8th European GSE/IBM TU for z/VSE, z/VM and Linux on System z
October 20th - 22nd, 2014, Dresden

VM02: Secure Key Crypto mit Linux for System z - Erfahrungen -

Dr. Reinhard Bündgen
RAS & Crypto Architect
IBM Germany R&D
buendgen@de.ibm.com

Dr. Manfred Gnirss
Senior IT Specialist
IBM Client Center, IBM Germany R&D
gnirss@de.ibm.com

Hilton Hotel
Dresden

Trademarks

Notice Regarding Specialty Engines (e.g., zIIPs, zAAPs and IFLs):

Any information contained in this document regarding Specialty Engines ("SEs") and SE eligible workloads 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). 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.

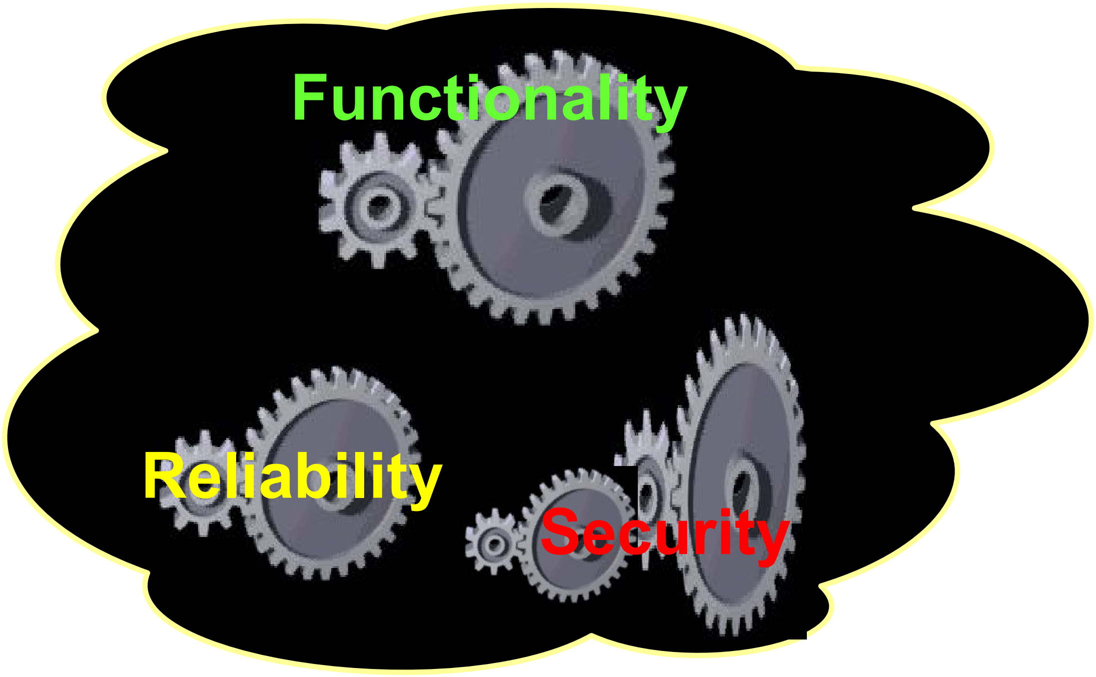
IBM offers SEs at a lower price than General Processors/Central Processors because customers are authorized to use SEs only to process certain types and/or amounts of workloads as specified by IBM in the AUT.

Abstract

Standards für IT Sicherheit gewinnen immer mehr Bedeutung. Die gängigen Standards, wie z.B. Payment Card Industry Data Security Standard (PCI DSS), verlangen einen äußerst sicheren Umgang mit sensitiven Daten. Eine Möglichkeiten Daten zu schützen ist die konsequente Anwendung von kryptographischen Verfahren.

Zusätzlich zu einem Überblick über die aktuellen Möglichkeiten von Kryptographie mit Hardware Unterstützung auf Linux für System z, berichten wir über Erfahrungen, die wir bei der Implementierung von Secure Key Verfahren gemacht haben.

Security vs functionality vs reliability

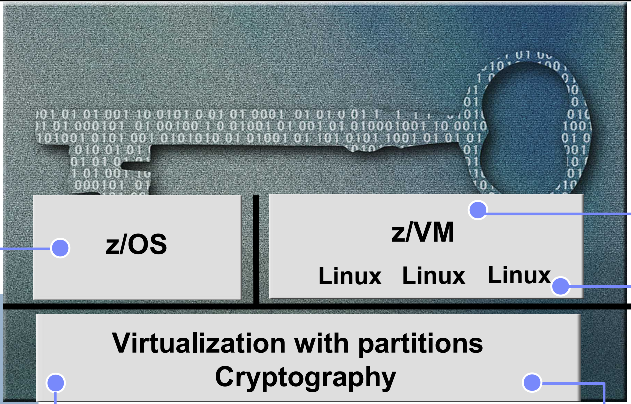


Security Standards

- Increasing importance of regulations and compliance of security standards.
- Some standards:
  - Comon Criteria with Operating System Protection Profile (OSPP)
  - Payment Card Industry Data Security Standard (PCI-DSS)
  - HIPAA
  - BSI Bundesdatenschutzgesetz – Grundschutzkatalog - ZK
  - SOX
  - BASEL II
  - Solvency
  - . . .
- Idea: Even if PCI DSS is not mandatoy for all IT environments it is a good orientation to think about security . . .

System z evaluations and certifications

The Common Criteria program establishes an organizational and technical framework to evaluate the trustworthiness of IT Products and protection profiles



z/OS

- **Common Criteria EAL4+**
  - with CAPP and LSPP
  - z/OS 1.7 → 1.10 + RACF
  - z/OS 1.11 + RACF (OSPP)
  - z/OS 1.12 + RACF (OSPP)
  - z/OS 1.13 + RACF (OSPP)
- **Common Criteria EAL5 +**
  - **z/OS RACF 1.12 (OSPP)**
- **z/OS 1.10 IPv6 Certification by JITC**
- **IdenTrust™ certification for z/OS PKI Services**
- **FIPS 140-2**
  - System SSL z/OS 1.10 →1.12
  - z/OS ICSF PKCS#11 Services
    - z/OS 1.11

System z196 , z114, zEC12 and zBC12

Common Criteria EAL5+ with specific target of Evaluation -- LPAR

Crypto Expr.3 & Crypto Expr.4s Coprocessors
- FIPS 140-2 level 4 Hardware Evaluation
- Approved by German ZKA

CP Assist
- FIPS 197 (AES)
- FIPS 46-3 (TDES)
- FIPS 180-3 (Secure Hash)

z/VM

- System Integrity Statement

Common Criteria

z/VM 5.3 is EAL 4+ for CAPP and LSPP

z/VM 6.1 is EAL 4+ for OSPP

z/VM 6.3 will be EAL 4+ for OSPP (under evaluation)

z/VM 6.3 System SSL Crypto Module validated for FIPS140-2

Linux on System z

- **Common Criteria**
  - SUSE SLES10 certified at EAL4+ with CAPP
  - Red Hat EL5 EAL4+ with CAPP and LSPP
  - SUSE SLES 11 EAL4+ with OSPP
  - RedHat EL6 EAL 4+ with OSPP
- **OpenSSL - FIPS 140-2 Level 1 Validated**
- **CP Assist - SHA-1 validated for FIPS 180-1 - DES & TDES validated for FIPS 46-3**

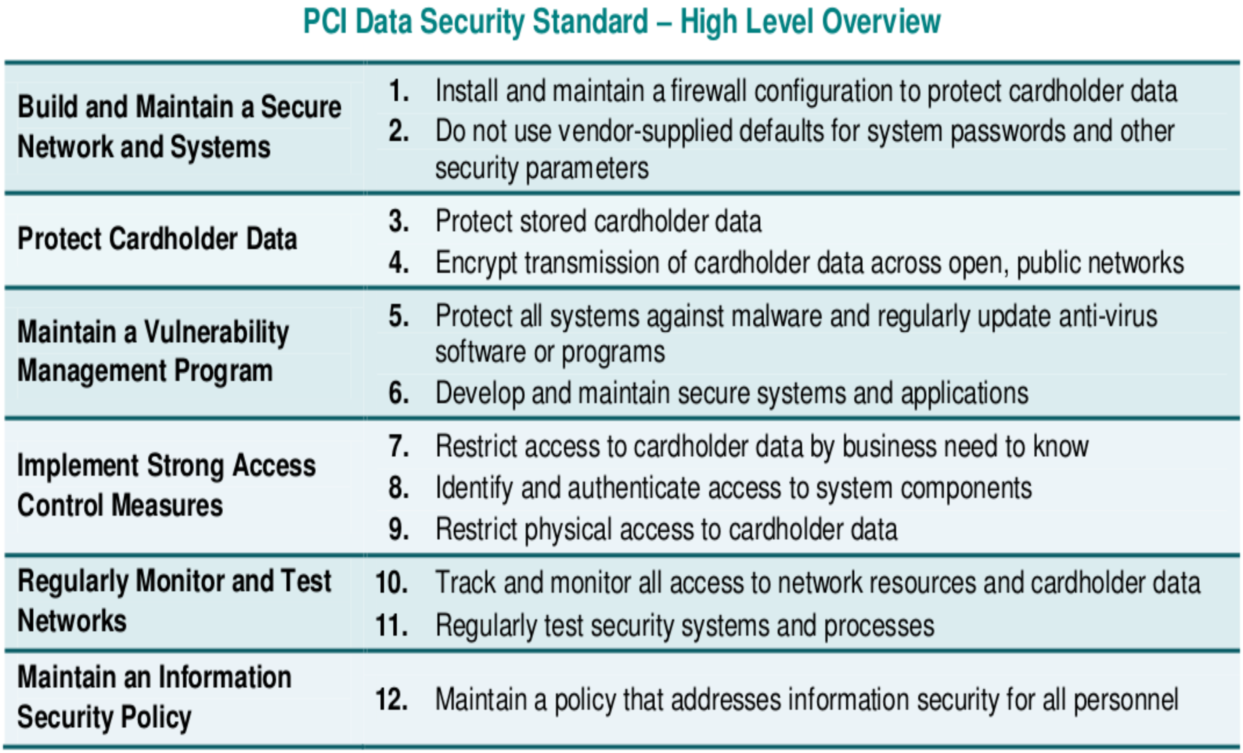
Notes:
- **Common Criteria Certification with Protection Profiles CAPP and LSPP or OSPP requires auditing capabilities**
- **z/OS, z/VM: via RACF**
- **Linux: via Linux Audit Framework**

PCI DSS Overview

- The Payment Card Industry Data Security Standard (PCI DSS) was developed to encourage and enhance cardholder data security and facilitate the broad adoption of consistent data security measures globally. PCI DSS provides a baseline of technical and operational requirements designed to protect cardholder data. PCI DSS applies to all entities involved in payment card processing—including merchants, processors, acquirers, issuers, and service providers, as well as all other entities that store, process or transmit cardholder data (CHD) and/or sensitive authentication data (SAD)
- VISA, MasterCard, American Express, …
- PCI DSS comprises a minimum set of requirements for protecting cardholder data, and may be enhanced by additional controls and practices to further mitigate risks, as well as local, regional and sector laws and regulations
- Use of a Payment Application Data Security Standard (PA-DSS) compliant application by itself does not make an entity PCI DSS compliant, since that application must be implemented into a PCI DSS compliant environment

