transparently**
- For all applications
- For a whole block device (partition or LVM logical volume)
  - Encrypts all data written to disk
  - Decrypts all data read from disk

— **How it works:**
- Uses in kernel-crypto
  - Can use Linux on Z and LinuxONE CPACF protected key crypto:
    - > PAES-CBC
    - > PAES-XTS (recommended)
- Encrypted volumes must be opened before usage
  - Opening provides encryption key to kernel
  - Establishes virtual volume in /dev/mapper



Using the PAES with dm-crypt – Plain Format

— **The plain dm-crypt format does not have a header describing the disk encryption**
- No formatting required

Once

— **Generate a file with a secure key**

```
# zkey generate seckey.bin -xts
```

- Requires access to CEX[5|6]C adapter

With every boot

— **Open block device as device mapper volume**

```
# cryptsetup open --type plain --key-file seckey.bin
    --key-size 1024 --cipher paes-xts-plain64 /dev/dasdb1
    plain_enc
```

- New virtual device mapper volume /dev/mapper/plain_enc will be created
- Requires access to CEX[5|6]C adapter

Business as usual

— **Use new device mapper volume**
- (only once) Create file system:

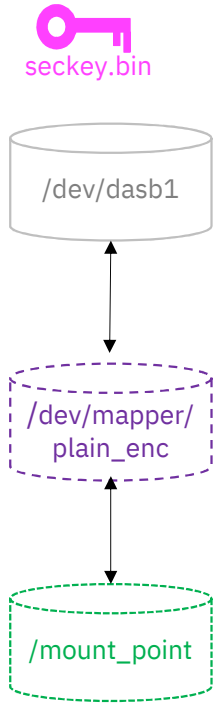
```
# mkfs.ext4 /dev/mapper/plain_enc
```

- Mount:

```
# mount /dev/mapper/plain_enc /mount_point
```

- Access:

```
# echo “hello world” > /mount_point/myfile
```



Using the PAES with dm-crypt – LUKS2 Format

Once

0: Insert new kernel modules during boot & generate secure keys

- requires access to CEX5C or CEX6C adapter
- `# zkey generate seckey.bin -xts`

Once

1: format "raw" volume as LUKS2 volume

```
# cryptsetup luksFormat --type luks2
  --cipher paes-xts-plain64
  --master-key-file seckey.bin --key-size 1024
  /dev/dasdb1
```

- Asks for a passphrase
- Writes LUKS2 header to disk

Requires cryptsetup 2.0.3

With every boot

2: Open dm-crypt volume and assign it a virtual volume name

```
# cryptsetup luksOpen /dev/dasdb1 sec_dev
```

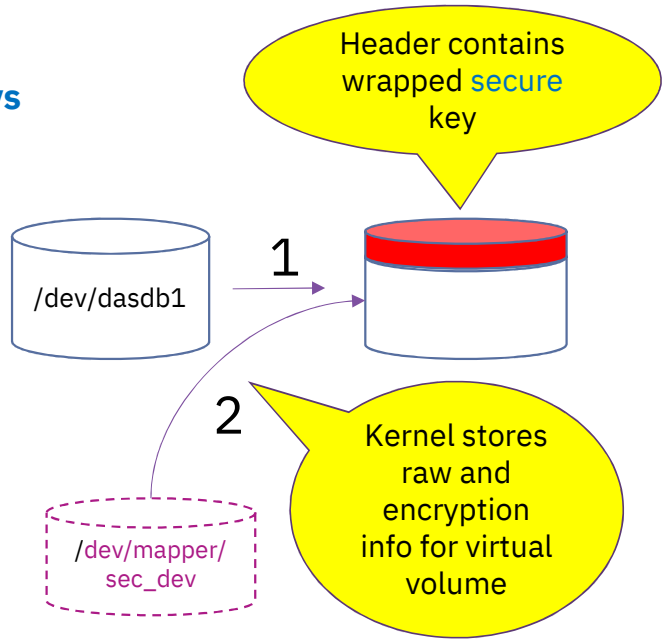
- Asks for passphrase (needs not be protected)
- Creates virtual volume in /dev/mapper

Business as usual

3: Use virtual volume

- mkfs (or mkswap)
- mount (or swapon)
- Any kind of standard I/O
- Do not use raw volume directly

Kernel translates I/O to virtual volume into I/O to raw volume and performs decryption/encryption



