PV204 Security technologies

Hardware Security Modules (HSM), crypto in cloud

Petr Švenda ✉ ***svenda@fi.muni.cz*** ***@rngsec***

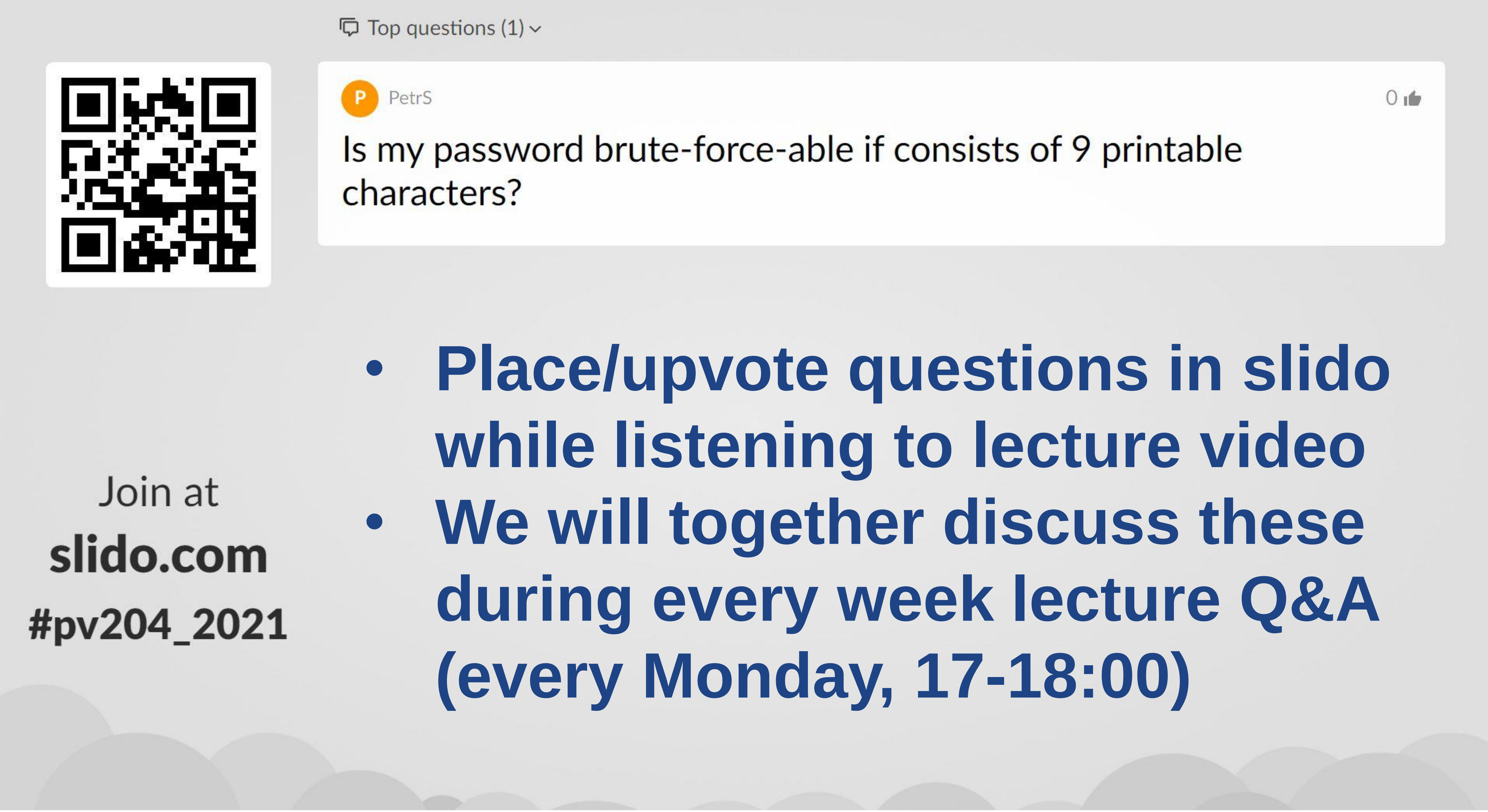
Centre for Research on Cryptography and Security, Masaryk University

CROCS

Centre for Research on Cryptography and Security

Please comment on slides with anything unclear, incorrect or suggestions for improvement

https://drive.google.com/file/d/1fYCLrz6cZmJVUYpY0EwYaV0vaUHW8p6N/view?usp=sharing



Top questions (1)

PetrS 0

Is my password brute-force-able if consists of 9 printable characters?

