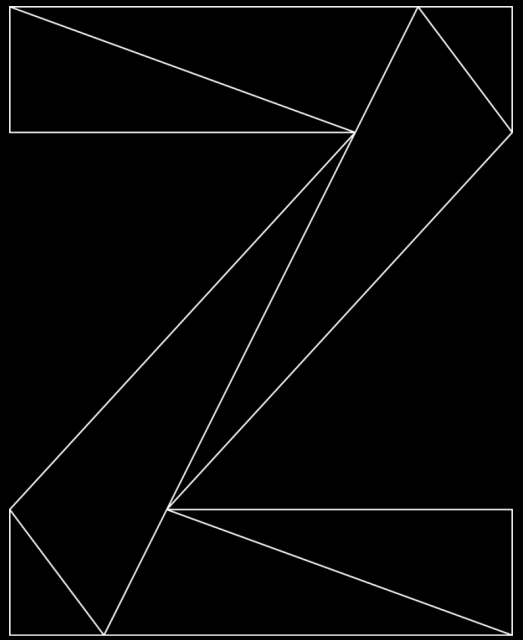
Pervasive Encryption with Linux on Z

郝庆丰/Hao QingFeng -- haoqf@cn.ibm.com

Software Developer for KVM on Z



IBM

Trademarks

The following are trademarks of the International Business Machines Corporation in the United States and/or other countries.

CICS*	Global Business Services*	MQ*	SPSS*	XIV*	z/VSE*
Cognos*	IBM*	Parallel Sysplex*	System Storage*	zEnterprise*	
DataStage*	IBM (logo)*	QualityStage	System x*	z/OS*	
DB2*	InfoSphere	Rational*	Tivoli*	z Systems*	
GDPS	Maximo*	Smarter Cities	WebSphere*	z/VM*	

* Registered trademarks of IBM Corporation

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.
IT Infrastructure Library is a Registered Trade Mark of AXELOS Limited.
ITIL is a Registered Trade Mark of AXELOS Limited.
Linear Tape-Open, LTO, the LTO Logo, Ultrium, and the Ultrium logo are trademarks of HP, IBM Corp. and Quantum in the U.S. and other countries.
Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.
Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.
Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation 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.
Cell Broadband Engine is a trademark of Sony Computer Entertainment, Inc. in the United States, other countries, or both and is used under license therefrom.
UNIX is a registered trademark of The Open Group in the United States and other countries.
VMware, the VMware logo, VMware Cloud Foundation, VMware Cloud Foundation Service, VMware vCenter Server, and VMware vSphere are registered trademarks or trademarks of VMware, Inc. or its subsidiaries in the United States and/or other jurisdictions.
Other product and service names might be trademarks of IBM or other companies.

Notes:

Performance is in Internal Throughput Rate (ITR) ratio based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput improvements equivalent to the performance ratios stated here.
IBM hardware products are manufactured from new parts, or new and serviceable used parts. Regardless, our warranty terms apply.
All customer examples cited or described in this presentation are presented as illustrations of the manner in which some customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics will vary depending on individual customer configurations and conditions.
This publication was produced in the United States. IBM may not offer the products, services or features discussed in this document in other countries, and the information may be subject to change without notice. Consult your local IBM business contact for information on the product or services available in your area.
All statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only.
Information about non-IBM products is obtained from the manufacturers of those products or their published announcements. IBM has not tested those products and cannot confirm the 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.
Prices subject to change without notice. Contact your IBM representative or Business Partner for the most current pricing in your geography.
This information provides only general descriptions of the types and portions of workloads that are eligible for execution on Specialty Engines (e.g, zIIPs, zAAPs, and IFLs) ("SEs"). IBM authorizes customers to use IBM SE only to execute the processing of Eligible Workloads of specific Programs expressly authorized by IBM as specified in the “Authorized Use Table for IBM Machines” provided at www.ibm.com/systems/support/machine_warranties/machine_code/aut.html (“AUT”). No other workload processing is authorized for execution on an SE. IBM offers SE at a lower price than General Processors/Central Processors because customers are authorized to use SEs only to process certain types and/or amounts of workloads as specified by IBM in the AUT.

Agenda

- **Background**
- Use HSM and CPACF to protect data with IBM Z
- Usage in Linux
- Upcoming features
- Application in Blockchain

Keys – Your Most Critical Data

Data protection and Security are Business Imperatives

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

Nearly **4 million** records stolen per day,

157,364 per hour

and **2,623** per minute. [1]

24%

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

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

Of the **7 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/

Encrypt the data

Access control within your system

- Authentication
  - Passwords, ssh keys, MFA, …
- Authorization
  - Users/goups/roles & rwx, ACLs, SELinux policies, …
- What about the almighty root user?
  - May give up privileges, but can you enforce this?
  - May be negligent
  - May be threatened
- What about vulnerabilities?
  - May circumvent authentication / authorization

Access control 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?

- Is there any effective access control?
- Do you own access control?

You better encrypt that data!

Cyptography

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

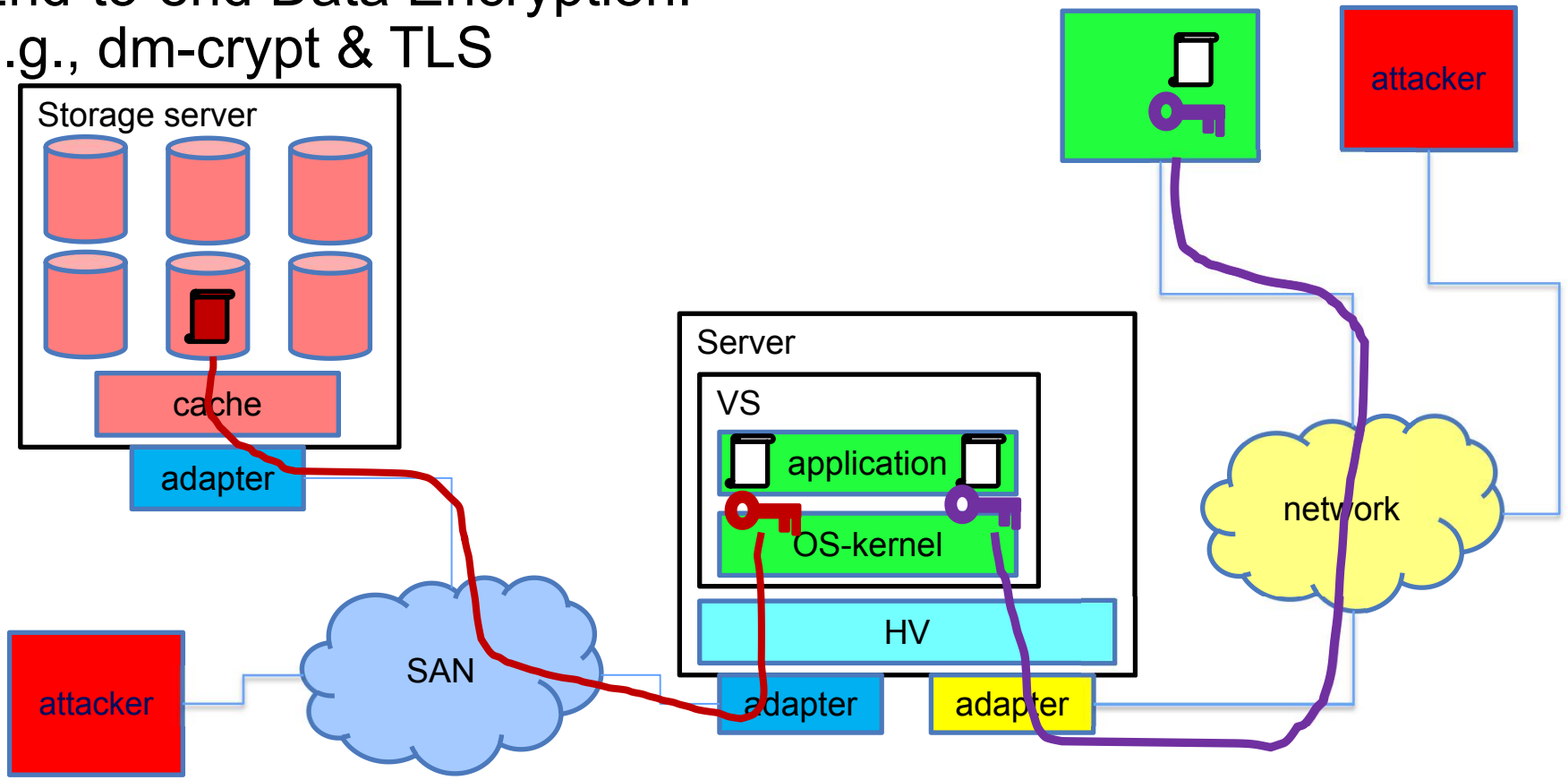
- protect you from your keys getting stolen