Key management for dm-crypt volumes

Key Management Overview

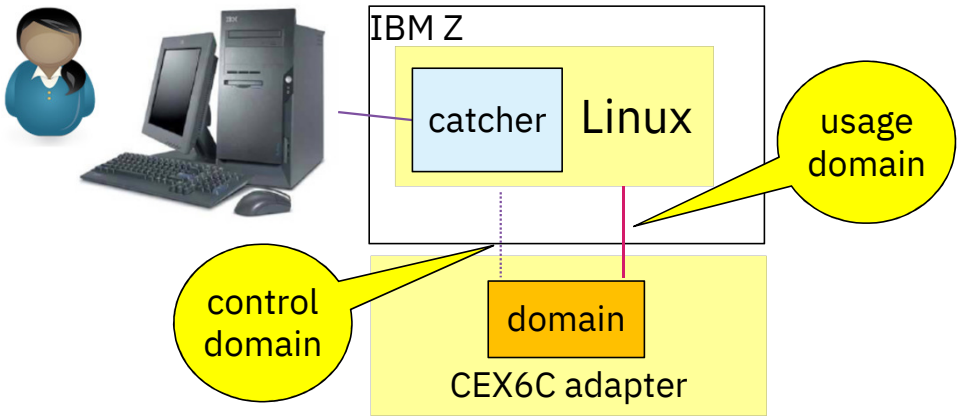
Standard Process

1. Load & Set AES Master Key in Domain of CCA co-processor(s)
2. Generate one key per volume
   - store keys in key repository
   - associate each key with volume
3. For LUKS2: format volume

Special Processes

- Back-up & restore keys
  - Back-up of volume keys / key repository
  - LUKS2: volume recovery
- Volume key roll
- CCA master key roll in Domain of CCA co-processor(s)

Master Key Configuration



Requires CCA package to be installed in Linux

— https://www.ibm.com/security/cryptocards/pciecc3/cex6s-linux-on-z
  - scroll down!

Production: use Trusted Key Entry Console (TKE)

— Configure MKs per domain per adapter (i.e. per APQN)
  - # of admins / # of key parts / admin signatures

— Setting master keys (MKs):
  - generate key parts
  - store MK parts on smart cards (to be kept in a safe)
  - load MK (from smart cards)
  - commit MK (loading complete)
  - set MK (ready to use)

The TKE can manage many domains located on many adapters plugged into multiple IBM Z systems

For test and development:

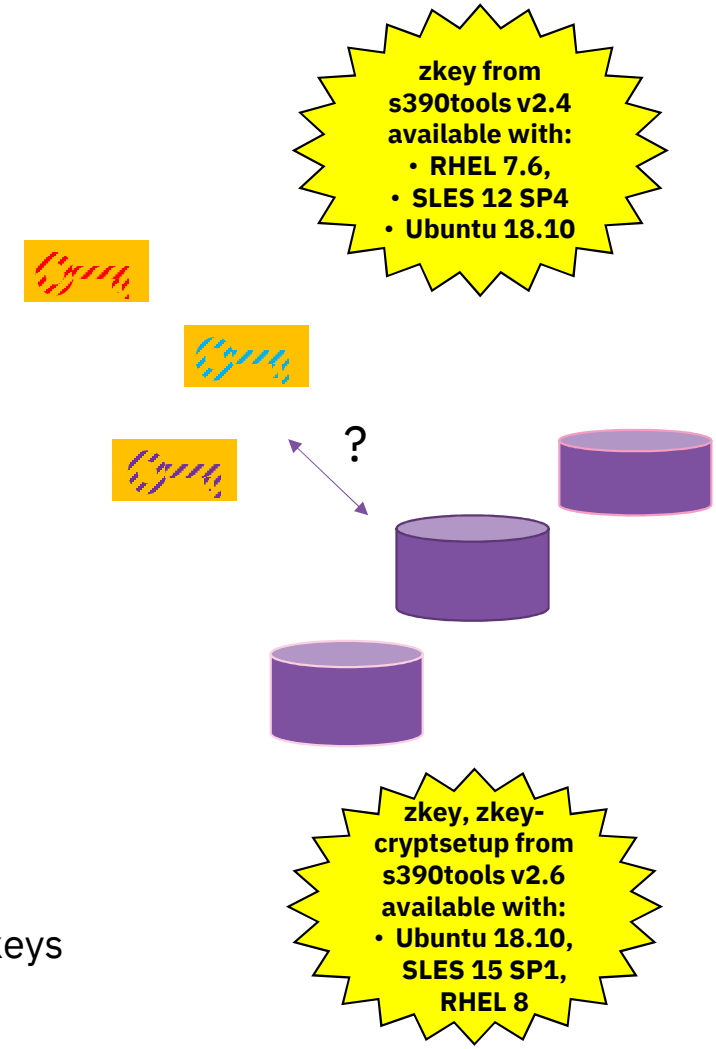
— The panel.exe tool can be used to set master keys for a local adapter domain

Key Management Tools for Protected Key dm-crypt

— **zkey tool released with s390tools 2.4.0**
- Stores secure keys (to be transformed into protected keys)
- Associating keys with
  - Volumes encrypted by that key
  - Cryptographic modes used
- Allows key generation, repository management (list, add, delete)
- Allows re-enciphering of keys if the CEX master key is rolled
- Generation of opening commands for plain format volumes
- Backup of repository easily possible with Unix tools (e.g. tar)