Join at
slido.com
#pv204_2021

- **Place/upvote questions in slido while listening to lecture video**
- **We will together discuss these during every week lecture Q&A (every Monday, 17-18:00)**

Hardware Security Module

HARDWARE SECURITY MODULE

Hardware Security Module - definition

- HSM is trusted hardware element
  - Contains own physical and logical protection
  - May provide increased performance (compared to CPU)
- Attached to or put inside PC/server/network box
- Provides in-device:
  - Secure key generation (and entry)
  - Secure storage (and backup)
  - Secure use (cryptographic algorithms)
- Should never export sensitive data in plaintext
  - Especially keys = Critical Security Parameters (CSP)

You already know one example of HSM – a cryptographic smartcard

Smart cards

- Price: $3-30
- 2-5 RSA/ECC signs/sec
- USB/serial connection
- Mostly disconnected
- No battery
- 3KB RAM, 100KB flash
- Limited algs. support

HSMs

- $100-$10000
- 100-10000 RSA/ECC signs/sec
- UTP/PCI connected
- Always connected
- Own battery (time…)
- MBs-GBs, SSD
- Wide range of algorithms
- Rich API + management
  - Common applications
- Trusted input interface (smartcard reader)

Typical use-cases for HSMs

- Payment industry (PIN and transaction verification)
- TLS accelerator (server’s private key)
- Certification authority (protection of CA private key)
- Key management (distribution, derivation)
- Software signing
- Custom uses (DRM…)

- Vendors - market is now consolidating
  - ~~IBM~~, ~~nCipher~~ , Thales, ~~Safenet,~~ ~~Gemalto,~~ Utimaco…

Many HSM forms possible

- Stand-alone Ethernet boxes (1U/2U)
- PCI cards
- Serial/USB tokens
- SmartCards, TPMs…
- *Note: we will focus on more powerful devices (smart cards already covered)*

https://www.thales-esecurity.com/products-and-services/products-and-services/hardware-security-modules

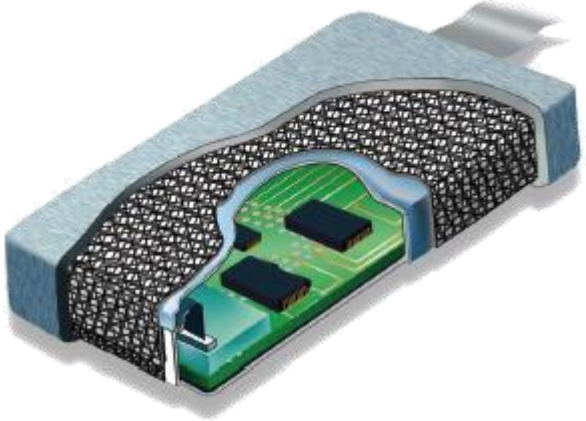
Hardware Security Module - specification

- Common functions
  - Generate functions (generate new key)
  - Load functions (import key, plain/wrapped by other key)
  - Use key functions (various cryptographic algorithms)
  - Export key functions (wrapping)
  - Access control functions (public, login user, login admin)
  - Destroy secrets functions
- Possibility to write custom “plugins”
  - Custom code running inside HSM
  - (usually invalidates certification)

Hardware Security Module - protection

- Protections against physical attacks (tamper)
  - Invasive, semi-invasive and non-invasive attacks
- Protection against logical attacks
  - API-level attacks, Fuzzing…
- Preventive measures
  - Statistical testing of random number generator
  - Self-testing of cryptographic engines (encrypt twice, KAT)
  - Firmware integrity checks
  - Periodic reset of device (e.g., every 24 hour)
  - …

www.techbriefs.com



HSM – tamper security

- Protection epoxy
- Wiring mesh
- Temperature sensors
- Light sensors
- Variations (glitches) in power supply
- Erasure of memory (write 0/random)
  - – After tamper detection to mitigate data remanence
- …

Which one is tamper resistance, evidence, detection and/or reaction?