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

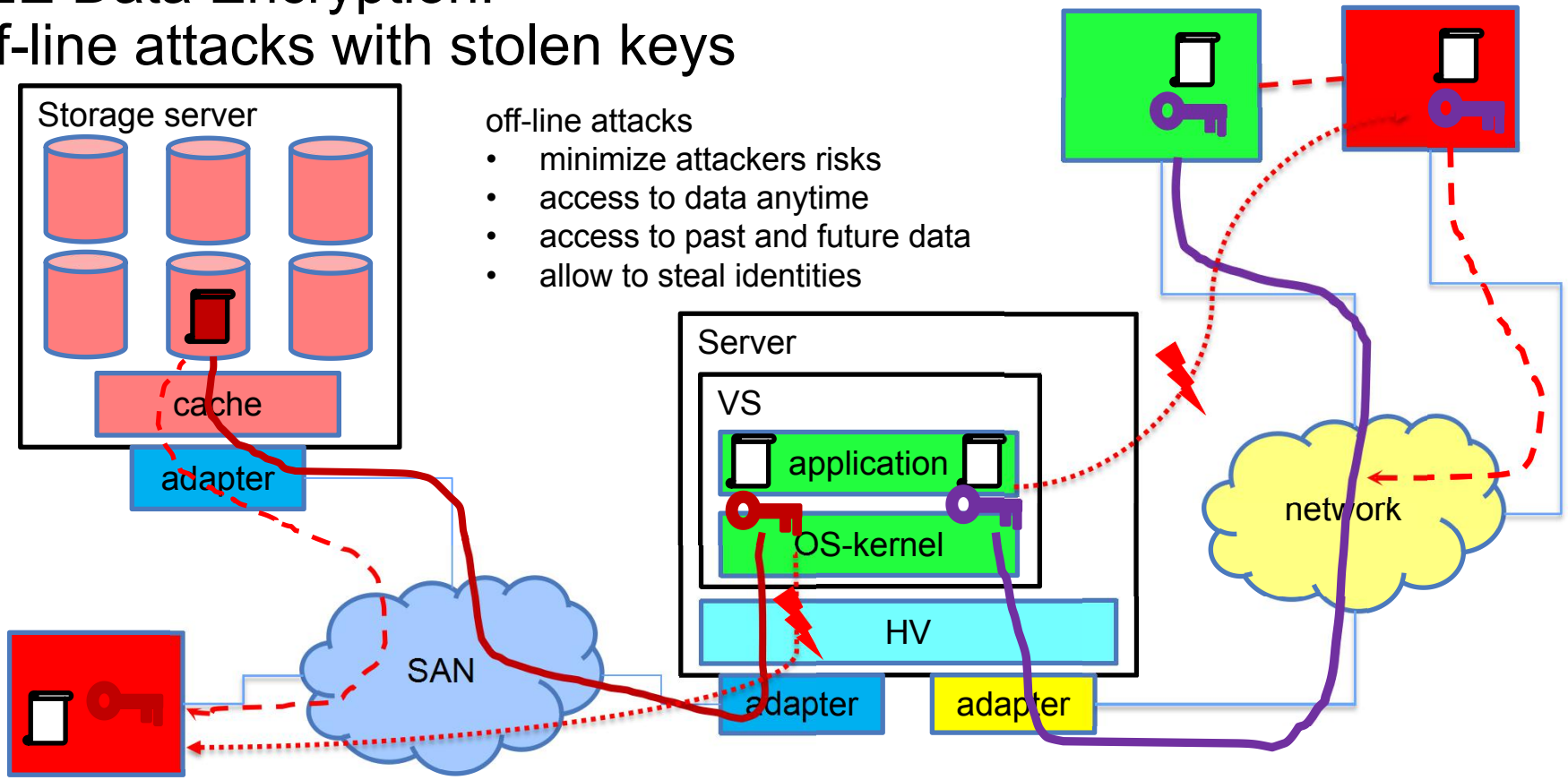
SPECTRE

MELTDOWN

End-to-end Data Encryption: e.g., dm-crypt & TLS



E2E Data Encryption: off-line attacks with stolen keys



Agenda

- Background
- **Use HSM and CPACF to protect data with IBM Z**
- Usage in Linux
- Upcoming features
- Application in Blockchain

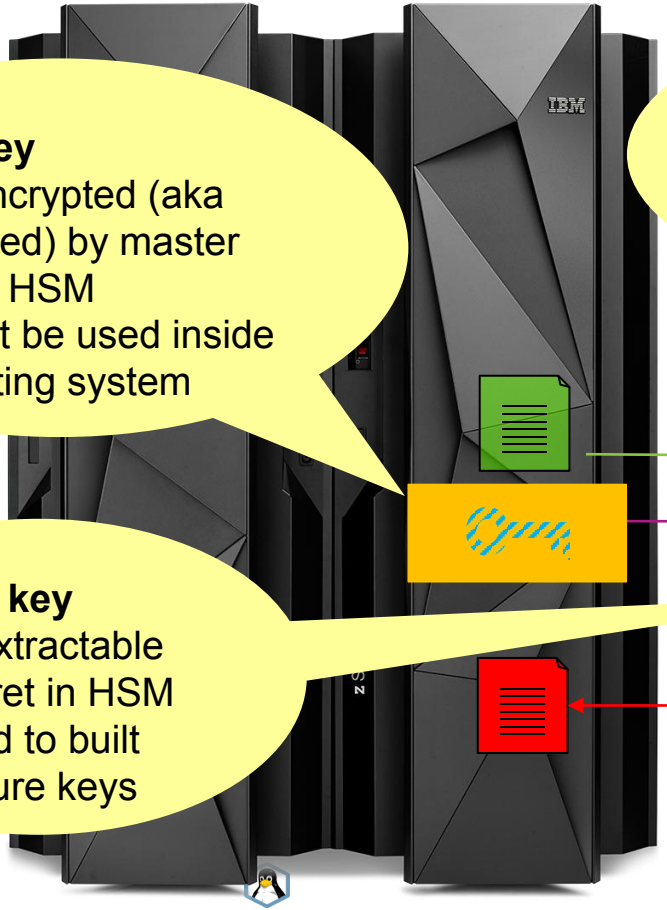
Protection Against Key Theft

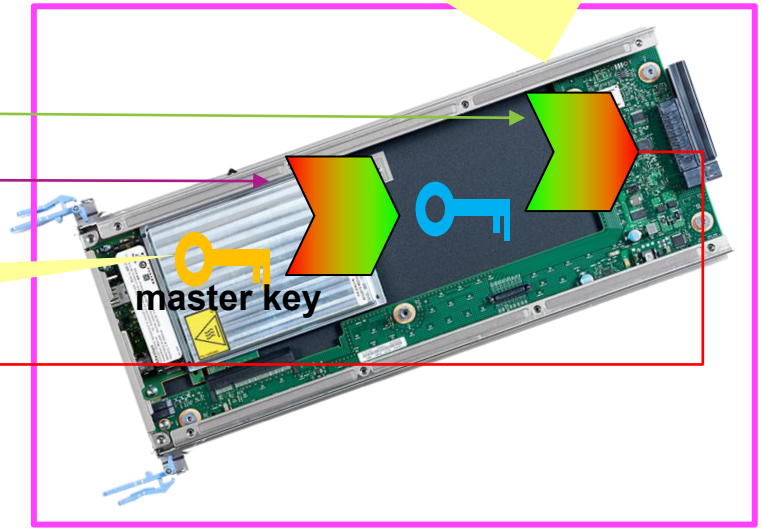
Hardware Security Modules (HSMs) & Secure Keys

secure key
- key encrypted (aka wrapped) by master key of HSM
- cannot be used inside operating system

master key
- unextractable secret in HSM
- used to built secure keys

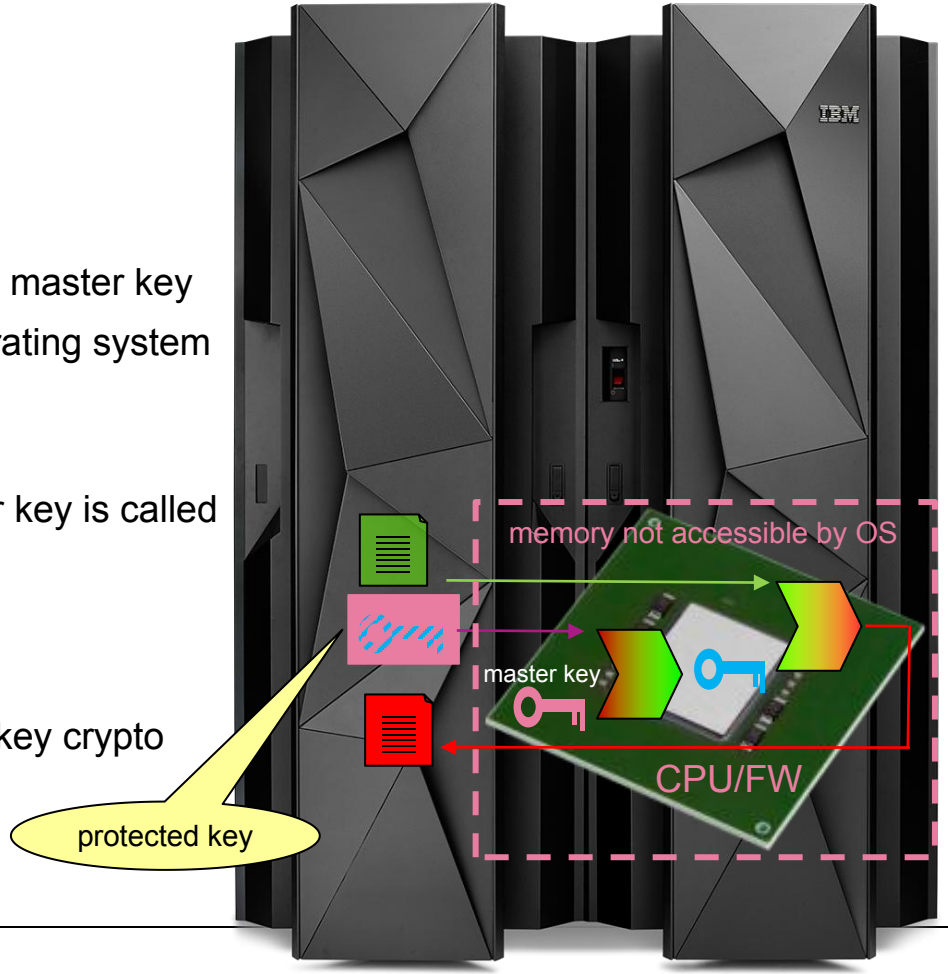
Hardware Security Module (HSM)
- special crypto HW
- tamper proof
- contains unextractable secret: master key





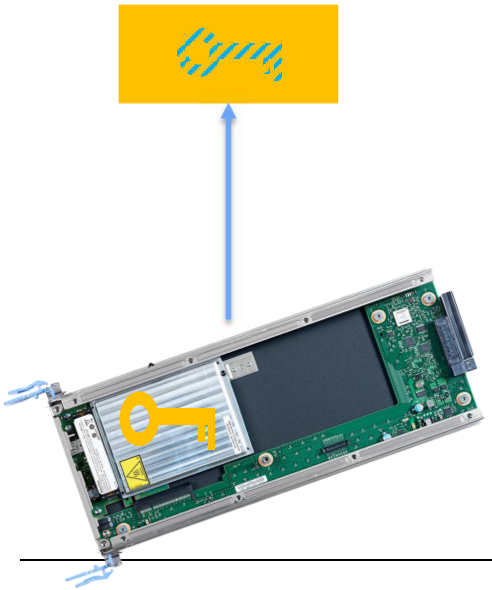
CPACF Protected Keys

- IBM Z function of the CPU (CPACF)
  - for symmetric encryption/decryption
  - MAC functions
- each virtual server (LPAR or guest) has a *hidden* master key
  - hidden master key is not accessible from operating system in LPAR or guest
  - is recycled after every boot
- a key wrapped by the hidden LPAR/guest master key is called a *protected* key
- protected key tokens can be generated
  - from secure keys using CEX Adapter (secure)
- Protected key crypto is much faster than secure key crypto
  - no I/O needed
  - almost as fast as CPACF with clear keys