- Existing Assistance: Requirements – with Test Procedures – with Guidance

PCI DSS requirements



PCI Data Security Standard – High Level Overview

Build and Maintain a Secure Network and Systems	**1.** Install and maintain a firewall configuration to protect cardholder data **2.** Do not use vendor-supplied defaults for system passwords and other security parameters
Protect Cardholder Data	**3.** Protect stored cardholder data **4.** Encrypt transmission of cardholder data across open, public networks
Maintain a Vulnerability Management Program	**5.** Protect all systems against malware and regularly update anti-virus software or programs **6.** Develop and maintain secure systems and applications
Implement Strong Access Control Measures	**7.** Restrict access to cardholder data by business need to know **8.** Identify and authenticate access to system components **9.** Restrict physical access to cardholder data
Regularly Monitor and Test Networks	**10.** Track and monitor all access to network resources and cardholder data **11.** Regularly test security systems and processes
Maintain an Information Security Policy	**12.** Maintain a policy that addresses information security for all personnel

Protect Cardholder Data

R3: Protect stored cardholder data

- Protection methods such as encryption, truncation, masking, and hashing are critical components of cardholder data protection.
- Crypto: If an intruder circumvents other security controls and gains access to encrypted data, without the proper cryptographic keys, the data is unreadable and unusable to that person.
- Other effective methods of protecting stored data should also be considered as potential risk mitigation opportunities. For example, methods for minimizing risk include not storing cardholder data unless absolutely necessary,

Protect Cardholder Data

R3: Protect stored cardholder data . . .

- PCI DSS requirements
  - Keep cardholder data storage to a minimum by implementing data retention and disposal policies, procedures and processes . . .
  - Do not store sensitive authentication data after authorization (even if encrypted)
  - Mask PAN (Primary Account Number) when displayed (the first six and last four digits are the maximum number of digits to be displayed) . . .
  - Render PAN unreadable anywhere it is stored (including on portable digital media, backup media, and in logs) . . .
  - Document and implement procedures to protect keys used to secure stored cardholder data against disclosure and misuse
  - Fully document and implement all key-management processes and procedures for cryptographic keys used for encryption of cardholder data

Protect Cardholder Data

R3: Protect stored cardholder data . . .

- PCI DSS requirements
  - Document and implement procedures to protect keys used to secure stored cardholder data against disclosure and misuse
    - Restrict access
    - Store secret and private keys used to encrypt/decrypt cardholder data in one (or more) of the following forms at all times:
      - Encrypted with a key-encrypting key that is . . .
      - Within a secure cryptographic device such as a host security module (HSM)
      - As at least two full-length key components or key shares, in accordance with an industry- accepted method
    - Store cryptographic keys in the fewest possible locations.
  - Fully document and implement all key-management processes and procedures for cryptographic keys used for encryption of cardholder data
- Linux on z: Consider Secure Key methods (HSM) for encryption of credit card data (access and management of keys)

Protect Cardholder Data

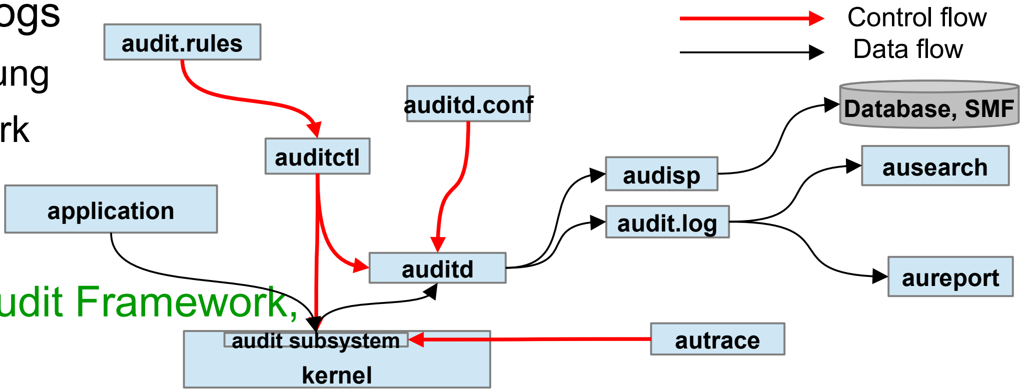
R4: Encrypt transmission of cardholder data across open, public networks

- Sensitive information must be encrypted during transmission over networks that are easily accessed by malicious individuals. Misconfigured wireless networks and vulnerabilities in legacy encryption and authentication protocols continue to be targets of malicious individuals
- Use strong cryptography and security protocols (for example, SSL/TLS, IPSEC, SSH, etc.) to safeguard sensitive cardholder data during transmission over open, public networks
- Security policies are defined and in use (use only trusted keys and certificates, encryption strength, never send unprotected PANs by “end-user messaging” technologies (eMail, chat,...)
- Linux for z: ok

Regularly Monitor and Test Networks

R10: Tack and monitor all access to network resources and cardholder data

- Logging and tracking user activities are critical in preventing, detecting, or minimizing the impact of a data compromise. Logs allow thorough tracking, alerting, and analysis when something does go wrong. Determining the cause of a compromise is very difficult, if not impossible, without system activity logs
  - Anwedung
  - Netzwerk
  - System
- Linux: Audit Framework, firewall



Schutz von Daten:
Kryptographie mit Hardware Unterstützung
auf Linux for System z

Überblick, Möglichkeiten und Erfahrungen

Crypto in general: Why?

- Traditionally: to hide the meaning of transferred or stored data, but also used to establish:
  - Data integrity (No alteration)
  - Authentication (Identity Verification)
  - Data confidentiality (Not disclosure)

- A required facility today for personal or industrial computing
- Hardware Cryptography
  - Offload cryptographic computation workload
    - Some algorithms consumes huge amounts of CPU%
  - Increased performance
    - Speed of computation by specialized coprocessors
  - Security
    - Always more secure than a software implementation
    - Can implement very sophisticated protection of secrets, depending on device

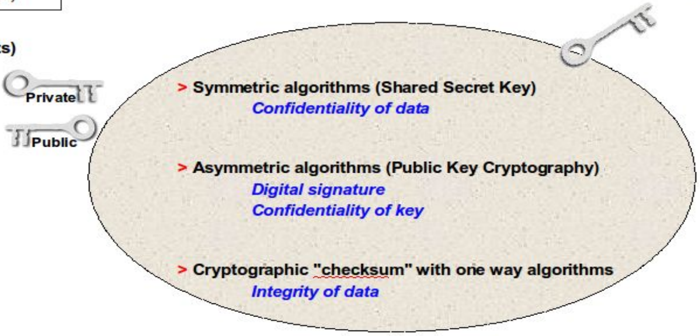
Crypto in general: Algorithms and their usage

DES (56 bits)
T-DES (168 bits)
AES (128 bits)
AES (192, 256 bits)

Data transfer (VPN, SSL/TLS, ..)
Data storage (database, archives, …)

Authentication, key distribution
(VPN, SSL/TLS handshake, ..)

ECC
RSA (512, 1024, 2048, 4096 bits)
DSA (512, 1024 bits)



> **Symmetric algorithms (Shared Secret Key)**
Confidentiality of data

> **Asymmetric algorithms (Public Key Cryptography)**
Digital signature
Confidentiality of key

> **Cryptographic "checksum" with one way algorithms**
Integrity of data

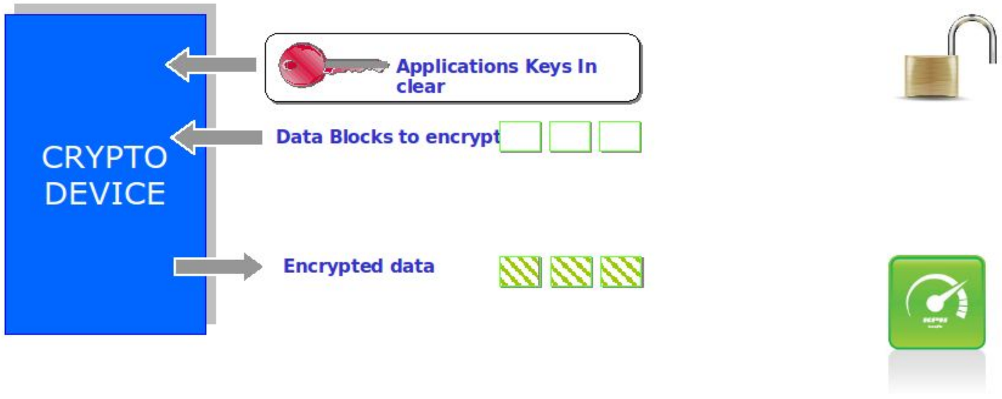
MD5 (128-bit hash)
MAC, MDC
SHA-1(160-bit hash)
SHA-256 (256-bit hash)
SHA-384 , SHA-512

Data transfer (VPN, SSL/TLS, ..)
Data storage (database, archives, …)

Crypto in general: Clear Key implementation

"Clear Key – key may be in the clear, at least briefly, somewhere in the environment"

CPACF, CEX4SA

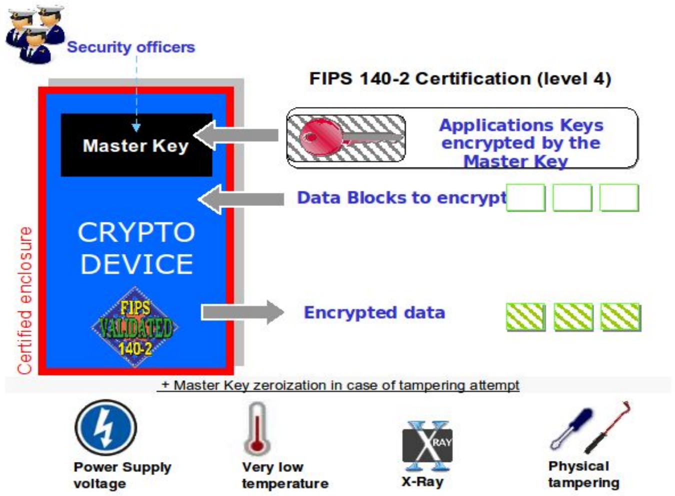


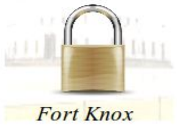
Crypto in general: Secure Key implementation