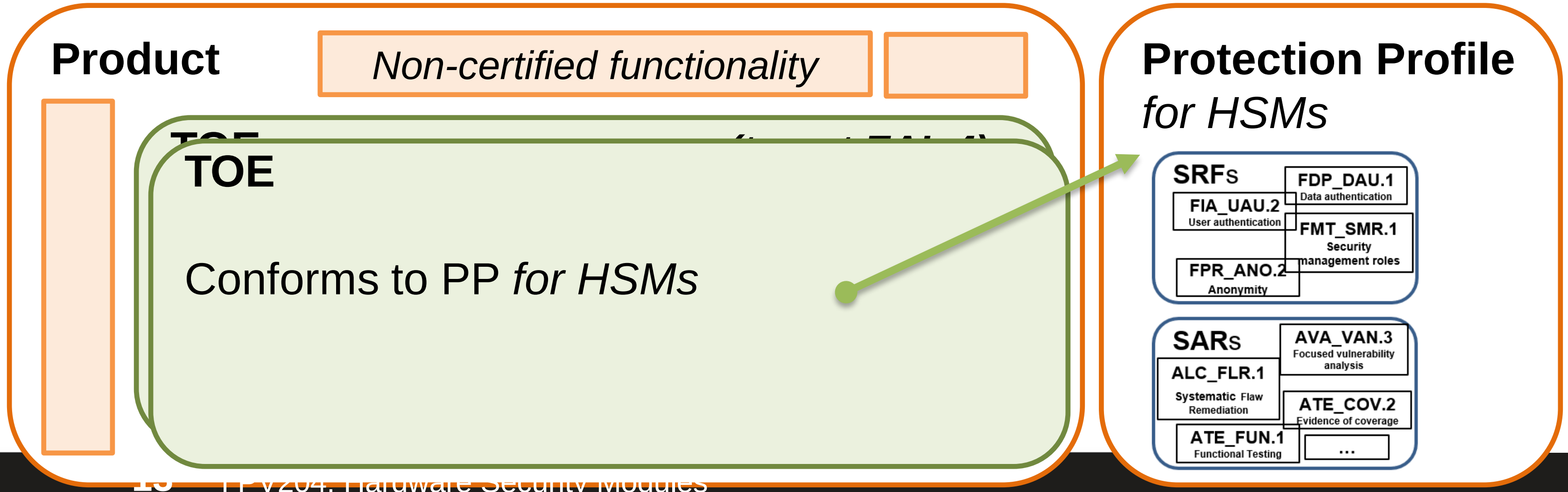
HSM – logical security

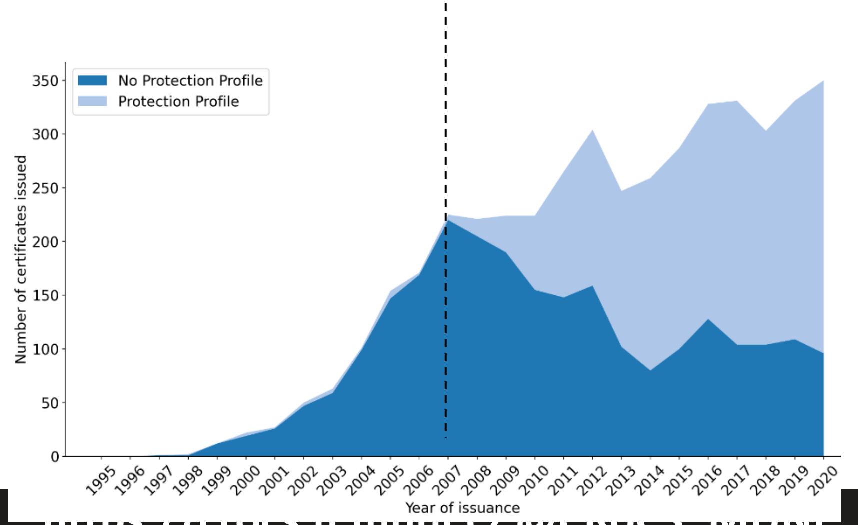
- Access control with limited/delayed tries
  - < 1:1000 000 probability of random guess of password
  - < 1:100 000 probability of unauthorized access in one minute
- Integrity and authentication of firmware update
  - Signed firmware updates
- Logical separation of multiple users (memory)
  - Additional protection logic for separate memory regions
- Audit trails
- …