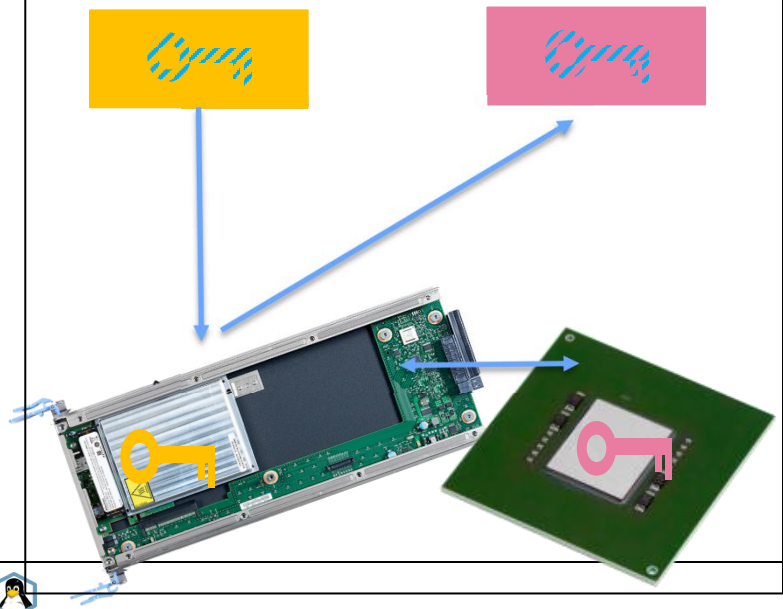


How to Generate Protected Keys

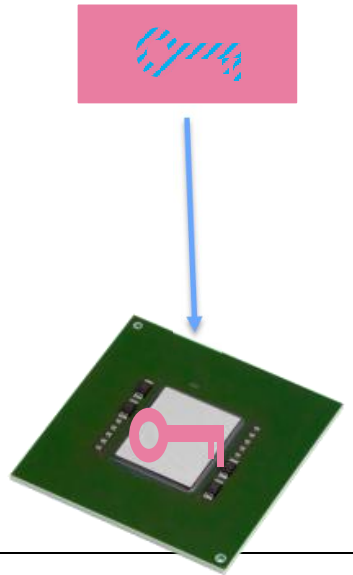
1. Generate secure key with help of CEX Adapter



2. Transform secure key into protected key with help of CEX Adapter which secretly negotiates with CPU&FW



3. use protected key with help of CPU



Secure Key Concepts

A secure key object comprises
- an encrypted clear key
- key attributes describing
  - Security measure, cipher
  - Allowed usage (encrpt, wrap, sign, ...)
  - Allowed export
  - Possibly a reference to master key
  - …

A secure key object
- prevents separation of key from its attributes

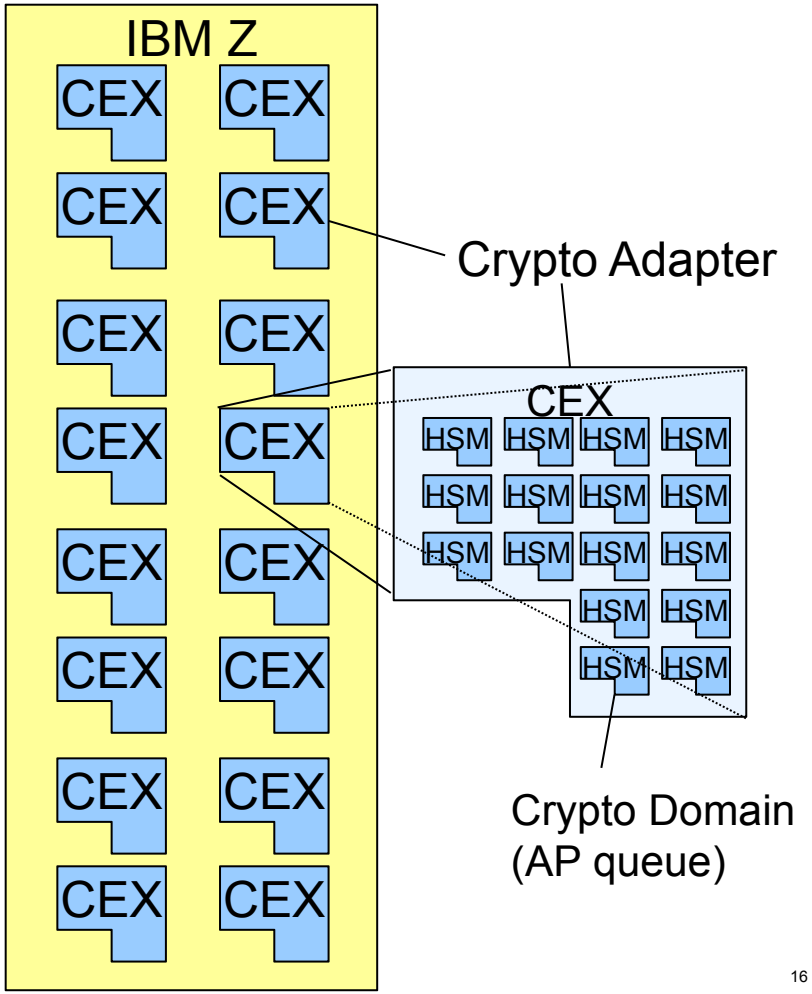
A protected key comprises
- an encrypted clear key
- a master key verification pattern

IBM Z Secure Key Support with Crypto Express

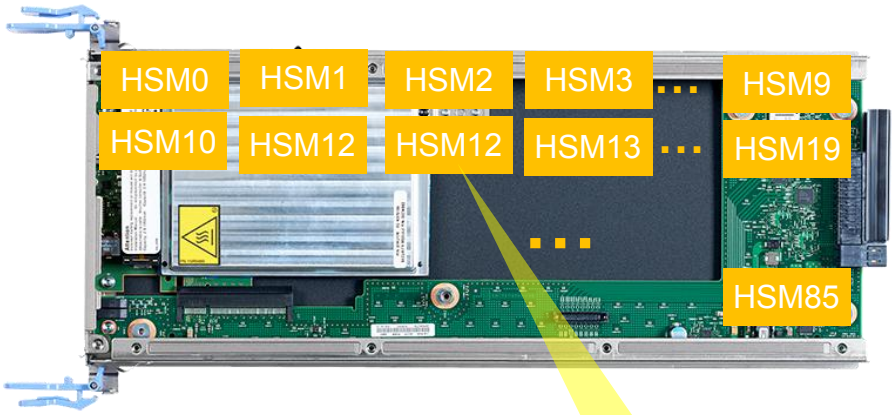
A CryptoExpress adapter
- in CCA or EP11 coprocessor mode
- contains an array of HSMs
- up to 16 per IBM Z CEC

With z13 or z14 each CryptoExpress adapter is partitioned into 85 domains
- is an independent HSM
  - has its own master key
- inherits the mode of its adapter (CCA or EP11)

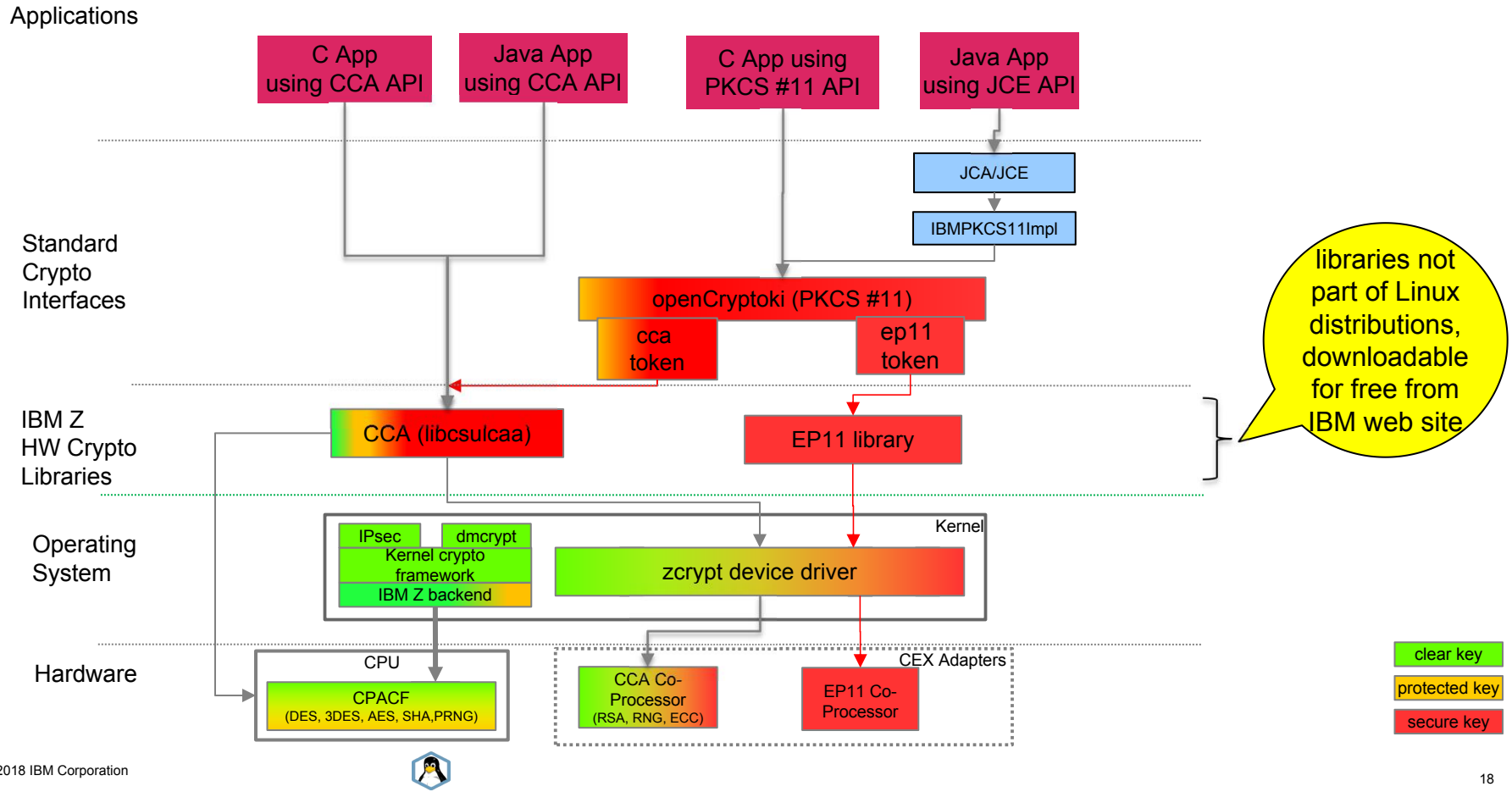
Adapter IDs and domain IDs can be assigned to
- LPARs or
- dedicated to KVM or z/VM guests
- No domain within an CEX adapter can be assigned/dedicated to two LPARs or guests