— **zkey tool extension released with s390tools 2.6.0**
- Adds LUKS2 volume type
- Allows generation of formatting commands for LUKS2 volumes

— **zkey-cryptsetup released with s390tools 2.6.0**
- Support for master key changes for LUKS2 volumes using protected keys



zkey Example: Key Generation & Validation

```
# zkey generate --name MySecureKey \
                --description "This is a secure key" \
                --xts --keybits 256 --apqns 03.003d \
                --volume-type luks2 --sector-size 4096 \
                --volumes /dev/dasdb1:enc_vol
# zkey list
Key                    : MySecureKey
-------------------------------------------------------------
  Description          : This is a secure key
  Secure key size      : 128 bytes
  Clear key size       : 512 bits
  XTS type key         : Yes
  Volumes              : /dev/dasdb1:enc_vol
  APQNs                : 03.003d
  Key file name        : /etc/zkey/repository/MySecureKey.skey
  Sector size          : 4096 bytes
  Volume type          : luks2
  Verification pattern : d4ce4d36c8dc9a43648fbb65c79b2d35
                         2171b893d817256ded4a3c94186ee193
  Created              : 2018-10-26 08:52:23
  Changed              : (never)
  Re-enciphered        : (never)
```

```
# zkey validate
Key                    : MySecureKey
-------------------------------------------------------------
  Status               : Valid
  Description          : This is a secure key
  ...
  1 keys are valid, 0 keys are invalid, 0 warnings
```

zkey Examples: Geneneration of Cryptsetup Commands

Generate cryptsetup commands for volumes in plain format

```
# zkey cryptsetup –volume-type plain
cryptsetup plainOpen --key-file '/etc/zkey/repository/mykey3.skey' \
--key-size 1024 --cipher paes-xts-plain64 --sector-size 4096 \
/dev/loop3 my_sec_vol3
```

Generate cryptsetup commands for volumes in LUKS2 format

```
# zkey cryptsetup –volume-type luks2
cryptsetup luksFormat --type luks2 --master-key-file '/etc/zkey/repository/MySecureKey.skey' \
--key-size 1024 --cipher paes-xts-plain64 --sector-size 4096 /dev/dasdb1
zkey-cryptsetup setvp /dev/dasdb1
```

If you want to open the volume via /etc/crypttab you may want to add the –pbkd pbkdf2 option -- see later.

CEX CCA Master Key Roll

Changing a CEX CCA master key

- Rarely needed
- Requires that all secure keys are re-enciphered
  - Otherwise they become invalid after next change of that master key
- Can be done concurrently to volume usage
- As long as system is neither migrated nor suspended

On Trusted Key Entry Console (TKE)

- For adapter domain (APQN) used by dm-crypt
- Load & set new AES master key
  - Do not do this twice before having re-encipherd all keys!

Re-enciphering volume keys

- **(1) in zkey repository**
  - `# zkey reencipher –apqn <apqns>`
  - Where <apqns> are the addresses (adapter ID, domain ID pairs) of the adapters where the AES master keys where changed.
- zkey also provides a per key file option to re-encipher keys

- **(2) in headers of LUKS2 volumes**
  - After formatting volume in LUKS2 format
    - `# zkey-cryptsetup setvp <raw volume>`
  - After loading and setting new master key on TKE:
    - `# zkey-cryptsetup reencipher <raw volume>`
  - For more options see man page of zkey-cryptsetup

Best Practices with Encrypted Volumes

Best Practices with (Protected Key) dm-crypt

— **Do not share volume keys**
- Generate a new (random) key for each dm-crypt volume

— **Use /etc/crypttab**
- To configure automated opening of volumes

— **Use dm-crypt volumes as LVM physical volumes**
- Allows transparent data migration

— **Protection against data loss**
- To deal with key loss (plain format)
  - Backup secure key
- To deal with LUKS superblock corruption:
  - back-up dm-crypt superblock (LUKS Header)
- To deal with HSM loss:
  - Make sure you have a back-up adapter with common master key
  - Or generate secure key from clear key in clean room environment and store clear key in safe

/etc/crypttab

- **Configuration file to describe how dm-crypt volumes are to be opened**
- **Will be read and interpreted before /etc/fstab**
- **Format**
  - Each line describes a dm-crypt volume:
    - **<dm-crypt volume> <path of block-device> <password file>|none [options]**

      for the plain format, the password file contains a key
    - Options may be set to describe volatile swap and tmp volumes
  - Example lines:

```
luks_vol   /dev/dasdb1   /root/PWs/sec_dev.pw  luks
Swap       /dev/dasdc1   /dev/urandom          swap,cipher=aes-xts-plain64,size=256
plain_vol  /dev/dasdd1   /root/seckey.bin      plain,cipher=paes-xts-plain64,hash=plain,size=1024
```