CERTIFICATIONS

Common Criteria certification primer

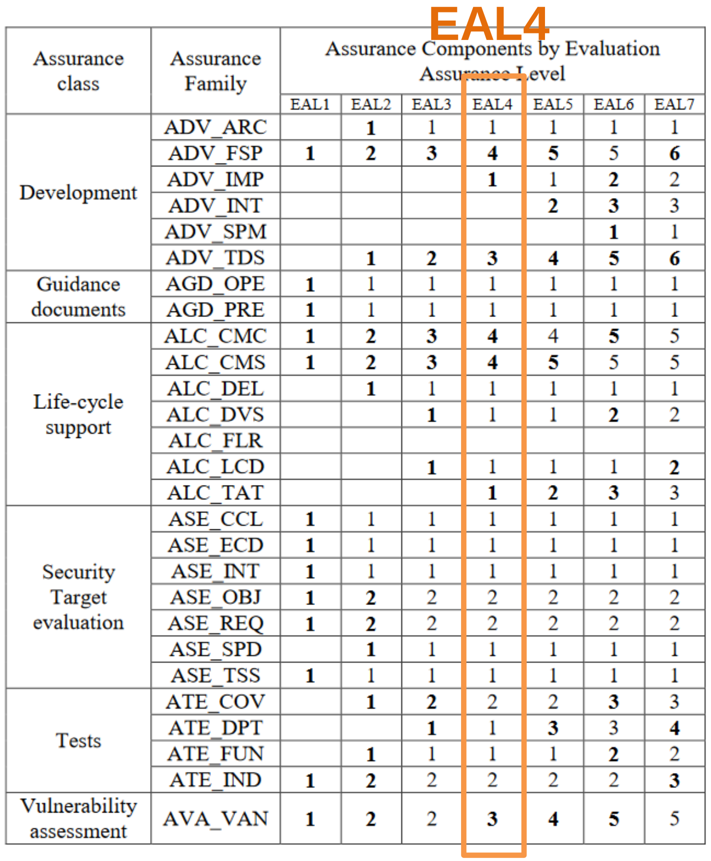
- Product vendor make some claims about Target of Evaluation (ToE, product or parts of it)
  - Security functional components (SFRs) – security functions provided by the ToE (product)
  - Security assurance components (SARs) – measures taken during the ToE lifetime
  - Implementation-dependent statement of security needs written in Security Target (ST) document
- Claims about product (ST) can be constructed from:
  1. Individual SFRs, SARs as mandated by the targeted Evaluation Assurance Level (EAL)
  2. Taken from an approved Protection Profile ("template" for ST, now significantly preferred option)





Common Criteria certification primer

- Evaluation Assurance Level (EAL) corresponds to extend of scrutiny
  - EAL1-7, augmented - particular EAL also mandates minimal SAR levels
  - Certificates mutually recognized up to EAL 2, up to EAL 4 inside EU
    - Common Criteria Recognition Arrangement (CCRA)
- Claims validated by accredited laboratories/evaluation facilities
  - If successful, product certificate is given and published
    - by Certificate Authorizing Members (e.g., French ANSSI, German BSI)
    - validity period typically 3 or 6 years
  - Maintenance Report(s) – smaller changes which doesn't require full recertification, or just continuation
    - submitted by vendor, again validated by lab
  - Labs comply with ISO/IEC 17025, national cert. bodies approved against ISO/IEC 17065