IBM Z Secure Key Support with Crypto Express



Linux on Z Secure Key Crypto Stack (Applications)



CCA vs EP11

Common Cryptography Architecture (CCA)
IBM proprietary interface

- Supported keys:
  - secure keys
  - protected keys
  - clear keys
- Supported functions
  - basic crypto functions
  - banking functions (PIN handling)
- Key attributes
  - as required for financial services (e.g. PCI)
- Language support
  - C, Java (multiple interfaces)
- Master key management
  - via TKE (recommended for production)
  - per command line tool

Enterprise PKCS #11 (EP11)
Standard PKCS #11 interface

- Supported keys
  - secure keys
- Supported functions
  - PKCS #11 functions and mechanisms
- Key attributes
  - according to PKCS #11
- Language support
  - C, Java (via IBMPKCSImpl)
- Master key management
  - via TKE

Some CCA Concepts

- The adapter(s) addressed is determined via
  - environment variable `CSU_DEFAULT_ADAPTER`
  - or CCA verb
- The domain addressed is the Linux default domain (`/sys/ap/bus/ap_domain`)
- CCA supports a key store
- in CCA verbs
  - keys referred to by key labels are keys from key store

Tools

- ipv.e:
  - check state of CCA adapter
- panel.exe:
  - manage master keys and key store
  - Unix groups
    - restrict master key management operations

PKCS #11 Concepts

- PKCS #11 can address multiple HSMs

- A **PKCS #11 token** represents crypto HW (e.g. an HSM)

- A PKCS #11 token is addressed via **slot**

- A PKCS #11 token is protected by two PINs
  - a **security officer (SO) PIN**
  - a **User PIN**

- the SO can initialize token and set/reset PINs
- the User can change its PIN and perform cryptographic operations

- Before being used a token MUST
  - be initialized (given a label)
  - have the default SO PIN changed
  - have the User PIN set

PKCS #11 object types
(keys are the most important PKCS #11 objects)

- **public** / **private** keys
  - the User PIN must be provided to access private objects

- **session** keys / **token** keys
  - session keys are volatile (do not live longer than process)
  - token keys are persisent
    - can live longer then process
    - can be shared by processes

openCryptoki Concepts

Token type
- describes the kind of crypto implementation / HSM
- defined by library implementing the token
- since version 3.8 multiple instances of the same token type supported

Token definition:
- in `/etc/opencryptoki/opencryptoki.conf`
  - token name
  - library implementing the token
  - token config file (for some token types)

Token specific data (PIN hashes, token objects)
- `/var/lib/opencryptoki/`tokenname
- contains PIN hashes
- token key repository:
  - `TOK_OBJ` sub directory contains token objects

pkcsslotd
- daemon that coordinates access to token resources
- must be started (e.g., per init)

pkcs11 group:
- processes using opencryptoki must belong to this group

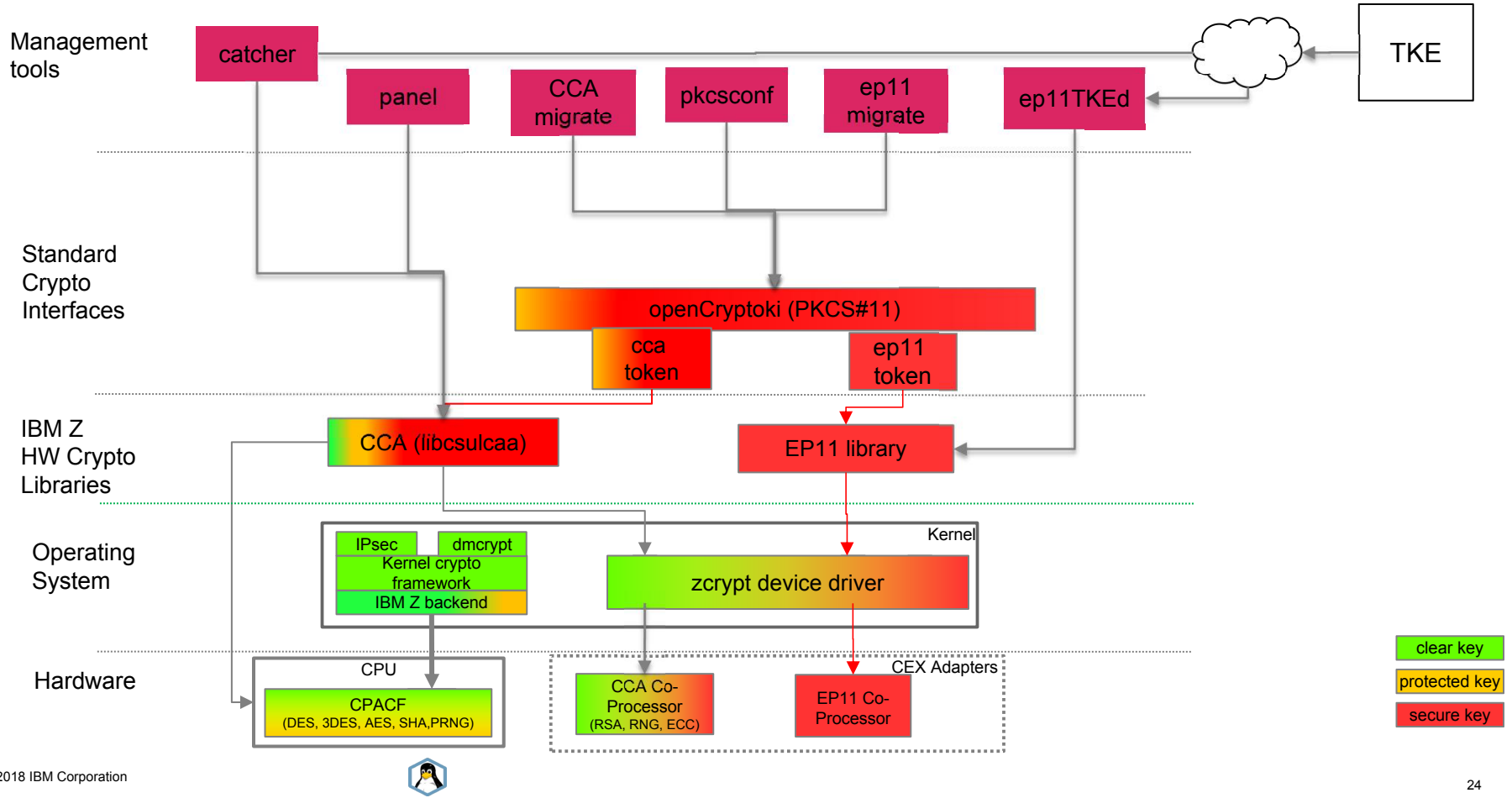
Token configuration

- FIRST: ensure HSM has master key configured

- pkcsconf tool
  - manages slots / tokens
  - initialize token
  - set SO PIN, User PIN
  - change SO PIN, user PIN
  - show slot / token information
  - show mechanisms supported by token

Master Key Management

Linux on Z Secure Key Crypto Stack (Management)



Define HSMs manageable by a Linux system

– On the Support Element
  - assign *control domains* to LPARs

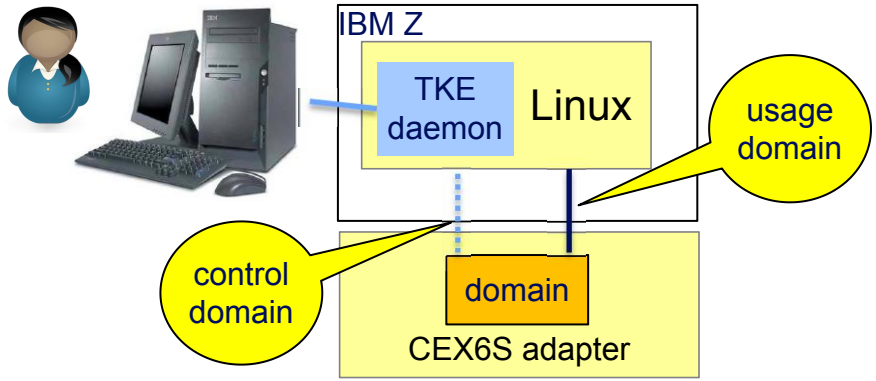
– If adapter A and control domain C is assigned to LPAR L
  - then L can manage the master key of the HSM (adapter domain) with APQN (A,C)

– For all KVM and z/VM guests the list of control domains is equal to the list of usage domains

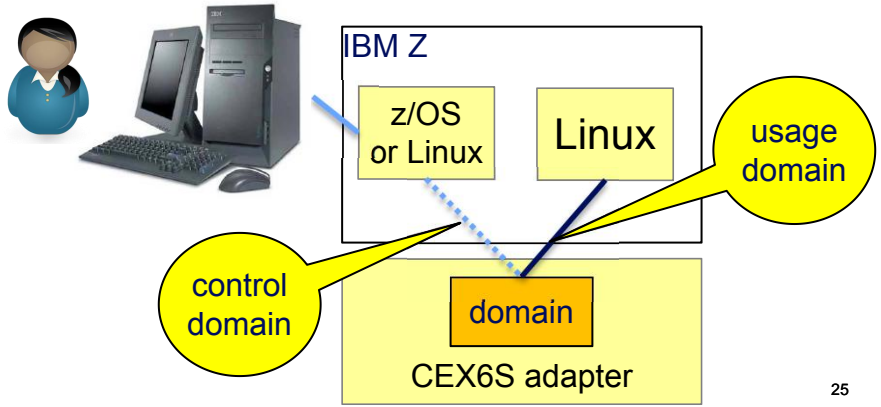
– In Linux on Z
  - `/sys/bus/ap/ap_control_domain_mask` shows control domains assigned to Linux system