Generating crypttab entries with zkey

— **zkey can generate crypttab entries matching key attributes**

— **For plain format**

- **`# zkey crypttab --volume-type PLAIN`**
  ```
  my_sec_vol3 /dev/dasdc1 /etc/zkey/repository/mykey3.skey  \
         plain,cipher=paes-xts-plain64,size=1024,hash=plain,sector-size=4096
  ```
- **Note:** a late version of systemd (commit a9fc640) is required to support 4096 byte sectors

— **For LUKS2 Format**

- **`# zkey crypttab --volume-type LUKS2`**
  ```
  my_sec_vol1 /dev/loop1
  ```
- Append path of file that contains passphrase - this file need not be secret!

- Out-Of-Memory may happen when opening LUKS2 volumes at system start up:
  - Use --pbkdf pbkdf2 option when formatting disk to avoid OOM due to excess memory consumption of Argon2i

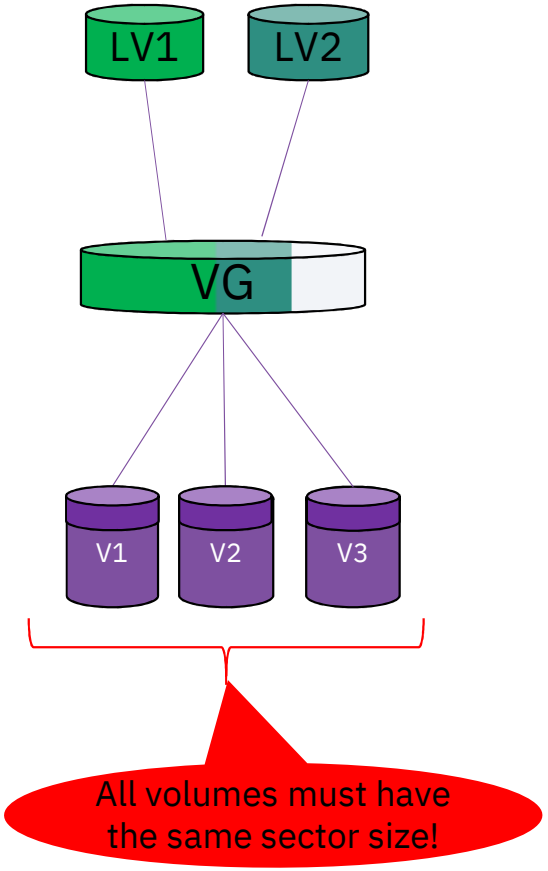
Good/Best Practice: Logical Volumes

Use logical volumes:

- allows multi-pathing
- striping, SW-RAID
- dynamic growth of volumes

How to build logical volume?

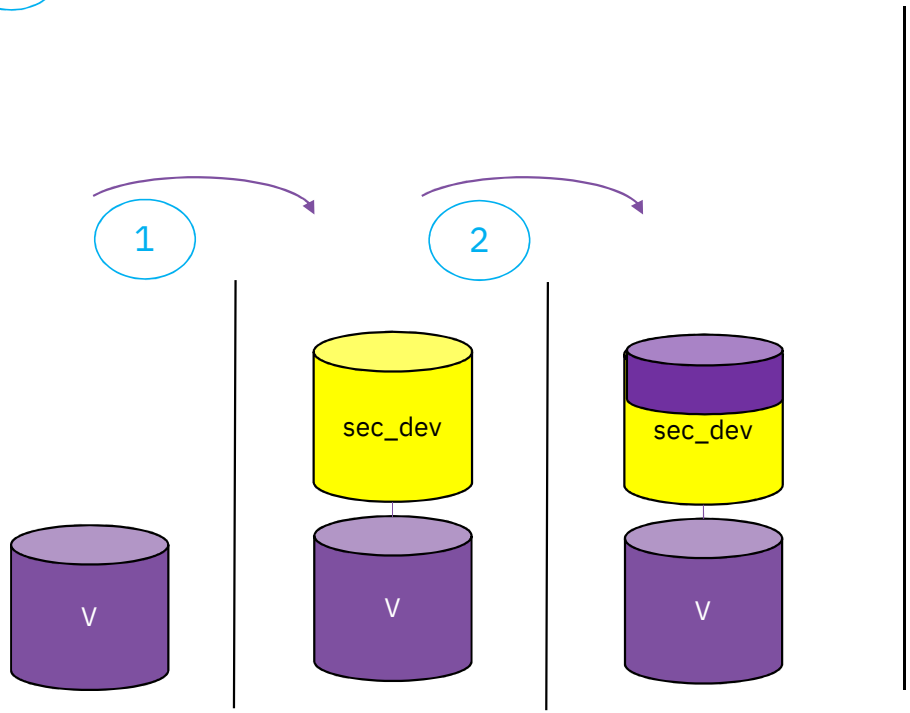
- Turn block device into “LVM physical volume”
  `# pvcreate <device>`
- Create a volume group
  `# vgcreate <volume group> <physical volume>`
- Add further physical volumes to volume group as needed
  `# vgextend <volume group> <physical volume>`
- Create logical volume
  `# lvcreate -L <size> <volume group> <logical volume>`
- Logical volume can now be accessed as
  `/dev/mapper/<volume group>-<logical volume>`