- Secure Coprocessor - HSM

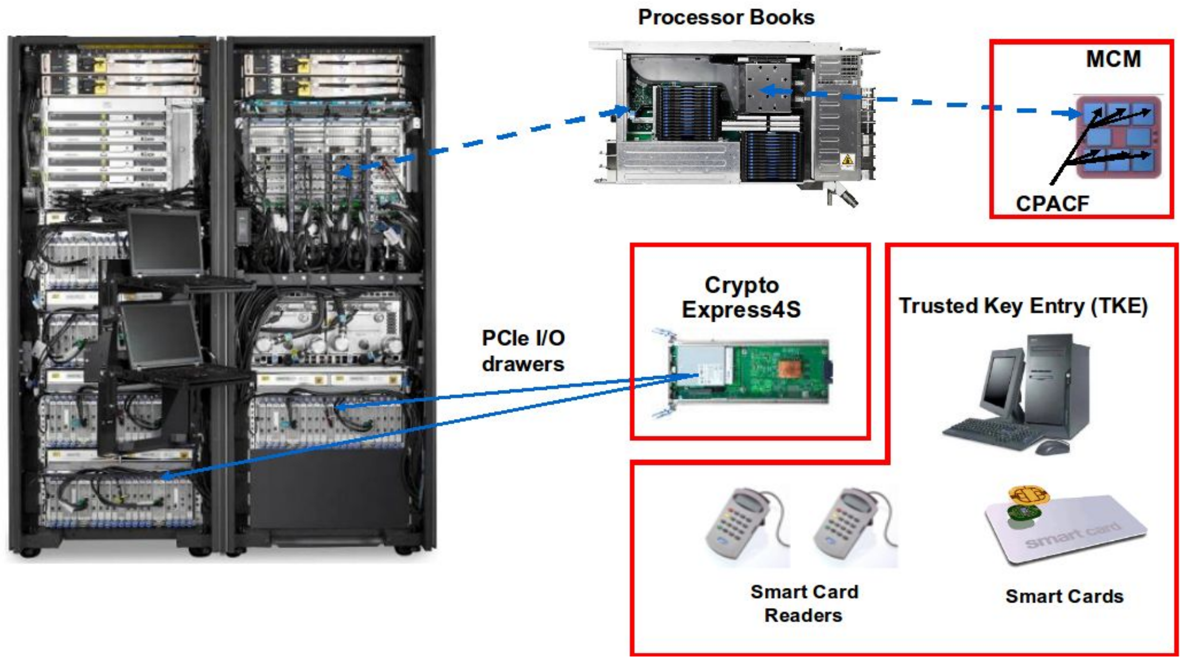
"Secure Key – key value does not exist in the clear outside of the HSM (secure, tamper-resistant boundary of the card)"

CEX4SC, CEX4SEP11





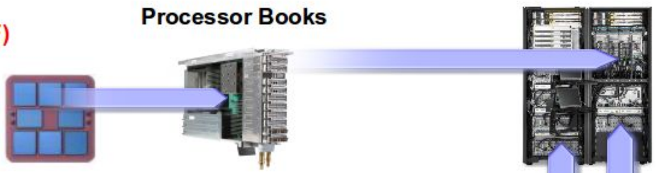
Crypto in general: HW Crypto support in System z



Crypto in general: System z HW Crypto implementation

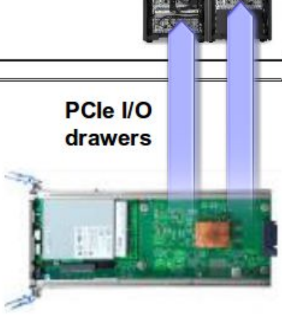
CP Assist for Cryptographic Functions (CPACF)

- A facility integrated in each PU
- Standard orderable feature
- Clear Key & Protected Key only
- Symmetric, hash, …



Crypto Express 4S (CEX4S)

- Priced feature
- 0-16 features in a system
- 1 secure **4765 coprocessors** per feature
- Secure key symmetric (DES, T-DES) and asymmetric (RSA)
- PR/SM sharable
- Manually configurable into an RSA accelerator (CEX3A)
- **FIPS140-2** Certified (As Coprocessor only)



Crypto Express4S

- One PCIe adapter per feature
  - Initial order – two features
- FIPS 140-2 Level 4
- Installed in the PCIe I/O drawer
- Up to 16 features per server
- Prerequisite: CPACF (FC 3863)

- Linux Unterstützung für alle 3 Modi (ICA, CCA, EP11) ist “vorhanden”
- CCA: Klassischer IBM Standard
- PKCS11: Industrie Standard (distr.)
- Heute: Alle Linux Secure Key Nutzer: CCA
- Heute: Falls Kunde Vorgabe hat “private key in HSM” → CCA

- **Three configuration options for the PCIe adapter**
  - Only one configuration option can be chosen at any given time
  - Switching between configuration modes will erase all card secrets
    - Exception: Switching from CCA to accelerator or vice versa
- **Accelerator**
  - For SSL acceleration
  - Clear key RSA operations
- **Enhanced: Secure IBM CCA coprocessor (default)**
  - Optional: TKE workstation (FC 0841) for security-rich, flexible key entry or remote key management
- **New: IBM Enterprise PKCS #11 (EP11) coprocessor**
  - Designed for extended evaluations to meet public sector requirements
    - Both FIPS and Common Criteria certifications
  - Required: TKE workstation (FC 0841) for management of the Crypto Express4S when defined as an EP11 coprocessor

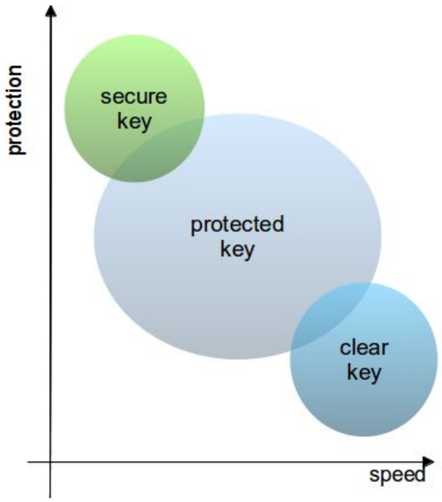
3 levels of protection – 3 levels of speed

- **Clear Key** – key is in the clear, at least briefly, somewhere in the environment
  - Example use: SSL transaction security

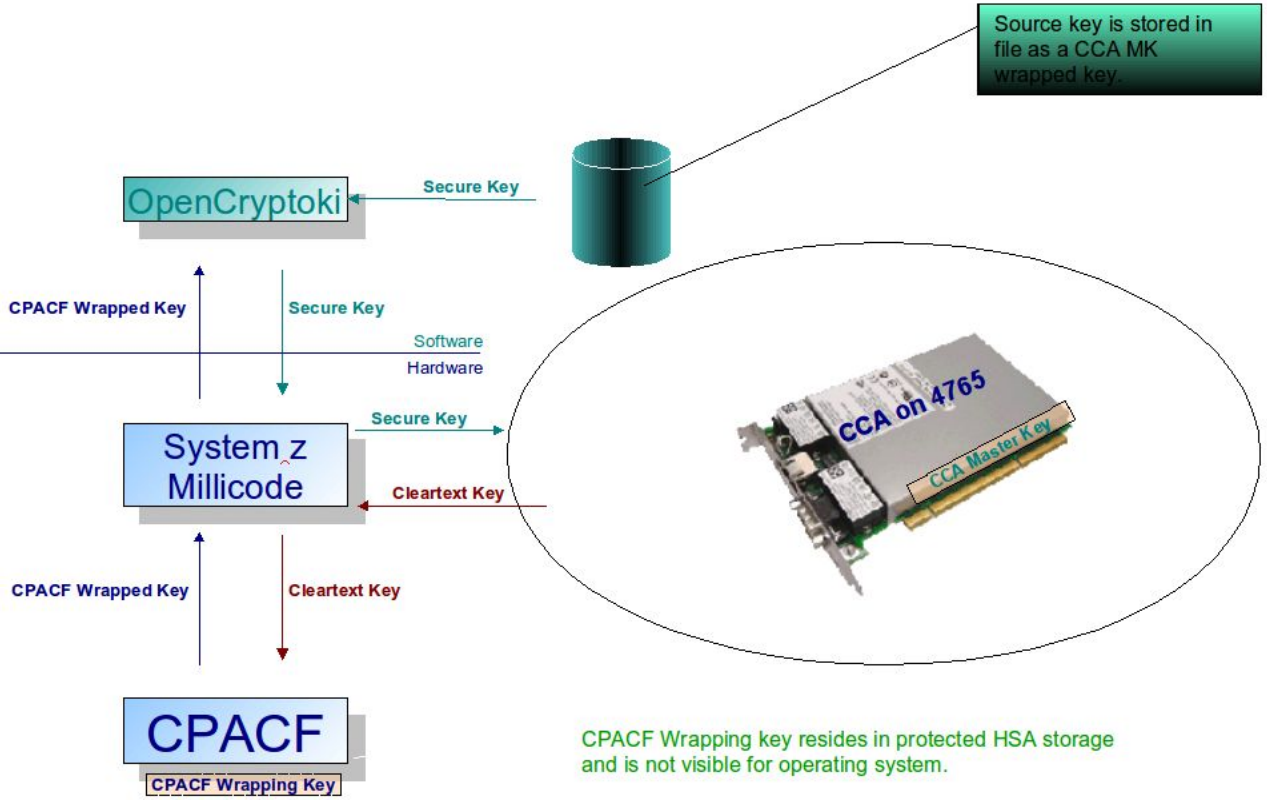
- **Protected Key** – key value does not exist outside of physical hardware, although the hardware may not be tamper-resistant
- Unique to System z
  - Example use: protection of data at rest

  - Enable: CEX4S is needed and Linux environment varialbe csu_hcpuaprt has to be set

- **Secure Key** – key value does not exist in the clear outside of the HSM (secure, tamper-resistant boundary of the card)
  - Example use: PIN handling and verification



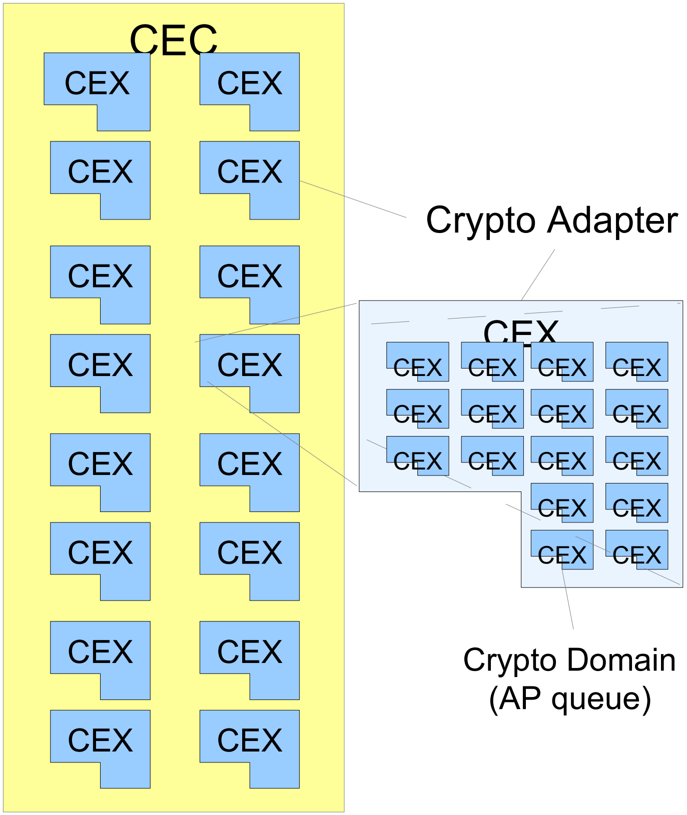
Secure Key CPACF - Key Wrapping



CPACF Wrapping key resides in protected HSA storage and is not visible for operating system.