How to access HSMs

– directly via CCA catcher.exe or ep11TKEd daemon



– indirectly via system with control domain to HSM



Master Key Configuration

Done on TKE
– per domain, per adapter

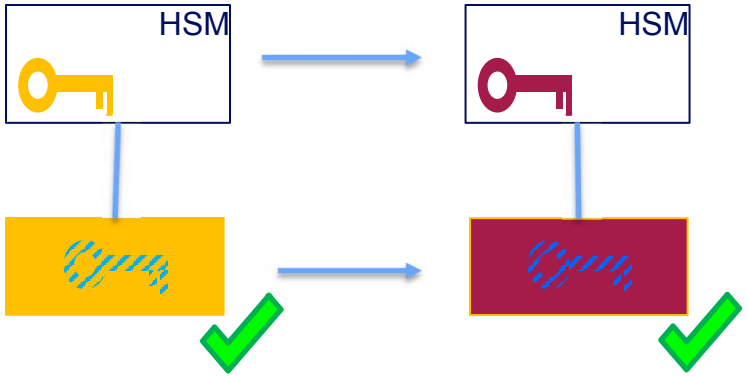
– domain configuration:
  - # of admins / # of key parts / admin signatures

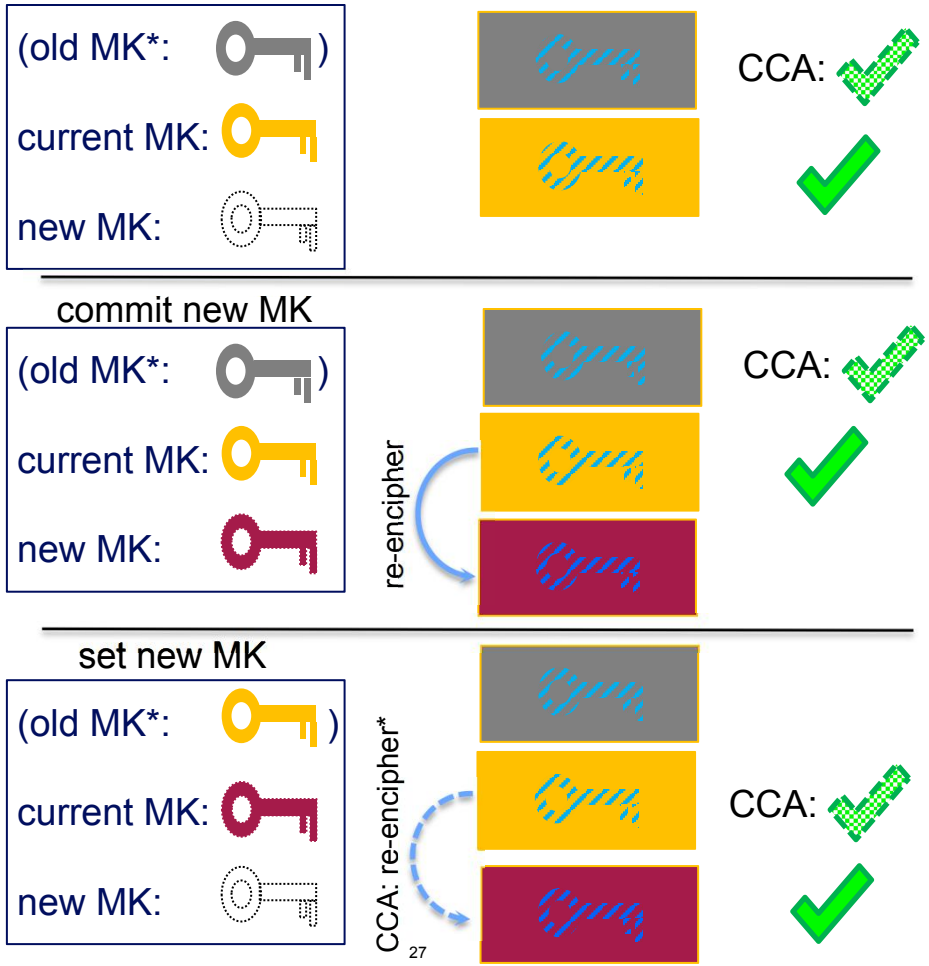
– Setting master keys (MKs):
  1. generate key parts
  2. store MK parts on smart cards (to be kept in a safe)
  3. load MK (from smart cards)
  4. commit MK (loading complete)
  5. set MK (ready to use)

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

Master Key Change

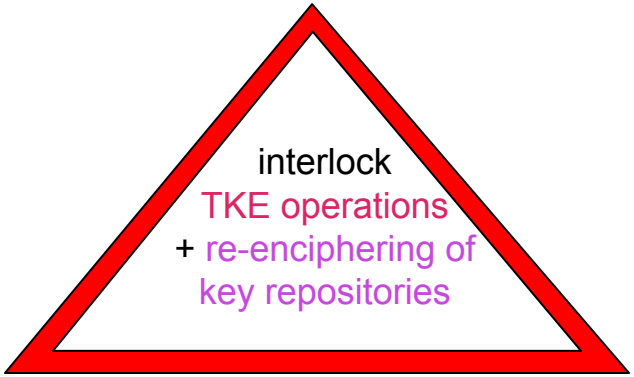
- a very rare operation
  - typically less frequent than once a year
- requires to re-encipher (re-wrap) all secure keys





*) old MK register: only on CCA HSMs

Master Keys Change Protocol



EP11

- token key repository of openCryptoki ep11 token
  - disruptive procedure
  1. stop all processes using openCryptoki EP11 token
  2. commit new MK on HSM
  3. use *the pkcsep11_migrate tool* to re-encipher token keys
  4. activate new MK on HSM
  5. restart processes using openCryptoki EP11 token

CCA

- Keys in CCA key store
  - non-disruptive procedure if all applications refer to keys only via key labels
  - applies to keys in key store
  1. perform MK change on HSM
  2. use *panel.exe tool* to re-encipher keys in CCA key store

- token key repository of openCryptoki cca token
  - disruptive procedure
  1. stop all processes using openCryptoki CCA token
  2. perform MK change on HSM
  3. use the pkcscca tool to re-encipher token keys
  4. restart processes using openCryptoki CCA token

- Note, CCA has 4 different MKs for different key types: (3)DES, AES, RSA, ECC
  - each MKs can be changed independently.

HSM Resilliency

Redundant HSMs

- multiple HSMs configured with the same MK
- TKE supports cloning MKs to configure redundant HSMs

- Redundant HSM assigned to the same Linux system
  - Linux kernel sends crypto request submitted to multiple (redundant) HSMs to any one of those
    - load balancing request among the targeted HSMs
    - failing over to another HSM if the first one fails
  - configuration of common master keys is responsibility of HSM admin

- In an HA / DR cluster
  - primary and back-up system(s) must have access to redundant HSMs

Addressing redundant HSMs

CCA

- set envionment variable CSU_DEFAULT_ADAPTER = DEV-ANY
- all CCA adapters assigned to Linux must be configured with same MKs

EP11

- in EP11 token configuration file define
  - white list of HSMs (APQNs) that have a common MK or
  - APQN_ANY if all EP11 adapters assigned to Linux are configured with same MKs

Master key change for redundant HSMs

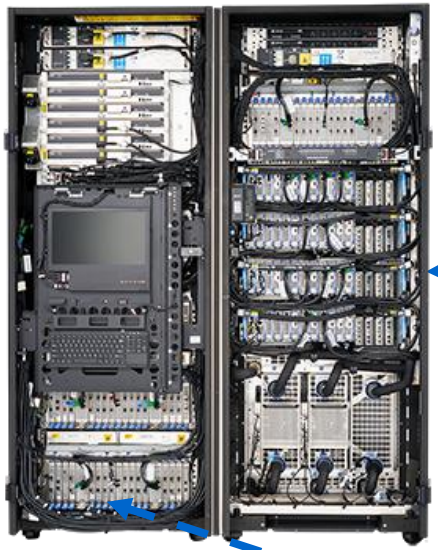
- change MKs on all HSMs
- re-encipher keys / key repositories only once

Agenda

- Background
- Use HSM and CPACF to protect data with IBM Z
- **Usage in Linux**
- Upcoming features
- Application in Blockchain

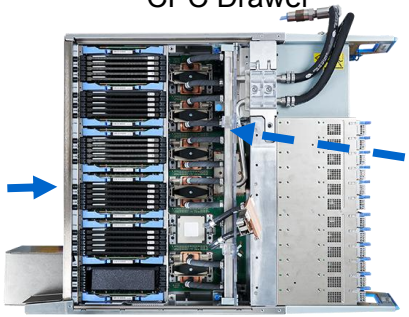
Using Crypto Express Adapters with Linux

Overview – HW Crypto support in IBM Z

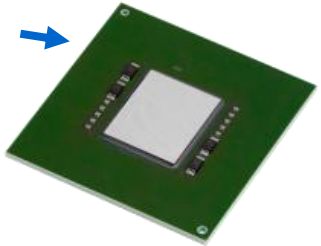


PCIe I/O drawers

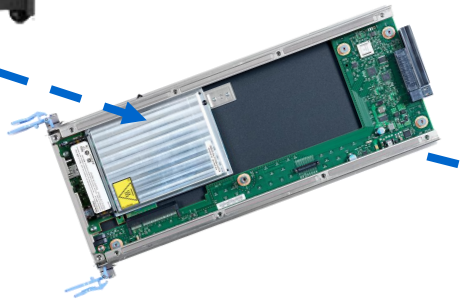
CPC Drawer



PU SCM
Each PU is capable of having the CPACF function
With protected key support



Crypto Express6S, *an HSM in CCA or EP11 mode*



Trusted Key Entry (TKE)

Smart Cards

Configuring HSMs usable by a Linux System

- On Support Element
  - assign adapters to LPARs
  - assign usage domains to LPARs

  - If adapter A and usage domain D is assigned to LPAR L
    - then L has access to the HSM (adapter domain) with APQN (A,D)

  - no two LPARs may access the same APQN

- in Linux on Z or KVM
  - `/sys/bus/ap/ap_usage_domain_mask` shows usage domains assigned to Linux system