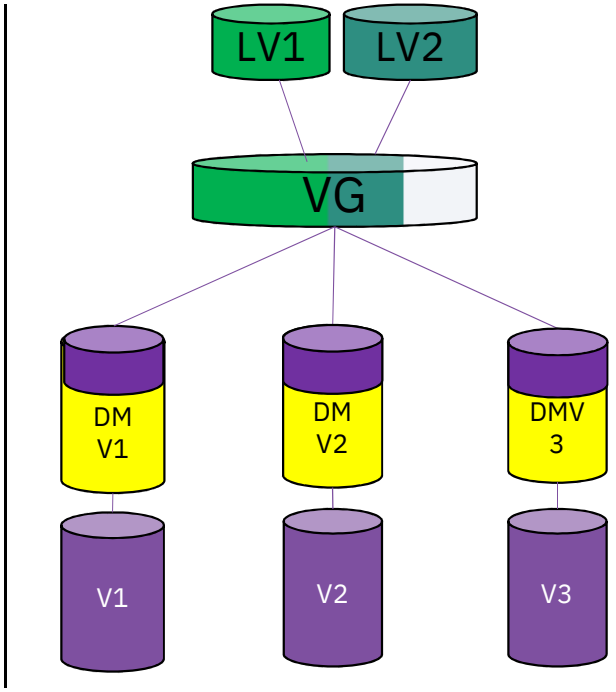


Using dm-crypt Volumes as Physical Volumes

— Use virtual dm-crypt devices as input to vgcreate:

1. `# cryptsetup luksOpen <device> sec_dev`
2. `# pvcreate /dev/mapper/sec_dev`





logical volumes

volume group

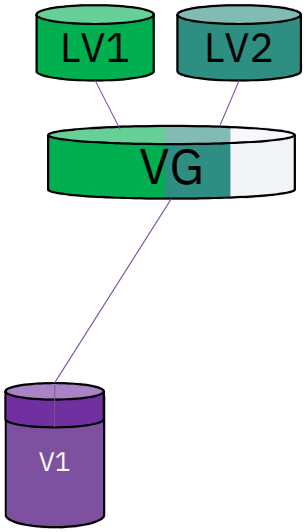
dm-crypt volumes with physical volume label

raw volumes

Migrate Data from Plaintext Volume to Encrypted Volume

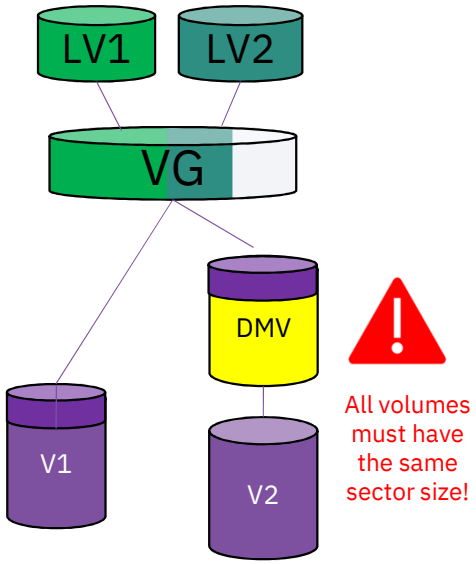
This works transparently to applications accessing LVs

0: plain text volume



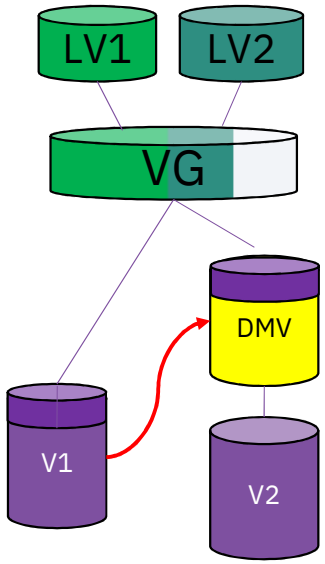
1: add dm-crypt based physical volume to volume group:

```
# vgextend VG DMV
```