On Features, Adapters, APs, Domains, Queues. . . .

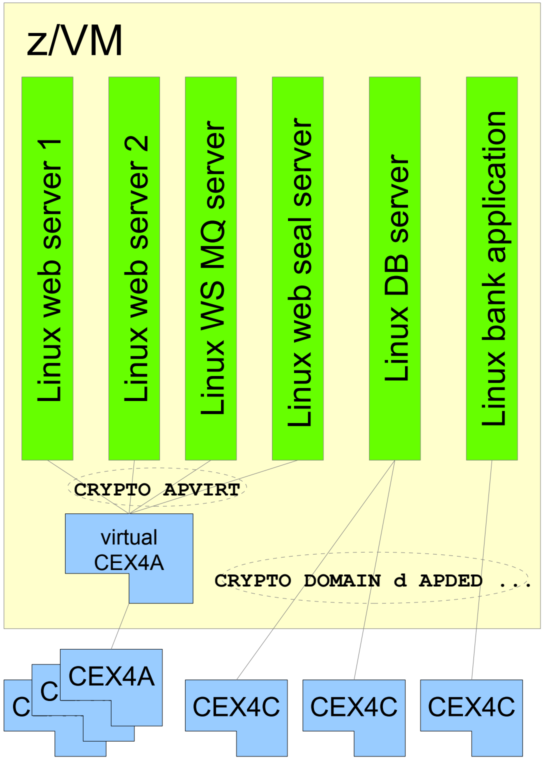
- CEX3 feature has 2 adapters (aka APs). Up to 8 CEX3 features per CEC
- CEX4 feature has 1 adapter (aka AP). Up to 16 CEX4 features per CEC
- Each adapter has an AP Id
- Each adapter has a mode
  - coprocessor or
  - accelerator
- Each adapter can be divided in upto 16 domains (HW virtualization)
- each domain in an AP is represented in SW by an AP queue
- Configuration constraints
  - each LPAR may be granted access to
  - a list (a_1, a_2, ..., a_k) of APs and
  - a list (d_1,d_2, … d_j) of domains
  - resulting in access to AP queues (a_1d_1, ..., a_1d_j, a_2d_1, ..., a_kd_j)
- The Linux on z device driver
  - only uses one domain/AP queue per AP



z/VM Crypto Guest Support

- A guest may have
  - <u>either</u> dedicated adapters
    - `CRYPTO DOMAIN d APDED a1 a2 ...`
  - <u>or</u> shared adapters
    - `CRYPTO APVIRT`
- Shared adapters
  - are of a single type
    - uses only highest priority type
    - priority:
  - CEX4A > CEX3A > CEX2A >CEX4C> CEX3C > CEX2C
  - clear key operations

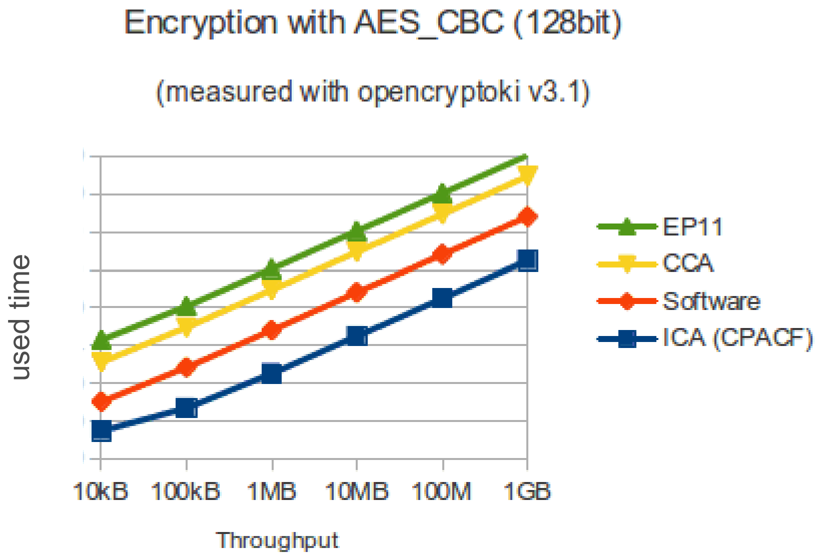
- Checking Crypto Configuration
  - show status of crypto facilities
    - `Q CRYPTO [ DOMAINs [Users] ]`
  - show status of crypto facilities of guest
    - `Q V CRYPTO`



Sorgfältige Planung

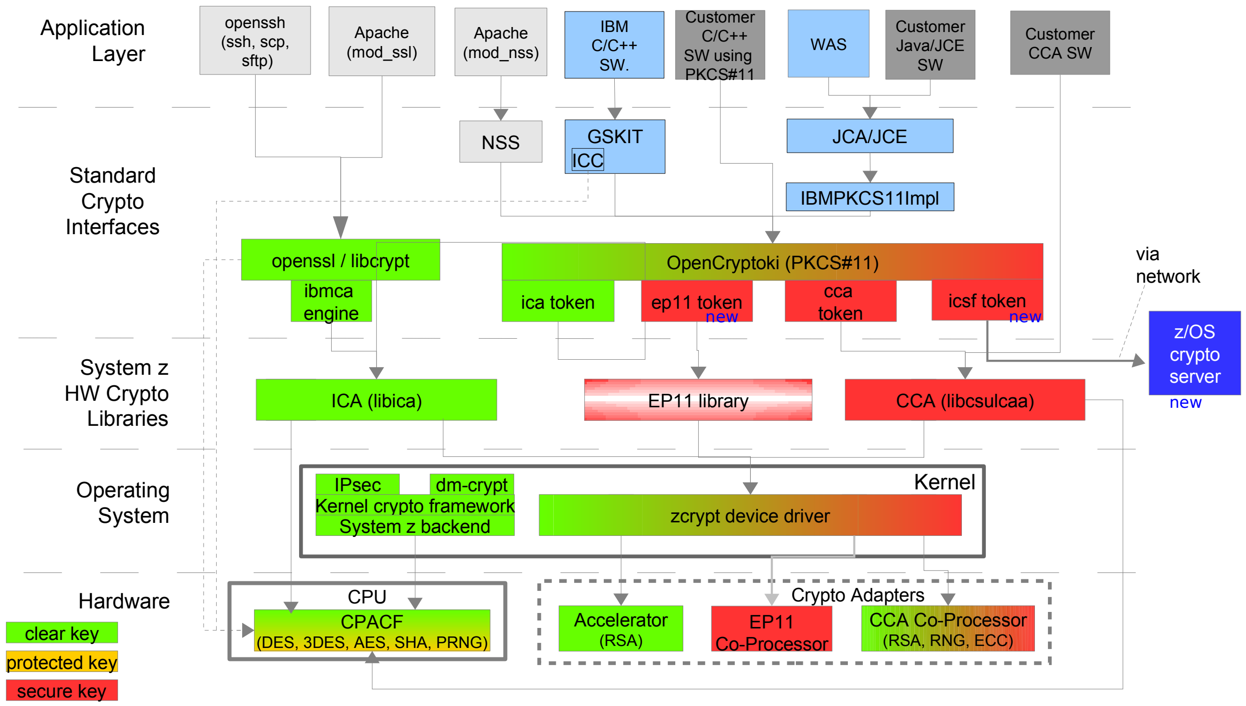
Crypto in general: Performance

- Nachteil von Secure key Verfahren: Performance
- Falls Regularien (intern, extern) es zulassen und hohe Performance benötigt wird: Protected key

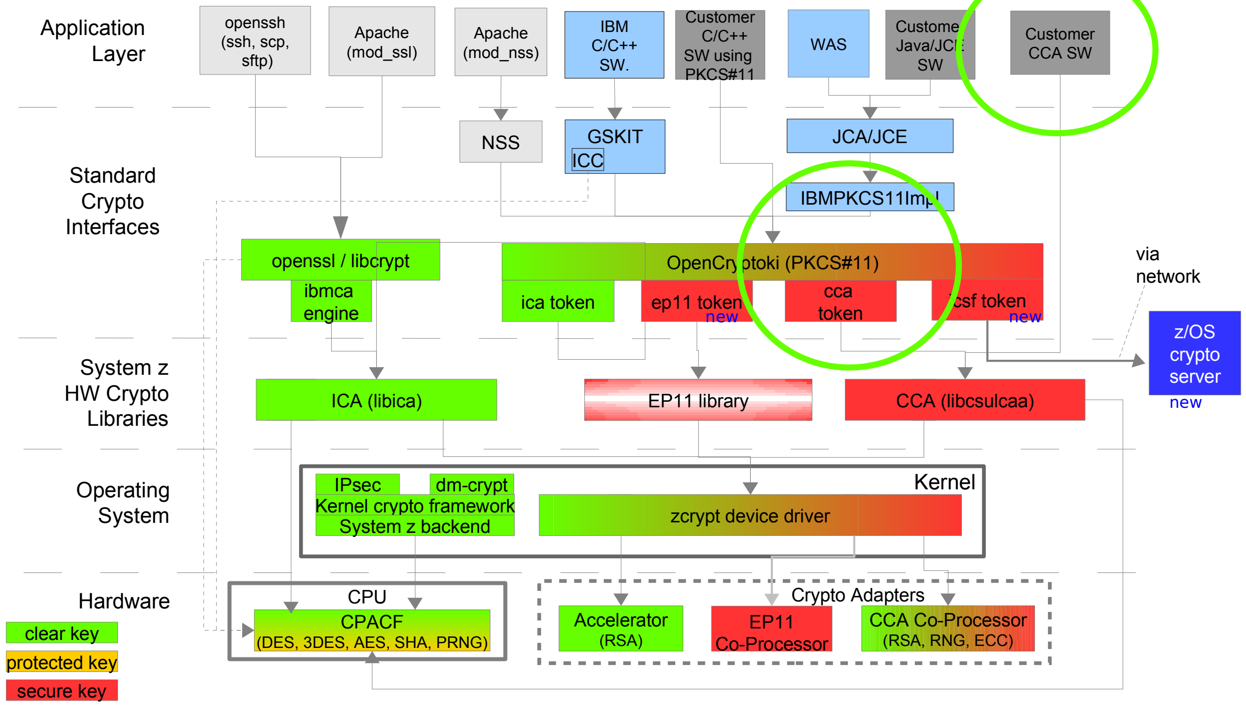


Note: Above figure is not based on official benchmark results, it contains only a preliminary first impression