- on z/VM
  - configure adapters and domains for a guest with APDED directory statement

HSMs in Linux on Z

Representation of adpaters

– `/sys/bus/ap/card`**xx**

adapter ID

– adapter attributes
  - [r/o] hwtype
    – CCA co-processor: CEXnC,
    – EP11 coprocessor: CEXnP
    – accelerator: CEXnA -- not a HSM
  - [r/o] raw_hwtype (numerical type identifier)
  - [r/w] online
  - [r/o] pendingq_count
  - [r/o] request_count
  - [r/o] requestq_count

– use `lszcrypt` to display adapter attributes
– use `chzcrypt` to set adapter online/offline

Representation of HSMs (AP queues)
as of kernel version 4.9

– `/sys/bus/ap/card`**xx**`/`**xx**`.yyy`

domain ID

– HSM (ap queue) attributes
– [r/w] online
– [r/o] interrupt (enabled or polling)
– [r/o] reset
– [r/o] pendingq_count
– [r/o] request_count
– [r/o] requestq_count

– use `lszcrypt` to display HSM attributes
– use `chzcrypt` to set HSM online/offline

Note, before kernel version 4.9 only one domain was supported: HSM attributes = adapter attributes

CCA

How to install

- download latest CCA package (Debian or RPM) from http://www.ibm.com/security/cryptocards/pciecc2/lonzsoftware.shtml

- install package as desribed in CCA book: https://www.ibm.com/support/knowledgecenter/en/linuxonibm/liaaf/lnz_r_ccacnt.html
- the CCA package contains
  - the CCA library (libcsulcca)
  - a tool to verify installtion of CCA (ivp.e)
  - the TKE daemon (catcher.exe)
  - a tool to manage master keys and a key store (panel.exe)

- start catcher.exe if you want to use your Linux to set master keys from the TKE

How to configure CCA

- CCA 5.2 only uses the default domain displayed in /sys/bus/ap/ap_domain

- use environment variables
  - CSU_DEFAULT_ADAPTER to select adapter DEV-ANY: use any of the CCA adapters (must have the same master key!!!)
  - CSU_HCPUACLR = 1 to let CCA use CPACF for clear key operations
  - CSU_HCPUAPRT = 1 to let CCA use CPACF for protected key operations for some symmetric secret key crypto

- Unix goups to control CCA operations:
  - cca_admin, cca_clrmk, cca_lfmkp (load 1st part), cca_cmpk (load middle & last part), cca_setmk

EP11

How to install

- download latest EP11 package (Debian or RPM) from
- http://www.ibm.com/security/cryptocards/pciecc2/lonzsoftware.shtml

- install package as desribed in EP11 book:
- https://www.ibm.com/support/knowledgecenter/en/linuxonibm/com.ibm.linux.z.lxce/lxce_usingep11.html
- the EP11 package contains
  - the EP11 library (libep11)
  - the TKE daemon (ep11TKEd)

- start ep11TKEd if you want to use your Linux to set master keys from the TKE

- install openCryptoki (shipped with your distribution)

How to configure

- to define a second token instance edit /etc/opencrptoki/opencryptoki.conf

- configure ep11 token using pkcsconf

- Provide EP11 config file
  - list of adressable HSMs
    - APQN_WHITELIST or
    - APQN_ANY

Example
EP11 Token Configuration

- opencryptoki.conf:

```
...
slot 4
{
stdll = libpkcs11_ep11.so
confname = ep11tok.conf
}
...
```

- check for available tokens

```
# pkcsconf -t
...
Token #4 Info:
        Label:
        Manufacturer: IBM Corp.
        Model: IBM EP11Tok
        ...
```

- ep11tok.conf:

```
## EP11 token configuration
#APQN_ANY
APQN_WHITELIST
5 2
6 2
END
```

- set label of ep11 token

```
# pkcsconf -I -c4
Enter the SO PIN: ********
Enter a unique token label: ep11token
```

initial PW: 87654321

- change SO pin of ep11 token:

```
# pkcsconf -P -c4
Enter the SO PIN: ********
Enter the new SO PIN: ********
Re-enter the new SO PIN: ********
```

- set user pin of ep11 token:

```
# pkcsconf -u -c4
Enter the SO PIN: ********
Enter the new user PIN: ********
Re-enter the new user PIN: ********
```

- change user pin of ep11 token:

```
# pkcsconf -p -c4
Enter the user PIN: ********
Enter the new user PIN: ********
Re-enter the new user PIN: ********
```

- verify configuration of ep11 token

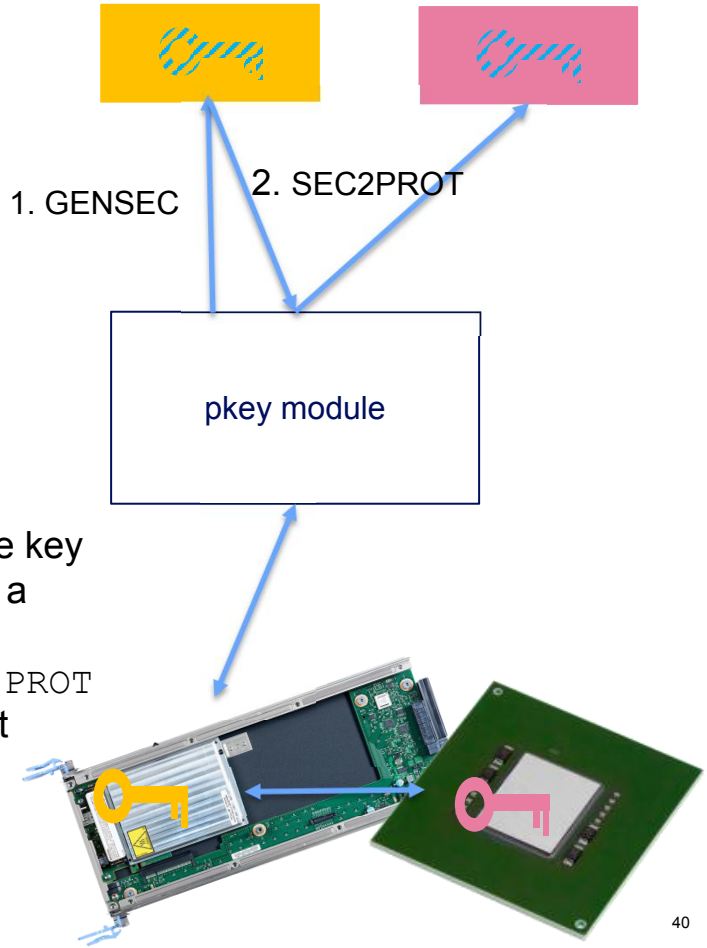
```
# pkcsconf -t -c4
Token #4 Info:
    Label: ep11token
    ...
    Flags: 0x880445 (RNG|LOGIN_REQUIRED|
    USER_PIN_INITIALIZED|CLOCK_ON_TOKEN
    |TOKEN_INITIALIZED)
    ...
```


Using Protected Keys in Linux

Generating Protected Keys

with the pkey kernel module

- prerequisite
  - access to CCA coprocessor (mit MK configured)
- activate with
  - `# modprobe pkey`
- implements misc device: `/dev/pkey`
- provides functions (IOCTLs) to
  - `PKEY_GENSEC`: generate a random CCA secure key
  - `PKEY_CLR2SEC`: generate a CCA secure key from a clear key
  - `PKEY_SEC2PROT`: generate a protected key from a CCA secure key
  - `PKEY_FINDCARD`: find an adapter and domain associated with a given secure key
  - `PKEY_SECK2PROTK`: first `PKEY_FINDCARD` then `PKEY_SEC2PROT`
  - `PKEY_GENSEC`, `PKEY_CLR2SEC`, `PKEY_SEC2PROT` have target adapter&domain as well as key type as arguments

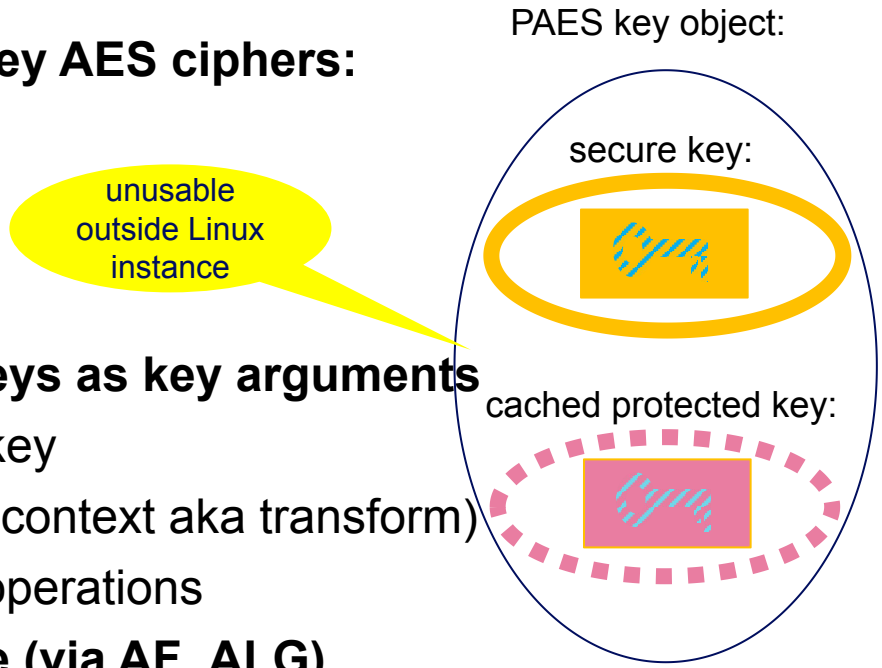


Secure Key Handling: the zkey Tool

- new tool in s390tools 1.39
- requires pkey module
- generate, validate, re-encipher secure AES keys to be transformed into protected keys
  - generate
    - generates file with AES secure key (AESDATA)
    - random key or from clear key
    - single key (for CBC) or two keys (for XTS)
    - size of secure keys: 64 bytes (single key), 128 bytes (XTS key) regardless of AES key size
  - validate
    - checks if input file contains valid AES secure key
    - if yes displays key attributes
  - re-encipher
    - support master key change on Crypto Express adapter
    - transforms a valid secure key wrapped by a current (or old) HSM master key into a secure key wrapped by a new (or current) master key
    - requires installation of CCA package