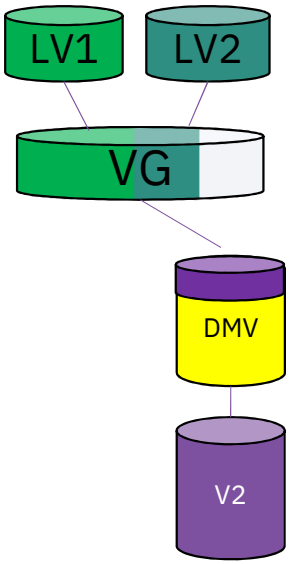
2: migrate data from V1 to DMV:

```
# pvmove V1 DMV
```



3: remove V1 from VG:

```
# vgreduce VG V1
# pvremove V1
```



Restrictons on using pvmove

— **File systems are aware of physical block sizes of their volumes**
  - thus of dm-crypt sector sizes

— **You must not pvmove data from a volume with a smaller sector size to a volume with a larger sector size**

— **E.g. do not**
  - Move from SCSI volume (512 bytes) to DASD (4kB)
  - Move from unencrypted SCSI volume (512 bytes) to dm-crypt volume with 4K sector size (4 kB)
  - Move from dm-crypt volume with default sector size (512 bytes) to dm-crypt volume with 4K sector size

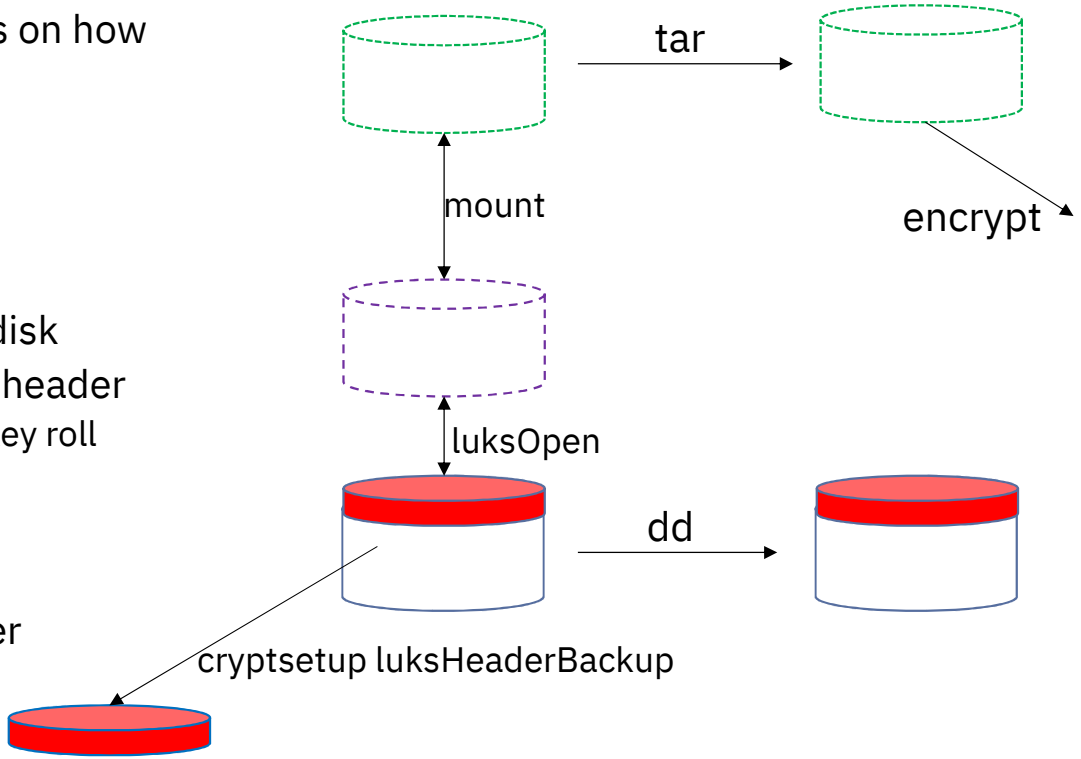
— **If your file systems in the logical volumes where created with the 4k sector size options the restrictions do not apply**
  - `# mkfs.ext4 -b 4096 <device>`
  - Default file system block sizes also depend on volume size
    - File systems on large volumes typically use 4 kb block sizes

Back-ups of Encrypted Data

— **Back-ups at file system level**
- E.g. with tar
- Back-up is plain text, encryption depends on how back up is stored