Crypto in general: Linux on z Crypto Stack



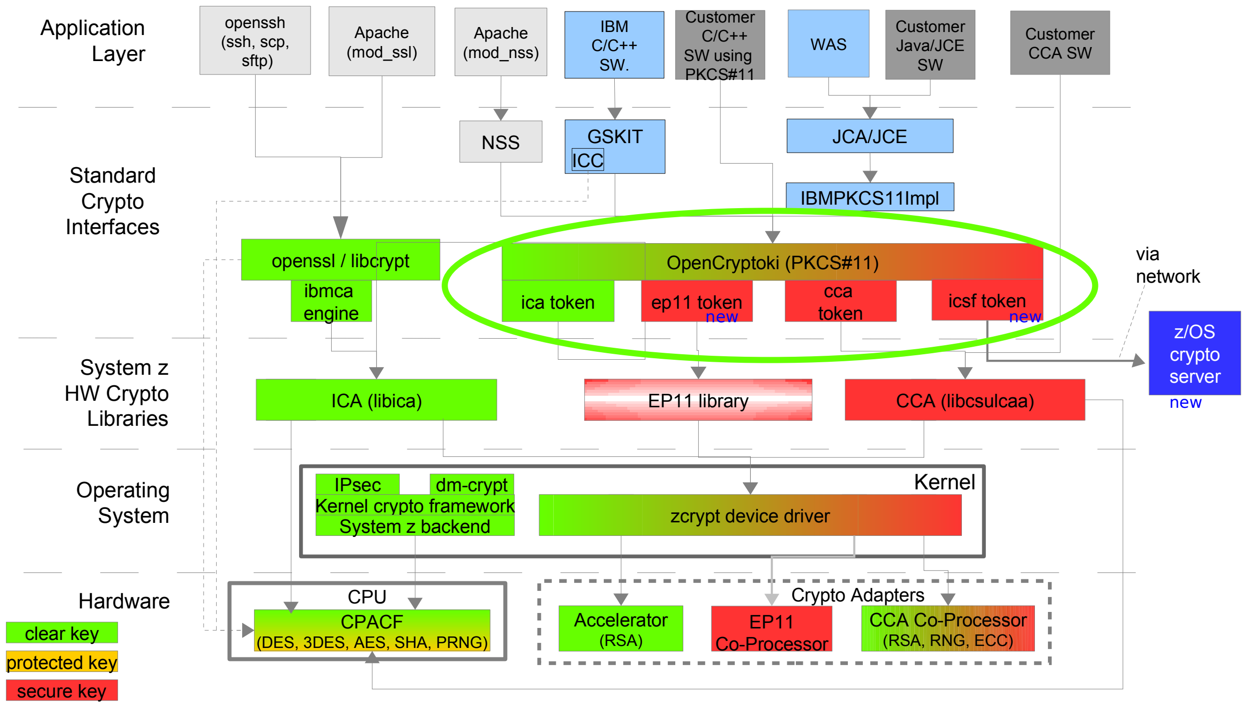
Crypto in general: Linux on z Crypto Stack



Linux auf z und CCA

- CCA:
  - Direkt auf CCA library
  - Über PKCS11 Schnittstelle (openCryptoki)
- Existierende CCA Anwendungen auf Linux for System z portieren
- Gut dokumentiertes Interface
- CCA Wechsel der Version: token in neuem Format abgespeichert – Migrationstool ist vorhanden
- Import von Schlüssel: Ein paar Einschränkungen (nur TDES, RSA in CRT Format
- Neue Anwendungen auf PKCS11 Schnitttstelle
- Existierende PKCS11 Anwendungen (distr. Umfeld) portieren (falls nötige Unterstützung vorhanden ist)
  – Bsp.: Java Crypto Provider nutzt OpenCryptoki (clear und secure key möglich)

Crypto in general: Linux on z Crypto Stack



What is the PKCS#11 Standard?

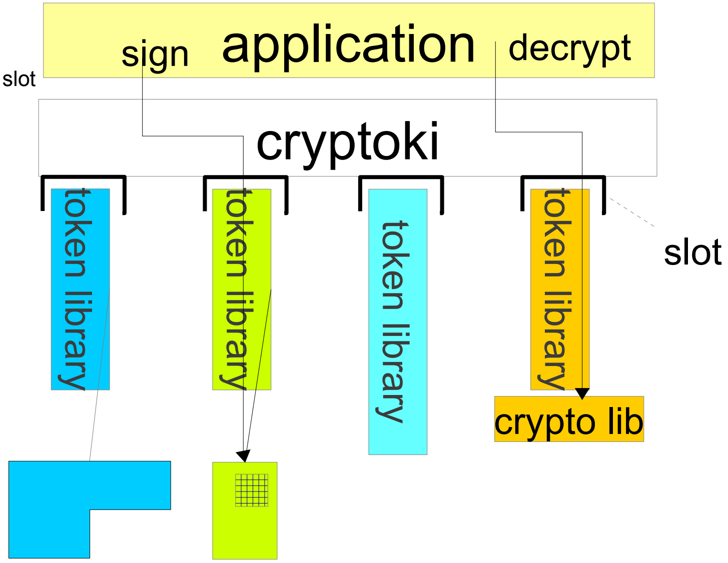
- Published by RSA
  - formerly hosted by RSA now being moved to OASIS
- cryptoki - name of C/C++ API
- current version: 2.2 (+ 3 amendments)
- draft version: 2.4
- widely recognized
  - but interpretation of some details in the standard varies, allows for HW specific feature
  - conservative interpretation is recommended to be on the safe side

PKCS#11 Concepts

- slots and tokens
- roles and session
- functions and mechanisms
- objects and keys (certificates)
- miscellaneous

PKCS#11 Concepts: Slots and Tokens

- Model: smart cards and readers
  - reader: slot
  - crypto processor: token to be inserted in slot
- slots and tokens may be HW specific
- slot and token functions
  - C_GetSlotList(), C_GetSlotInfo(),
  - C_WaitForSlotEvent()
  - C_InitToken(), C_GetTokenInfo()
  - C_initPIN(), C_SetPIN()
- slot info
  - token present
  - device removable
  - ...
- token info
  - login required,
  - too many wrong pins entered
  - has RNG
  - ...



PKCS#11 Concepts: Roles and Session

Roles

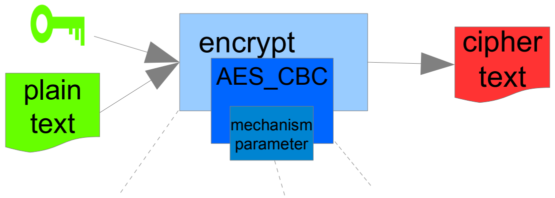
- Security Officer (SO) - one per token
  - has SO pin
  - initializes tokens
  - grants token to the normal user
    - can set user pin
- (Normal) User - one per token
  - has user PIN
  - can login to sessions
  - can create and access private objects
  - can perform cryptographic operations

Sessions

- context for crypto operations
- related to a token
- maintains state of multi-part functions
- “only one” operation per session at a time

- session types
  - read-only / read write
  - public / user session
- user session - after login
- Session functions
  - C_OpenSession() / C_CloseSession()
  - C_GetSessionInfo()
  - C_GetOperationState() / C_SetOperationState()
  - C_Login() / C_Logout()

PKCS#11 Concepts: Cryptographic Functions and Mechanisms



cryptographic function

- a generic cryptographic function
  - e.g. C_Encrypt(), C_Sign()
- runs ins the context of a session
- must be instanciated by a mechanism
- C_Init*Fkt*(session, mechanism, key,...)
- “single part” functions
  - C_*Fkt*Init(); C_*Fkt*()
- multi part functions (to process long messages)
  - C_*Fkt*Init(), C_*Fkt*Update(), ..., C_*Fkt*Update(); C_*Fkt*Final()
- it is token specific which functions are supported
  - C_GetFunctionList()

mechanism

- set of specific cryptographic processes (e.g. CKM_AES_CBC)
- used to implement cryptographic functions
- mechanism attributes defined in CK_MECHANISM_INFO structure
  - min/max key sizes
  - flags denoting supported functions
  - HW support flag
- mechanisms may have parameters, e.g. to specify the IV for CKM_xyz_CBC
- set of mechanism and their supported functions is token specific
  - C_GetMechanismList()
  - C_GetMechanismInfo()

PKCS#11 Cryptographic Functions

Fkt	C_*Fkt*Init	C_*Fkt*	C_*Fkt*Update	C_*Fkt*Final	Comment
Encrypt	x	x	x	x	
Decrpt	x	x	x	x	
Digest	x	x	x	x	no key arg for DigestInit
DigestKey		x			used like DigestUpdate
Sign	x	x	x	x	
SignRecover	x	x			single part function
Verify	x	x	x	x	
VerifyRecover	x	x			single part function
DigestEncrypt			x		each subfunction must be individually initialized and finalized
DecryptDigest			x		
SignEncrypt			x		
DecryptVerify			x		
GenerateKey		x			symmetric key
GenerateKeyPair		x			pair of asymmetric keys
WrapKey		x			implicit initialization
UnwrapKey		x			
DeriveKey		x			
SeedRandom		x			does not use mechanism
GenerateRandom		x			

C_VerifyUpdate()

PKCS#11 Mechanisms (Examples)

Mechanism	supported functions						
	Encrypt Decrypt	Sign Verify	SR VR	Digest	Gen Key / Key Pair	Wrap Unwrap	Derive
CKM_RSA_PKCS_KEY_PAIR_GEN					x		
CKM_RSA_PKCS	x	x	x			x	
CKM_SHA256_RSA_PKCS		x					
…							
CKM_EC_KEY_PAIR_GEN					x		
CKM_ECDSA		x					
CKM_ECDSA_SHA1		x					
CKM_ECDH1_DERIVE							x
CKM_AES_KEY_GEN					x		
CKM_AES_ECB	x					x	
CKM_AES_CBC	x					x	
CKM_AES_CBC_PAD	x					x	
CKM_AES_CTR	x					x	
CKM_AES_MAC		x					
CKM_SHA256				x			
CKM_SHA256_HMAC_GENERAL		x					
CKM_SHA256_HMAC		x					
CKM_SHA256_KEY_DERIVATION							x

This table has ca 300 rows in version 2.2 of the PKCS#11 standard.