The PAES in-kernel Cipher

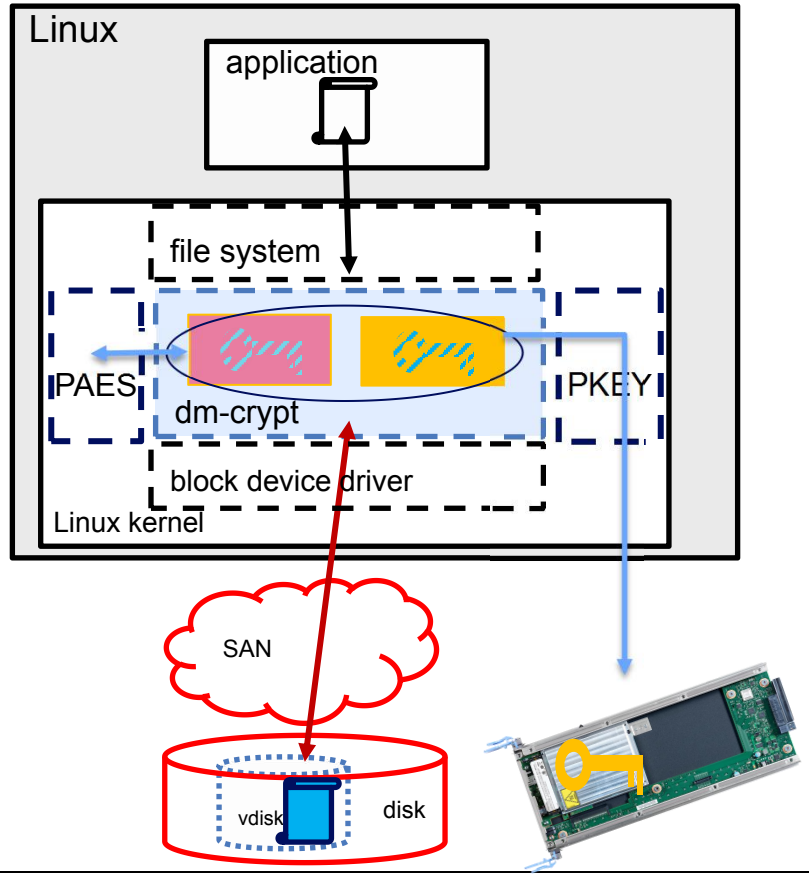
- **paes_s390 module upstream since kernel 4.11**
- **paes module implements Protected key AES ciphers:**
  - ecb(paes), cbc(paes)
  - xts(paes), ctr(paes)
- **requires the pkey module as prereq**
- **paes ciphers take CCA AES secure keys as key arguments**
  - transforms secure key into protected key
  - caches protected key (into encryption context aka transform)
  - uses protected key for cryptographic operations
- **can be used by kernel and user space (via AF_ALG)**



Linux Volume Encryption with Protected Keys

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

- **E2E data encryption**
  - The complete I/O path outside the kernel is encrypted:
    - HV, 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 LV)
  - encrypts all data written to disk
  - decrypts all data read from disk
- **How it works:**
  - uses in kernel-crypto
    - can use IBM Z CPACF Crypto:
      - AES-CBC (not recommended)
      - XTS-AES
      - XTS-PAES
  - 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

- **generate a file with a secure key**

once

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

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

- **open block device as device mapper volume**

with every boot

```
# 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

- **use new device mapper volume**

BAU

  - (only once) create file system:

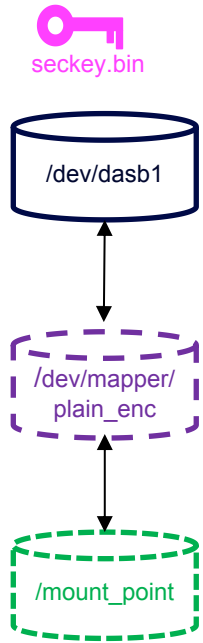
```
# mkfs.ext4 /dev/mapper/plain_enc
```

  - mount:

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

  - access:

```
# echo "hello world" > /mount_point/myfile
```



/etc/crypttab

open dm-crypt volumes automatically at boot time

- `/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> <key file> [options]
    - sample entry for PAES encrypted volume:

```
plain_enc /dev/dasdd1 /root/seckey.bin plain,cipher=paes-xts-plain64,hash=plain, size=1024
```

Note, the syntax of `/etc/crypttab` entry may vary by distribution.

Storing Keys

- **Storing a clear key**
  - requires to protect the key by access control or better by encrypting the key (like LUKS keys)
    - the latter requires
      - interactively query a password or
      - to securely store the wrapping keys …

- **Storing a secure or protected keys**
  - on an unencrypted disk does not reveal secrets outside your system.
  - a secure keys need not be password protected when stored
  - they can be stored on the boot disk (initrd) and allow for automatic opening of encrypted volumes
    - no need to interactively query a password

Keys chosen

Which Keys to keep as Secure or Protected Keys?

- All keys?
  - yes, if you can afford it
  - but secure keys crypto
    - is harder to manage than clear key crypto
    - may be slow, especially for bulk data

- There may be laws or regualtions that force you to use an HSM and secure keys
  - then you have little choice

- Protect your most valuable keys:

  - keys that have a long life time
    - private signing keys
    - keys to encrypt data at rest

  - keys that protect the most valuable asset
    - personal data
    - PINS
    - other keys

- For performance reasons
  - use protected keys to encrypt large amounts of data

Summary

Your cryptographic keys are your most valuable pieces of data

– IBM Z secure and protected key crypto protects your keys:

  • prevent stolen key to be used for offline attacks

IBM Z Crypto Express provides for two secure key disciplines

– CCA: implements an IBM Proprietary Standard developed together with customers from finance industry

– EP11: implements the popular PKCS #11 standard

IBM Z CPACF protected keys, provide almost HSM like security

– for bulk data in a very efficient manner

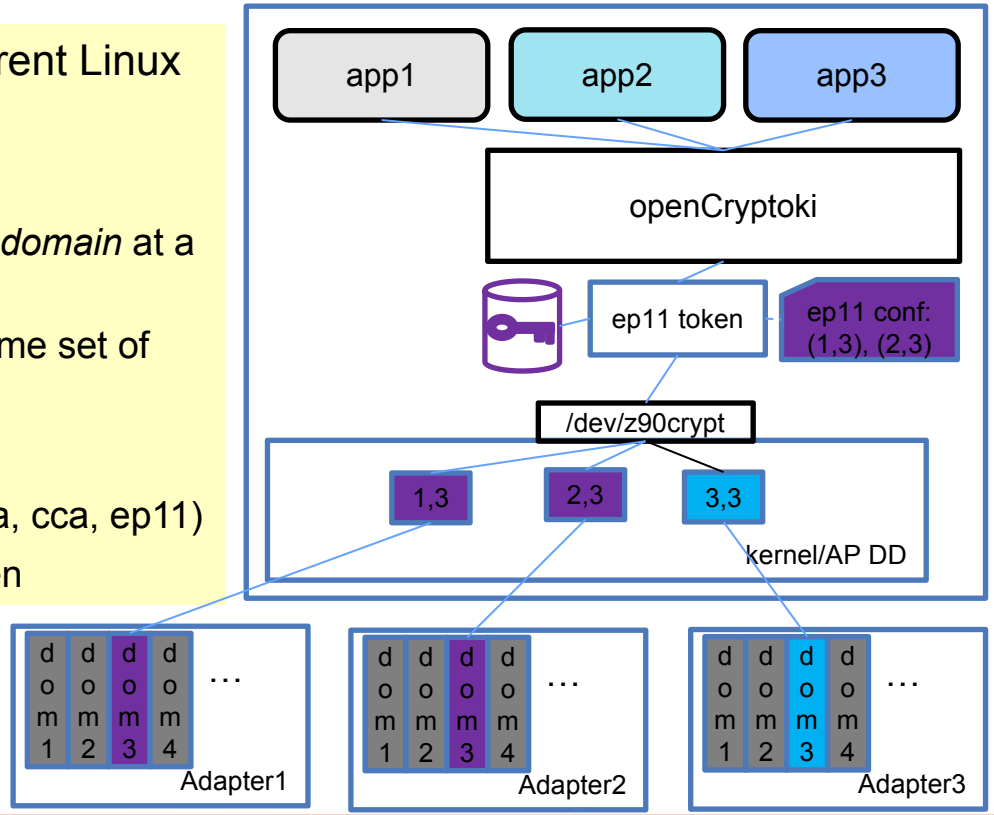
– can protect Linux volumes using dm-crypt

Agenda

- Background
- Use HSM and CPACF to protect data with IBM Z
- Usage in Linux
- **Upcoming features**
- Application in Blockchain

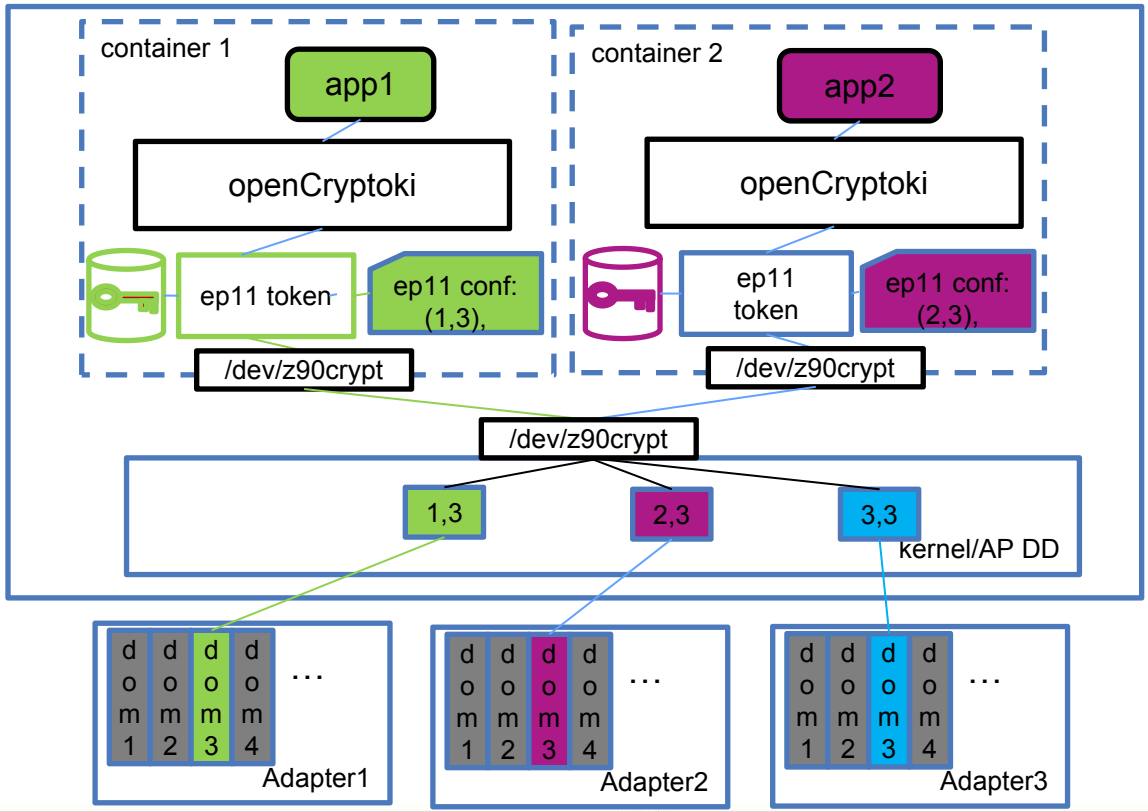
Upcoming features – “Today”

- “Today” = before kernel 4.10, in current Linux distribution releases
- Linux instance supports
  - multiple crypto *adapters but only one domain* at a time
  - all applications have access to the same set of adapters (/dev/z90crypt)
- openCryptoki supports
  - one token instance per token type (ica, cca, ep11)
  - one global configuration for each token



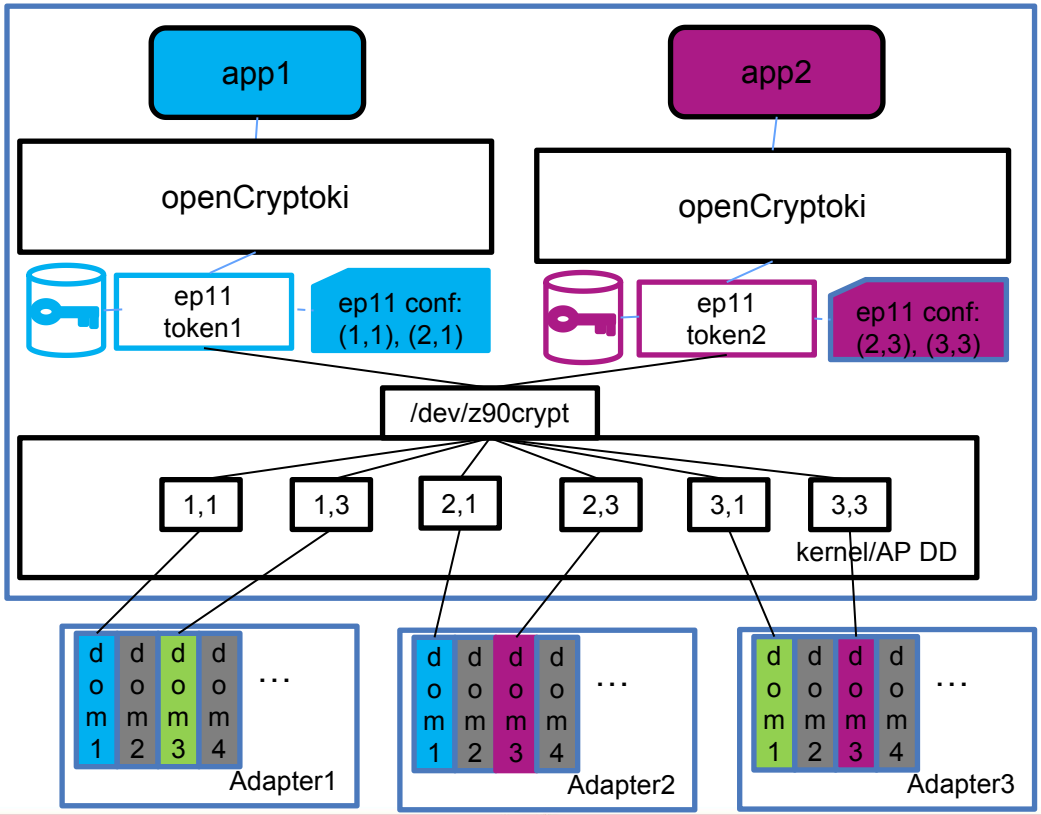
Upcoming features – Today with Docker

- attach crypto device to container with option
- `--device /dev/z90crypt:/dev/z90crypt`
- allows to define a different openCryptoki configuration for each container
- *but* each container can access all adapters and all domains



Upcoming features – “Tomorrow”

- “Tomorrow” = **Multi-Tenant Crypto Support**, in upcoming distribution releases
- Kernel 4.10 supports accessing multiple domains
- openCryptoki 3.8 supports multiple token instances per token type
- allows to specify that different applications access different adapters and domains
- e.g multiple ep11 tokens each configurable to use a different set adapters and domains

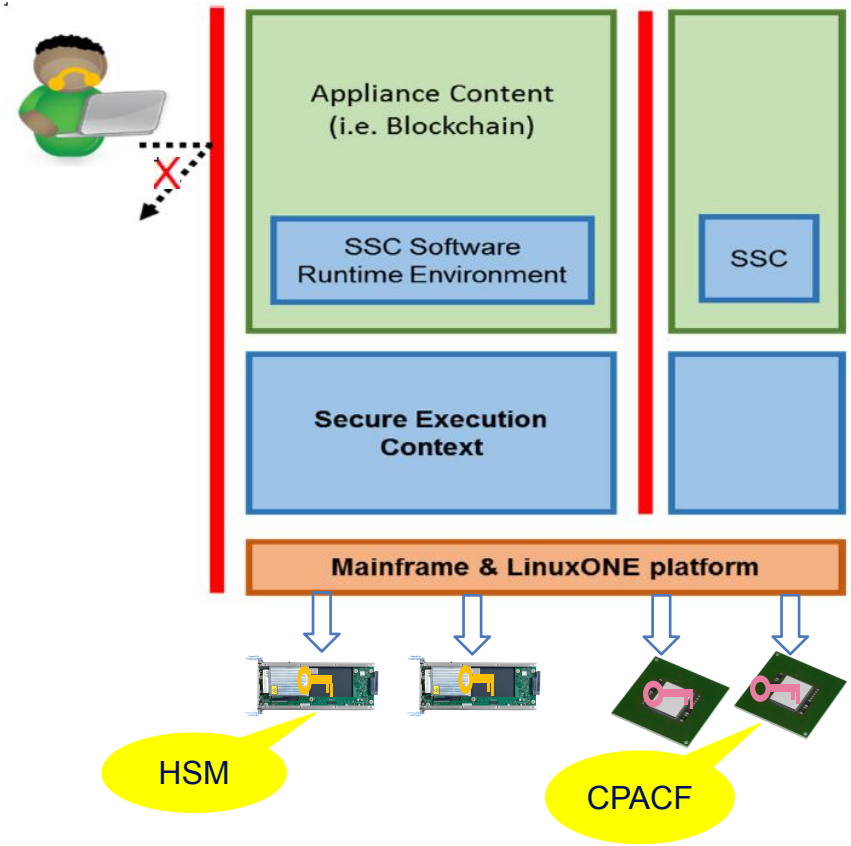


Agenda

- Background
- Use HSM and CPACF to protect data with IBM Z
- Usage in Linux
- Upcoming features
- **Application in Blockchain**

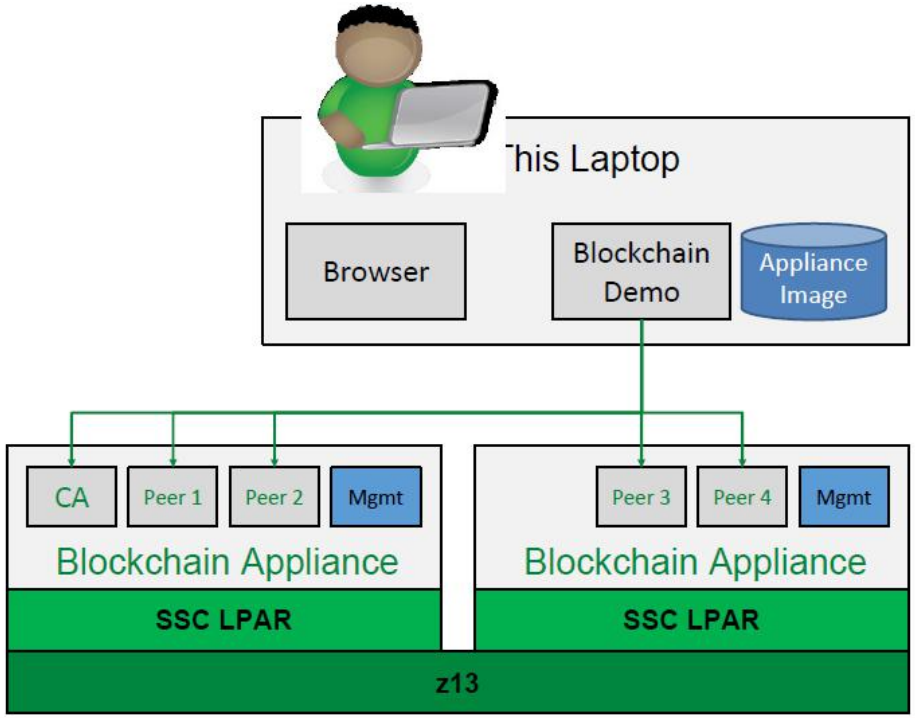
Blockchain as a Service on Z

- Tampering resistant with HSM
- Entities (Peers) Isolation
- Pervasive Encryption on Z14
  - Encryption data in-flight and at rest
  - Encryption on the whole volume with CPACF & HSM
  - High performance, transparent
- No SSH, only auth RESTful API is allowed



Blockchain as a Service on Z

- Deployment in 15 minutes



1. Create LPARs
2. Install zBlockchain Appliances
   - Image stored on Laptop
   - Secure Service Container Installer part of z13
3. Deploy zBlockchain Fabric
   - 1x certificate authority (CA)
   - 4x peers
   - Use management API
4. Connect Blockchain Demo (client) to deployed Fabric