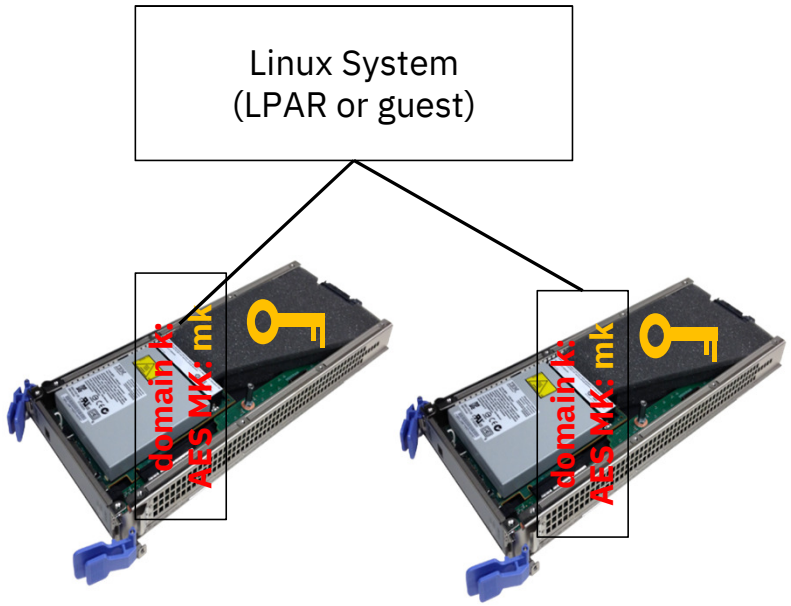
— **Raw disk image**
- E.g. dd from /dev/dasdb1
- Back-up is encrypted exactly as original disk
- LUKS: self-contained as it contains LUKS header
  - Don’t forget to re-encipher on CCA master key roll

— **Always a good idea:**
- LUKS: make a back-up of the LUKS header
  - Protects against header corruption
- Plain format: always back-up key file



Redundant Crypto Adapters

- **Attach two domains located on two different CCA co-processors to Linux system**
- **Configure both domains with the same AES master key**



The Linux kernel (pkey module)

- Uses any of those domains to generate a protected key
- Will not use a domain with a ”wrong” master key
- If an adapter failure occurs it will transparently failover to the other adapter