PKCS#11 Concepts: Objects and Keys (Certificates)

Objects

- object classes
  - session objects (volatile) vs token objects (persistent)
  - private vs public
  - read-only vs read-write
- private objects
  - a user session (login!) is required to access private objects
- objects attributes
  - type, value, value length
- object management functions
  - C_CreateObject(),
  - C_CopyObject()
  - C_DestroyObject()
  - C_GetObjectSize()
  - C_{Get|Set}AttributeValue()
  - C_FindObjects[Init|Final]()

Key Objects

- private keys
- public keys
- secret keys (i.e. symmetric)

- some key object attributes:
  - CKA_WRAP
  - CKA_SENSITIVE (not for public keys)
  - CKA_MODULUS (RSA only)
- key management functions
  - C_GenerateKey()
  - C_GenerateKeyPair()
- to import a key
  - use C_UnwrapKey() or C_CreateObject() setting all key attributes

Certificate objects

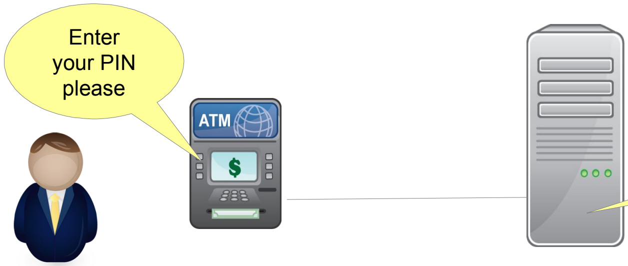
- X.509
- WTLS
- no functions on certificates

PKCS#11 Concepts: Miscellaneous

- supporting parallel access to tokens (e.g. multi threading)
  - C_Initialize() has an agrument that describes
    - threading capabilities
    - synchronization functions (Mutex)

- slot management
  - C_WaitForSlotEvent()
    - e.g. token insertion / removal

- token PIN entry at physical token
  - CKF_PROTECTED_AUTHENTICATION_PATH flag must be set in token info
  - C_Login() is then called with NULL_PTR as PIN



```
...
C_Login(session,
          CKU_USER,NULL_PTR,0);
...
```

A Typical PKCS#11 Flow (Simplified C Code, no Error Handling, ...)

```
#include <pkcs11types.h>
...
rc = c = C_Initialize(...);
rc = C_GetSlotList(...);
rc = C_GetSlotInfo(slot,...);
rc = C_GetTokeInfo(slot,...);
rc = C_OpenSession(slot,...,
&session);
rc = C_Login(session, ...)


rc = C_Logout(session)
rc = C_CloseSession(session)
rc = C_Finalize(...)
```

Initialization / session handling

```
rc = C_EncryptInit(session, mechanism, key);
while (/*there are still pieces of the message*/) {

    rc = C_EncryptUpdate(session, message_part,...
    );
    ...
}
rc = C_EncryptFinal(session, last_part, ...);
...
```

cryptographic operation(s)
inside a session

OpenCryptoki

- open source implementation of PKCS#11 version 2.2 (C API)
  - maintained by IBM (LTC)
  - source: http://sourceforge.net/projects/opencryptoki/
  - latest versions
    - 2.x branch (available with most distributions): 2.4.3.1
    - 3.x branch: 3.1
- comes with support for different tokens
- shipped with
  - RHEL 6.5: version 2.4.3.1
  - RHEL 7.0: version 3.0
  - SLES 11 SP3 (incl. maintenance): version 2.4.3.1

OpenCryptoki Components

- `libopencryptoki.so`: a library for the generic part of opencryptoki → PKCS#11 APIs
- one *s*lot *t*oken *d*ynamic *l*ink *l*ibrary (*stdll*) per token
- `pkcsconf`: a configuration tool
- `pkcsslotd`: slot manager daemon,maintains a shared memory region to arbitrate access to shared token resources by multiple processes
- `pkcs11_startup` (v 2.x): script for initial configuration
  - creates non-customizable configuration file `/var/lib/opencryptoki/pk_config_data`
- `/etc/opencryptoki/opencryptoki.conf` (v 3.x): a customizable configuration file
- `/var/lib/opencryptoki`: a directory containing
  - configuration file `pk_config_data` generated by `pkcs11_startup` (only pre v 3.0)
  - token specific directories containing
    - `NVTOK.DAT`: configuration data and state
    - `MK_SO`: an encrypted master key to encrypt SO's private objects
    - `MK_USER`: an encrypted master key to encrypt the user's private objects
    - `TOK_OBJ`: a directory for token objects
      - each private object is represented by an encrypted file
- man pages
  - v 2.4.3.1: pkcsconf(1), pkcs_startup(1), pkcs_slot(1), pkcsicfs(1), pk_config_data(5), opnecryptoki(7), pkcsslotd(8)
  - v 3.x: pkcsconf(1), pkcsicsf(1), opencryptoki.conf(5), opnecryptoki(7), pkcsslotd(8),

Installing OpenCryptoki & Token Configuration

base configuration

- install openCryptoki via anaconda/yum (Red Hat), yast/zypper (SUSE) or RPM
- make sure token prerequisites are installed (see token specific info)
- v 2.4.x: run `pkcs11_startup`
- v 3.x: edit `/etc/opencryptoki/opencryptoki.conf` if needed
- start pkcsslotd
- processes calling openCryptoki must be members of the group pkcs11, add Unix group pkcs11 if needed

- initialize each token:
  1) SO initializes token (sets token label): `pkcsconf -i -c` <slot id>
  2) SO changes SO PIN (from default 87654321): `pkcsconf -P -c` <slot id>
  3) SO sets user pin: `pkcsconf -u -c` <slot id>
  4) user changes user pin: `pkcsconf -p -c` <slot id>

- useful commands
  - list all tokens: `pkcsconf -t`
  - show mechanism list of a token: `pkcsconf -m -c` <slot id>
  - more info on pkcsconf: `pkcsconf -h` or `man pkcsconf`

Limits and Restrictions of OpenCryptoki

- processes calling the openCryptoki library must be members of the Unix group pkcs11
- tokens
  - maximal number of slots/tokens: 32
  - there is only one instance of each token
  - only static (non-removable) tokens are supported
    - no support for C_WaitForSlotEvent()
- multi threading:
  - only supported with locking from native operating system

Token Objekte sind in FS abgespeichert, alle user aus pkcs11 Gruppe haben gleiche Rechte:
Ggf. mehrere unterschiedliche Linux Server nutzen

OpenCryptoki tokens for Linux on z

- **ica token**
  - provides clear key cryptographic functions
  - exploits CPACF, CryptoExpress accelerators and CCA co-processors
  - System z specific
- **cca token**
  - provides secure key cryptographic functions
  - exploits CryptoExpress CCA co-processors
  - System z specific
- **soft token**
  - provides clear key cryptographic functions
  - pure software implementation, relies on libcrypt (openssl)
  - platform independent
- **icsf token** (since openCryptoki 3.0)
  - remote access to cryptographic functions on a z/OS based ICFS crypto server
  - uses LDAP protocol
  - platform independent
- **ep11 token** (since openCryptoki 3.1)
  - provides secure key cryptographic functions
  - exploits CryptoExpress EP11 co-processors
  - System z specific

ICA Token

- prerequisites
  - CPACF feature installed on system,
  - libica library installed,
  - z90crypt kernel module loaded to exploit CryptoExpress adapters
- `pkcsconf -t` shows attribute `Modell: IBM ICA` for ica token
- token directory: `/var/lib/opencryptoki/lite`
  - token objects are stored in `/var/lib/opencryptoki/lite/TOK_OBJ`
- hardware exploitation:
  - SHA-1, SHA-256/386/512 via CPACF
  - DES, 3DES, AES128/192/256 using ECB, CBC, CTR modes of operation via CPACF
  - starting with version 3.0: 3DES, AES128/192/256 using OFB, CFB, CBC-MAC modes of operation via CPACF
  - RSA with1024-4096 bit keys using Crypto Express adapters or software fall back (openssl)
  - (pseudo) random numbers: CPACF and Crypto Express CCA coprocessor if available

ICA Token Mechanisms (openCryptoki 3.1)

Mechanism	supported functions						
	E/D	S/V	SR	Dig	Gen	W/U	Der
CKM_RSA_PKCS_KEY_PAIR_GEN					x		
CKM_RSA_PKCS	x	x	x			x	
CKM_PKCS_X_509	x	x	x			x	
CKM_MD5_RSA_PKCS		x					
CKM_SHA1_RSA_PKCS		x					
CKM_SHA256_RSA_PKCS		x					
CKM_SHA384_RSA_PKCS		x					
CKM_SHA512_RSA_PKCS		x					
CKM_SSL3_PRE_MASTER_KEY_GEN					x		
CKM_SSL3_MASTER_KEY_DERIVE							x
CKM_SSL3_KEY_AND_MAC_DERIVE							x
CKM_SSL3_MD5_MAC		x					
CKM_SSL3_SHA1_MAC		x					
CKM_SHA256				x			
CKM_SHA256_HMAC		x					
CKM_SHA256_HMAC_GENERAL		x					
CKM_SHA384				x			
CKM_SHA384_HMAC		x					
CKM_SHA384_HMAC_GENERAL		x					
CKM_SHA512				x			
CKM_SHA512_HMAC		x					
CKM_SHA512_HMAC_GENERAL		x					
CKM_MD5				x			
CKM_MD5_HMAC		x					
CKM_MD5_HMAC_GENERAL		x					
CKM_SHA1				x			
CKM_SHA1_HMAC		x					
CKM_SHA1_HMAC_GENERAL		x					

Mechanism	supported functions						
	E/D	S/V	SR	Dig	Gen	W/U	Der
CKM_DES_KEY_GEN					x		
CKM_DES_ECB	x					x	
CKM_DES_CBC	x					x	
CKM_DES_CBC_PAD	x					x	
CKM_DES_CBC_MAC		x					
CKM_DES_CBC_MAC_GENERAL		x					
CKM_DES_OFB64	x						
CKM_DES_CFB8	x						
CKM_DES_CFB64	x						
CKM_DES3_KEY_GEN					x		
CKM_DES3_ECB	x					x	
CKM_DES3_CBC	x					x	
CKM_DES3_CBC_PAD	x					x	
CKM_DES3_CBC_MAC		x					
CKM_DES3_CBC_MAC_GENERAL		x					
CKM_DES3_OFB64	x						
CKM_DES3_CFB8	x						
CKM_DES3_CFB64	x						
CKM_AES_KEY_GEN					x		
CKM_AES_ECB	x					x	
CKM_AES_CBC	x					x	
CKM_AES_CBC_PAD	x					x	
CKM_AES_CTR	x					x	
CKM_AES_CBC_MAC		x					
CKM_AES_CBC_MAC_GENERAL		x					
CKM_AES_OFB	x						
CKM_AES_CFB8	x						
CKM_AES_CFB64	x						
CKM_AES_CFB128	x						

blue: since version 2.4.3, green since 3.0

Example: Configure openCryptoki 2.4.x with ICA Token

- load crypto adapter

```
# modprobe z90crypt
```

- install libica (if needed)
  - check: `rpm -qa | grep -i libica`
  - install for RHEL: `yum install libica`
  - install for SLES: `zypper install libica`
- initialize openCryptoki

```
# pkcs11_startup
# pkcsslotd
```

- check which tokens are available

```
# pkcsconf -t
Token #0 Info:
        Label:
        Manufacturer: IBM Corp.
        Model: IBM ICA
       ...
Token #1 Info:
        Label:
        Manufacturer: IBM Corp.
        Model: IBM CCA Token
        ...
Token #2 Info:
        Label:
        Manufacturer: IBM Corp.
        Model: IBM SoftTok
        ...
```

- Set label of ica token

```
# pkcsconf -I -c0
Enter the SO PIN: ********

Enter a unique token label: icatoken
```

- change SO pin of ica token:

```
# pkcsconf -P -c0
Enter the SO PIN: ********
Enter the new SO PIN: ********
Re-enter the new SO PIN: ********
```

- set user pin of ica token:

```
# pkcsconf -u -c0

Enter the SO PIN: ********

Enter the new user PIN: ********

Re-enter the new user PIN: ********
```

- change user pin of ica token:

```
# pkcsconf -p -c0

Enter the user PIN: ********

Enter the new user PIN: ********

Re-enter the new user PIN: ********
```

- verify configuration of ica token

```
# pkcsconf -t -c0

Token #0 Info:

  Label: icatoken

  ...

  Flags: 0x44D (RNG|LOGIN_REQUIRED|

  USER_PIN_INITIALIZED|CLOCK_ON_TOKEN      |

TOKEN_INITIALIZED)
```

CCA Token

- prerequisites
  - libcsulcca library installed
    - http://www-03.ibm.com/security/cryptocards/pciecc/ordersoftware.shtml
  - z90crypt kernel module loaded to exploit CryptoExpress CCA co-procssor
  - master key(s) must be set in CCA co-processor
- `pkcsconf -t` shows attribute `Modell: IBM CCA TOKEN` for cca token
- token directory: `/var/lib/opencryptoki/ccatok`
  - token objects are stored in `/var/lib/opencryptoki/ccatok/TOK_OBJ`
- hardware exploitation:
  - DES, DES3, AES128/192/256 using ECB, CBC via Crypto Express CCA coprocessor
  - RSA with1024-4096 bit keys via Crypto Express CCA coprocessor
  - ECDSA via Crypto Express 3 (or later) CCA coprocessor
- all keys generated are CCA secure keys
- importing keys some restrictions apply, e.g.:
  - RSA key pairs can be imported with a call to C_CerateObject() for each key component
  - the imported private RSA key will be wrapped with the CCA master key
  - only private RSA keys in CRT format can be imported

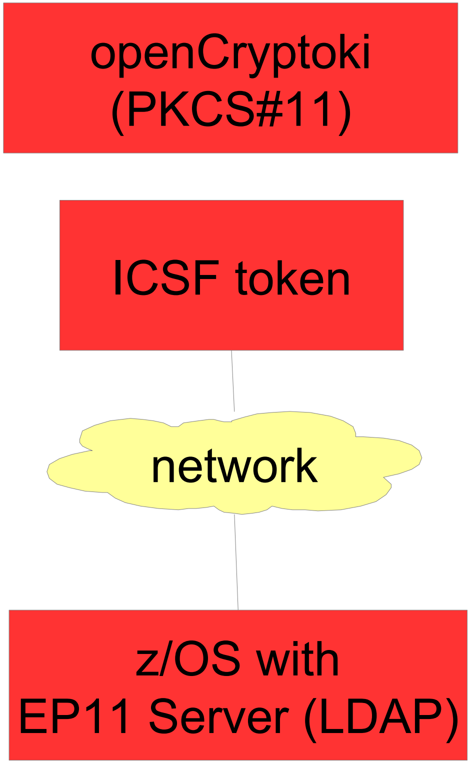
CCA Token Mechanisms (openCryptoki 2.4.3.1)

	supported functions						
Mechanism	**E/D**	**S/V**	**SR**	**Dig**	**Gen**	**W/U**	**Der**
CKM_RSA_PKCS_KEY_PAIR_GEN					x		
CKM_RSA_PKCS	x	x					
CKM_PKCS_X_509							
CKM_MD5_RSA_PKCS		x					
CKM_SHA1_RSA_PKCS		x					
CKM_SHA256_RSA_PKCS		x					
CKM_ECDSA_KEY_PAIR_GEN					x		
CKM_ECDSA		x					
CKM_ECDSA_SHA1		x					
CKM_SHA256				x			
CKM_SHA256_HMAC		x					
CKM_SHA256_HMAC_GENERAL		x					
CKM_MD5				x			
CKM_MD5_HMAC		x					
CKM_MD5_HMAC_GENERAL		x					
CKM_SHA1				x			
CKM_SHA1_HMAC		x					
CKM_SHA1_HMAC_GENERAL		x					
CKM_DES_KEY_GEN					x		
CKM_DES_CBC	x						
CKM_DES_CBC_PAD	x						
CKM_DES3_KEY_GEN					x		
CKM_DES3_CBC	x						
CKM_DES3_CBC_PAD	x						
CKM_AES_KEY_GEN					x		
CKM_AES_ECB	x						
CKM_AES_CBC	x						
CKM_AES_CBC_PAD	x						

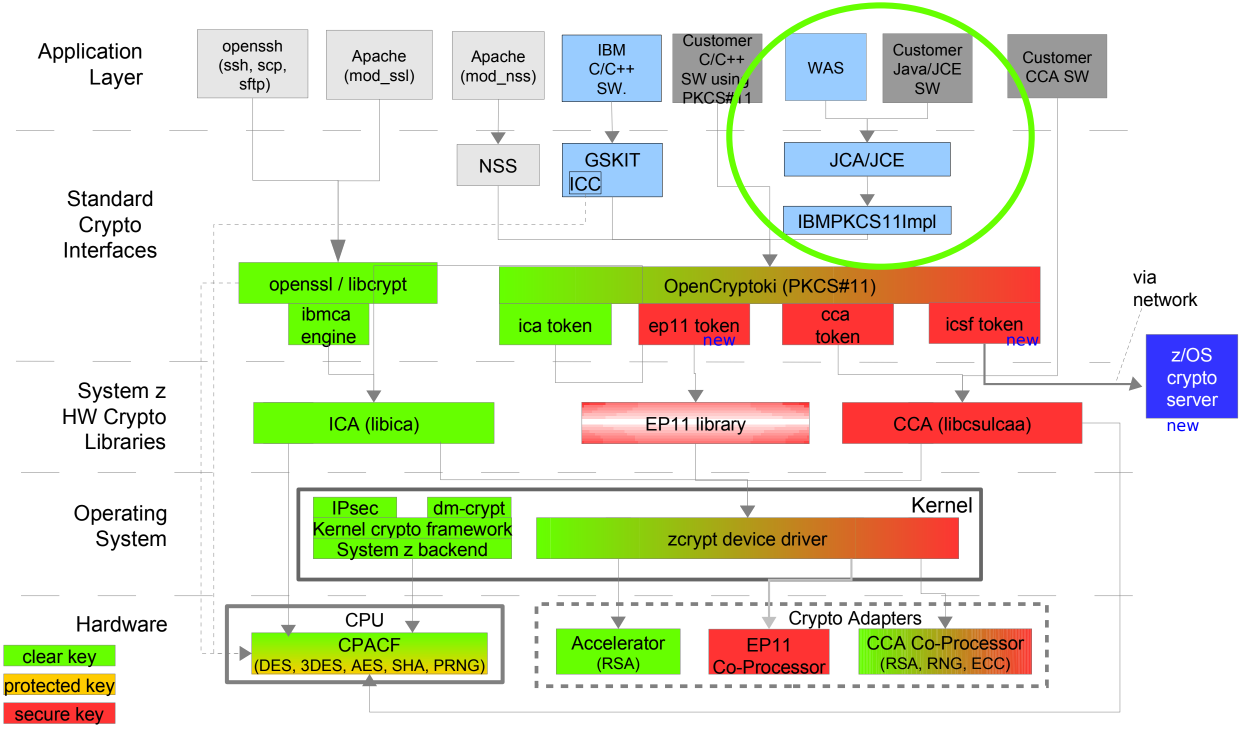
blue: since version 2.4.1

ICSF Token Overview

- Linux client part of z/OS Crypto as a Service
- available with z/OS 2.1 and RHEL 7.0
- icsf token forwards crypto requests to ICSF on z/OS server
  - using LDAP protocol
  - Simple and SASL authentication
- key objects are stored on z/OS server
- requires LDAP client set up on Linux
- `pkcsicsf` utility
  - sets up icsf token configuration in opencryptoki.conf
  - can provide LDAP bind information to openCryptoki
  - may prompt for RACF password for simple authentication
- token directory `/var/lib/opencryptoki/icsf`
- token configration file to be refered to in opencryptoki.conf

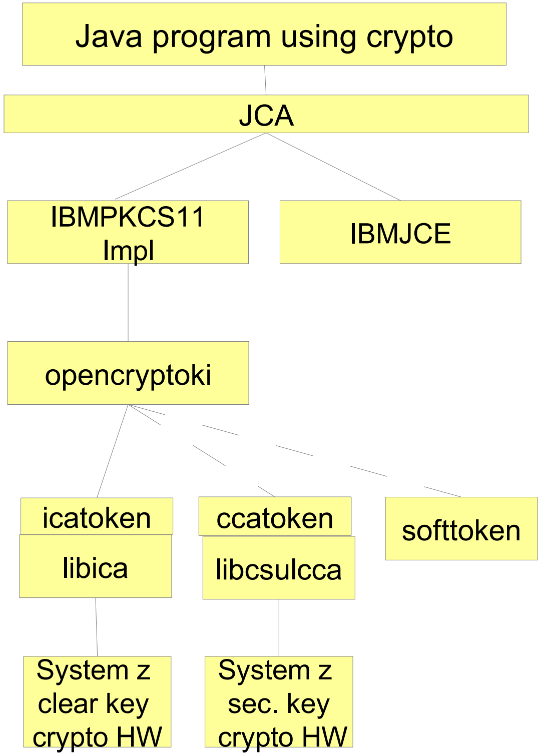


Crypto in general: Linux on z Crypto Stack



PKCS#11 und Java
The Java Cryptographic Architecture (JCA)

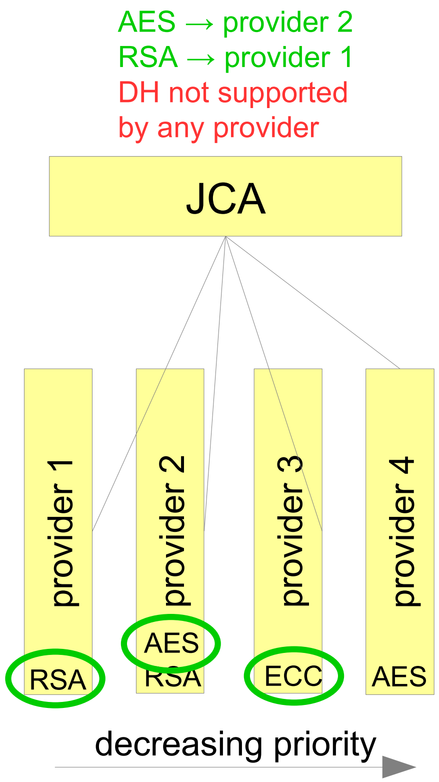
- Java Cryptographic Architecture (JCA)
  - provider architecture for security APIs
  - supports multiple providers with different priorities and capabilities
  - providers that implement the JCE API:
    - IBMJCE (software implementation by IBM equivalent to SunJCE)
    - IBMPKCS11Impl calls openCryptoki which can be configured to use a specific token to exploit crypto HW support
      - clear key crypto via libica
      - secure key crypto via CCA library
- Java Cryptographic Extension (JCE)
  - API for basic cryptographic functions