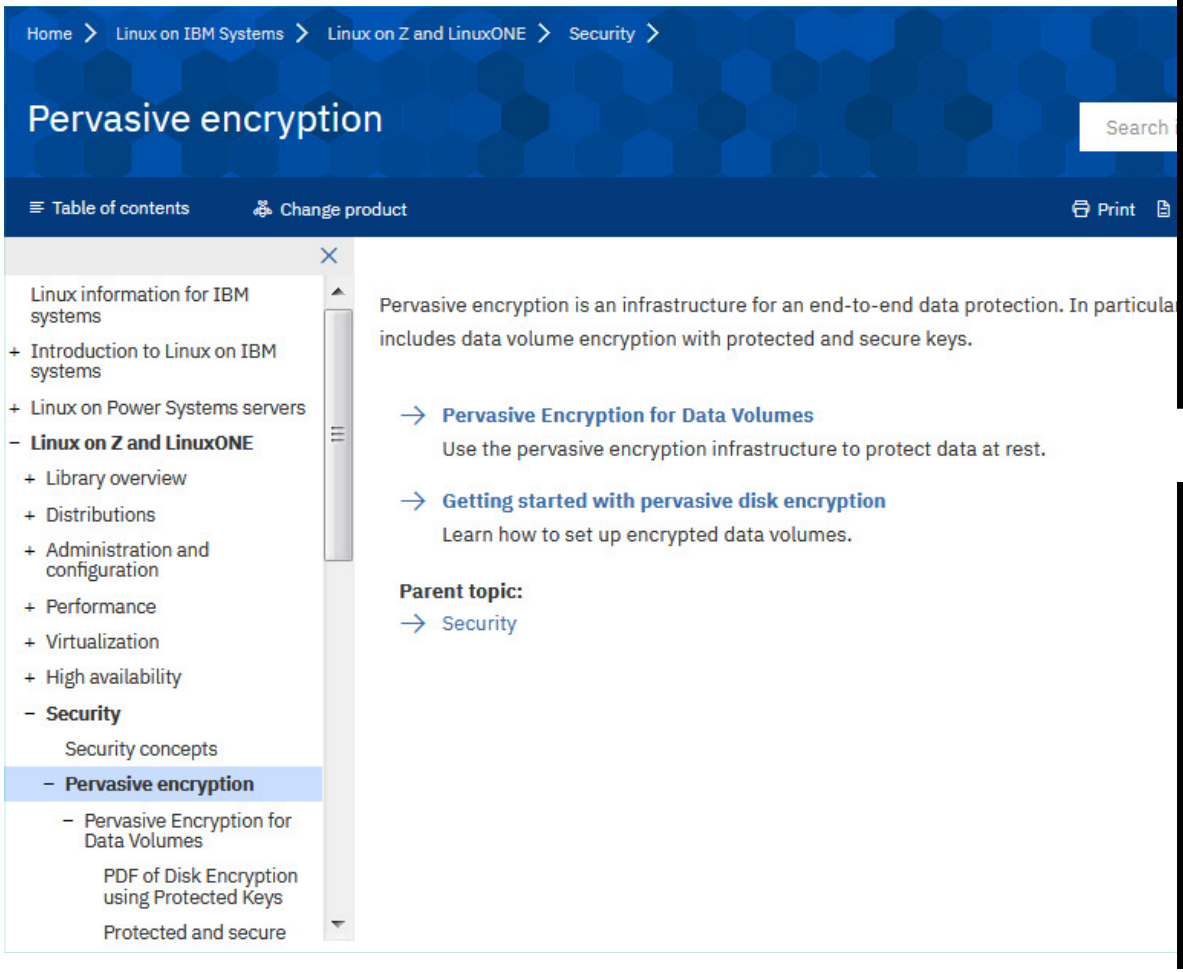
PAES Support for dm-crypt Overview

Function	Components Upstream versions	Distribution releases
PAES support for dm-crypt: plain format	kernel 4.11: pkey, paes_s390 s390tools 1.39: zkey	RHEL 7.5, SLES 12 SP4, SLES15, Ubuntu 18.04
master key rolling support for PAES keys for dm-crypt with plain format	kernel 4.11: pkey, paes_s390 s390tools 1.39: zkey	RHEL7.5, SLES 12 SP4, SLES 15, Ubuntu 18.04
fast dm-crypt using 4kB sectors	kernel 4.12 cryptesetup 2.0.0	RHEL7.6, SLES 15, Ubuntu 18.04
4kB sector support for plain volumes in /etc/crypttab	systemd patches	RHEL 8 SLES 15SP1 Ubuntu 18.10
PAES key repository for dm-crypt with plain format	s390tool 2.4: zkey	RHEL7.6, SLES 12 SP4, Ubuntu 18.10
PAES support for dm-crypt: LUKS2 format	kernel 4.11: pkey, paes_s390 cryptsetup 2.0.3	RHEL7.6, SLES 15SP1, Ubuntu 18.10
PAES key repository for dm-crypt with LUKS2 format	s390tools 2.6: zkey	RHEL 8, SLES 15SP1, Ubuntu 18.10
master key rolling support for PAES keys for dm-crypt with LUKS2 format	s390tools 2.6: zkey-cryptsetup	RHEL 8, SLES 15SP1, Ubuntu 18.10

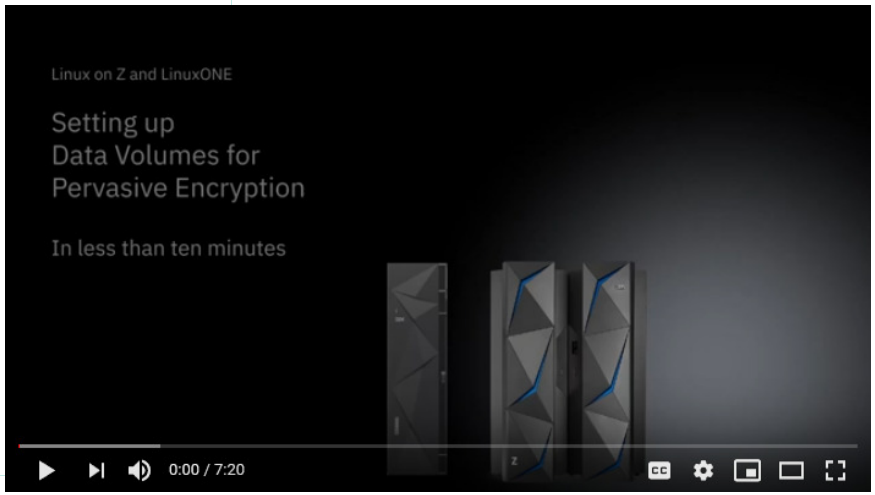
Documentation

https://youtu.be/jDK3ZwEdX4I

http://**ibm.com**/support/knowledgecenter/linuxonibm/liaaf/lnz_r_crypt.html



https://youtu.be/t2Ph_h0LcsQ



Summary

Pervasive Encryption for Linux provides

- Fast and consumable data protection for data in-flight and data at-rest
- Transparent fast encryption for data at rest
- Extended security for data at-rest with protected key dm-crypt
- A trusted computing environment for sensitive appliances through Secure Service Containers

Questions?

Notices and disclaimers

Notices and disclaimers continued

- 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 about this publication 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. IBM does not warrant the quality of any third-party products, or the ability of any such third-party products to interoperate with IBM's products. **IBM expressly disclaims all warranties, expressed or implied, including but not limited to, the implied warranties of merchantability and fitness for a purpose.**
- The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents, copyrights, trademarks or other intellectual property right.
- IBM, the IBM logo, ibm.com and [names of other referenced IBM products and services used in the presentation] are 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: www.ibm.com/legal/copytrade.shtml