Java program using crypto
JCA
IBMPKCS11 Impl
IBMJCE
opencryptoki
icatoken
libica
ccatoken
libcsulcca
softtoken
System z clear key crypto HW
System z sec. key crypto HW

Implicit Selection of the IBMPKCS11Impl Provider

the Java Cryptographic Architecture (JCA)

- provides plug-in mechanism for providers of cryptographic functions
- *XXX*getInstance() function selects provider for class *XXX*
- implicit provider selection:
  - no provider defined in *XXX*getInstance() call
  - for each crypto functions selects provider based on provider capability and priority
  - provider priority is defined by provider sequence in `java.security` file

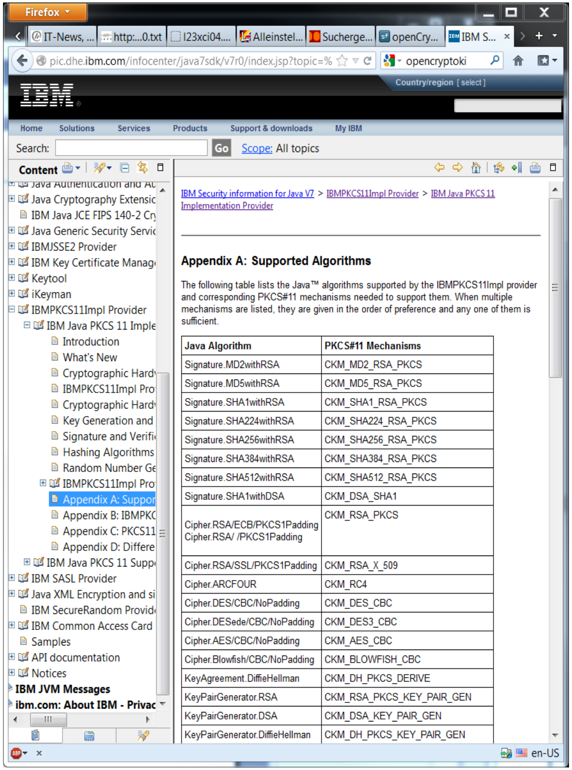


IBMPKCS11Impl: Supported Algorithms

The set of Java crypto algorithms (objects) supported by crypto HW is the **intersection** of

- the capabilities of the IBMPKCS11Impl provider
- and the mechanisms supported by opencryptoki token.

- The list of Java crypto algorithms that the IBMPKCS11Impl provider can support in theory as described in
  - http://pic.dhe.ibm.com/infocenter/java7sdk/v7r0/topic/com.ibm.java.security.component.doc/security-component/pkcs11implDocs/supportedalgorithms.html



Firefox | IT-News, ... | http:...0.txt | I23xci04.... | Alleinstel... | Sucherge... | openCry... | IBM S...

pic.dhe.ibm.com/infocenter/java7sdk/v7r0/index.jsp?topic=% | opencryptoki

IBM | Country/region [select]

Home | Solutions | Services | Products | Support & downloads | My IBM

Search: Go Scope: All topics

Content

- Java Authentication and Au
- Java Cryptography Extensic
- IBM Java JCE FIPS 140-2 Cry
- Java Generic Security Servic
- IBMJSSE2 Provider
- IBM Key Certificate Manag
- Keytool
- iKeyman
- IBMPKCS11Impl Provider
  - IBM Java PKCS 11 Imple
    - Introduction
    - What's New
    - Cryptographic Hardw
    - IBMPKCS11Impl Pro
    - Cryptographic Hardw
    - Key Generation and
    - Signature and Verifi
    - Hashing Algorithms
    - Random Number Ge
    - IBMPKCS11Impl Pro
    - Appendix A: Suppor
    - Appendix B: IBMPKC
    - Appendix C: PKCS11
    - Appendix D: Differe
  - IBM Java PKCS 11 Supp
- IBM SASL Provider
- Java XML Encryption and si
- IBM SecureRandom Provid
- IBM Common Access Card
- Samples
- API documentation
- Notices
- **IBM JVM Messages**
- **ibm.com: About IBM - Privac**

IBM Security information for Java V7 > IBMPKCS11Impl Provider > IBM Java PKCS 11 Implementation Provider

Appendix A: Supported Algorithms

The following table lists the Java™ algorithms supported by the IBMPKCS11Impl provider and corresponding PKCS#11 mechanisms needed to support them. When multiple mechanisms are listed, they are given in the order of preference and any one of them is sufficient.

Java Algorithm	PKCS#11 Mechanisms
Signature.MD2withRSA	CKM_MD2_RSA_PKCS
Signature.MD5withRSA	CKM_MD5_RSA_PKCS
Signature.SHA1withRSA	CKM_SHA1_RSA_PKCS
Signature.SHA224withRSA	CKM_SHA224_RSA_PKCS
Signature.SHA256withRSA	CKM_SHA256_RSA_PKCS
Signature.SHA384withRSA	CKM_SHA384_RSA_PKCS
Signature.SHA512withRSA	CKM_SHA512_RSA_PKCS
Signature.SHA1withDSA	CKM_DSA_SHA1
Cipher.RSA/ECB/PKCS1Padding Cipher.RSA/ /PKCS1Padding	CKM_RSA_PKCS
Cipher.RSA/SSL/PKCS1Padding	CKM_RSA_X_509
Cipher.ARCFOUR	CKM_RC4
Cipher.DES/CBC/NoPadding	CKM_DES_CBC
Cipher.DESede/CBC/NoPadding	CKM_DES3_CBC
Cipher.AES/CBC/NoPadding	CKM_AES_CBC
Cipher.Blowfish/CBC/NoPadding	CKM_BLOWFISH_CBC
KeyAgreement.DiffieHellman	CKM_DH_PKCS_DERIVE
KeyPairGenerator.RSA	CKM_RSA_PKCS_KEY_PAIR_GEN
KeyPairGenerator.DSA	CKM_DSA_KEY_PAIR_GEN
KeyPairGenerator.DiffieHellman	CKM_DH_PKCS_KEY_PAIR_GEN

en-US

Configuring Java for HW Crypto Usage

The `java.security` file maintains a list of available JCA providers

standard location:

```
      /usr/lib/jvm/java-<version>-ibm-<ext. version>.s390x/jre/lib/security/java.security
```

Example extract from java.security

```
      ...
      # List of providers and their preference orders (see above):
      #
      security.provider.1=com.ibm.crypto.pkcs11impl.provider.IBMPKCS11Impl /root/zpkcs.cfg
      security.provider.2=com.ibm.crypto.provider.IBMJCE
      #security.provider.3=com.ibm.security.jgss.IBMJGSSProvider
      ...
```

The IBMPKCS11Impl has a configuration file as argument

Example configuration file for IBMPKCS11Impl:

```
      name = Sample
      description = Sample config for z/linux
      library = /usr/lib64/pkcs11/PKCS11_API.so
      # the following references the icatoken
      slot = 0
      # the following references the ccatoken
      #slot = 1
      # the following references the softtoken
      #slot = 2
      disabledmechanisms = { CKM_SHA_1 }
```

Achtung:
Syntax ist sehr “empfindlich”
SHA1 in SW!

Explicitly Selecting the IBMPKCS11Impl Provider

Without explicit argument JCA uses the first provider in `java.security` that provides an object.

- `aesCipher = Cipher.getInstance("AES/ECB/NoPadding");`

When generating an object instance the provider can be explicitly specified:

optional provider argument

- `aesCipher = Cipher.getInstance("AES/ECB/NoPadding","IBMPKCS11Impl-Sample");`

provider base name as in java.security

provider suffix derived value of attribute name from provider configuration file specified in java.security

```
.../java.security:

...
security.provider.1=com.ibm.crypto.pkcs11impl.provider.IBMPKCS11Impl /root/zpkcs.cfg
...
```

```
/root/zpkcs.cfg:
name = Sample
...
```

PKCS#11 and Standard SW

standard middleware often provides for a plug-in option for PKCS#11 libraries

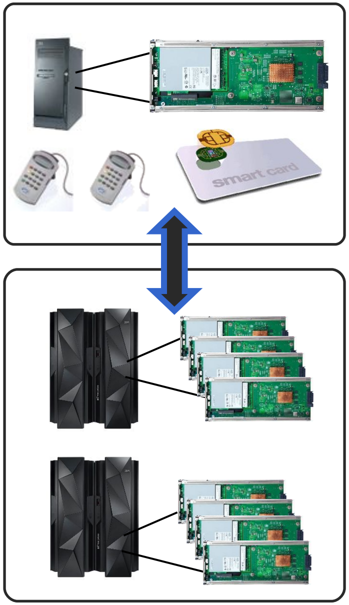
- IBM WebSphere Application Server (WAS) via Java
- Also other Application Server via Java
- IBM HTTP Server (IHS) via GSKIT
- NSS

configuration files of such software may allow to specify

- library path of opencryptoki
- slot or token id
- user PIN

Administration of Master Keys

- Bordmittel für Test – panel.exe (nur CCA)
- Trusted Key Entry Station (TKE) für Produktion
  - z/OS
  - Linux for System z:
    - Für CCA: TKE spricht mit catcher.exe
    - Für EP11:TKE spricht mit ep11TKEd
  - Rollen, 4-Augen-Prinzip
  - Mit TKE alle CEC4S eines CECs managen!



Administration of User / Application Keys

- Möglicherweise große Zahl von Benutzer- oder Anwendungs-Schlüssel und Zertifikaten

Zusammenfassung

- Linux for System z:
  - Requirements are known
  - Methods and tools are available
  - Clear key crypto (SW) is easy. If supported using HW crypto features of z for clear key is easy
  - HSM is required for secure key crypto. Crypto Express can be used as HSM
  - Setup and configuration for secure key requires planning and “some” effort r also Secure-Key methods
  - If CCA and EP11 (PKCS11) is required, you need separate cards
  - TKE for handling Masterkeys (for EP11 TKE is required)
  - Handling of user keys is extra topic
- If handling with sensitive data, sooner of later you will need to use cryptographic methods

Appendix

More Information

- PCI Security Standards Council (PCI SCC) website www.pcisecuritystandards.org
- PKCS#11 Artikel: http://enterprisesystemsmedia.com/article/using-linux-on-system-z-hardware-cryptography-with-the-pkcs11-cryoptography#sr=g&m=o&cp=or&ct=-tmc&st=%28opu %20qspwjefe%29&ts=1411391472
- CCA RPM download from http://www-03.ibm.com/security/ cryptocards/pciecc/ordersoftware.shtml
- Redbooks
- CCA documentation
- IBM PKCS#11 (EP11 token) documentation

Crypto Driver + Bibliotheken in Linux Distro

CCA und EP11 Bibliothek von IBM (also nicht Open Source)

- Heute: Anwendung soll nicht direkt EP11 Bibliothek nutzen, sondern PKCS11 Interface (openCryptoki)