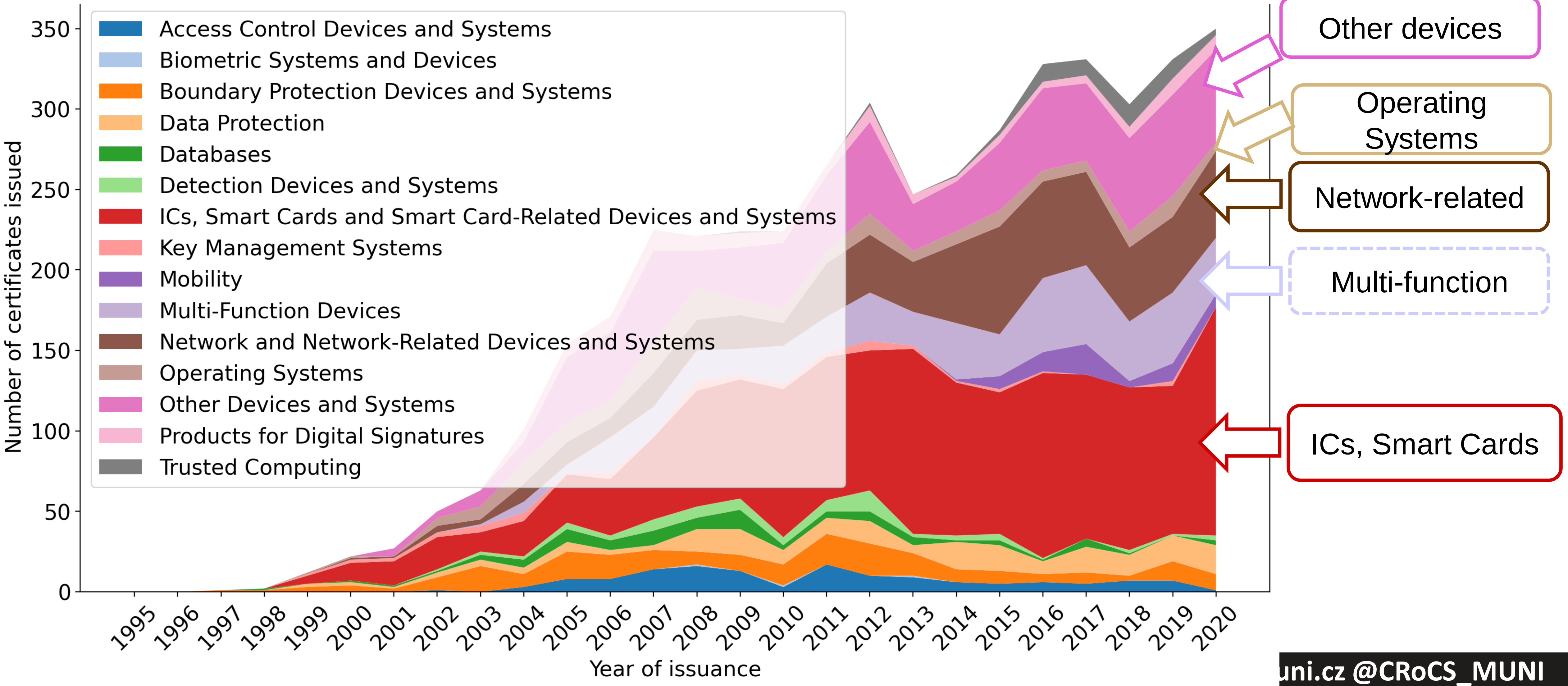
EAL4

Assurance class	Assurance Family	Assurance Components by Evaluation Assurance Level						
		EAL1	EAL2	EAL3	EAL4	EAL5	EAL6	EAL7
Development	ADV_ARC		**1**	1	1	1	1	1
	ADV_FSP	**1**	**2**	**3**	**4**	**5**	5	**6**
	ADV_IMP				**1**	1	**2**	2
	ADV_INT					**2**	**3**	3
	ADV_SPM						**1**	1
	ADV_TDS		**1**	**2**	**3**	**4**	**5**	**6**
Guidance documents	AGD_OPE	**1**	1	1	1	1	1	1
	AGD_PRE	**1**	1	1	1	1	1	1
Life-cycle support	ALC_CMC	**1**	**2**	**3**	**4**	4	**5**	5
	ALC_CMS	**1**	**2**	**3**	**4**	**5**	5	5
	ALC_DEL		**1**	1	1	1	1	1
	ALC_DVS			**1**	1	1	**2**	2
	ALC_FLR							
	ALC_LCD			**1**	1	1	1	**2**
	ALC_TAT				**1**	**2**	**3**	3
Security Target evaluation	ASE_CCL	**1**	1	1	1	1	1	1
	ASE_ECD	**1**	1	1	1	1	1	1
	ASE_INT	**1**	1	1	1	1	1	1
	ASE_OBJ	**1**	**2**	2	2	2	2	2
	ASE_REQ	**1**	**2**	2	2	2	2	2
	ASE_SPD		**1**	1	1	1	1	1
	ASE_TSS	**1**	1	1	1	1	1	1
Tests	ATE_COV		**1**	**2**	2	2	**3**	3
	ATE_DPT			**1**	1	**3**	3	**4**
	ATE_FUN		**1**	1	1	1	**2**	2
	ATE_IND	**1**	**2**	2	2	2	2	**3**
Vulnerability assessment	AVA_VAN	**1**	**2**	2	**3**	**4**	**5**	5

Documents produced and publicly available

- Documents produced and/or publicly available
  - Security Target document – provided by vendor (or on behalf) to Evaluation facility
  - Certification Report – issued by Cert. Auth. Member (e.g., French ANSSI), after checks by accredited Evaluation facility/lab (e.g., Serma Technologies)
  - Maintenance Report(s) – smaller changes which doesn't require full recertification
  - Protection Profiles documents – template for specific functionality, single vendor or collaborative
  - CSV/HTML pages with some additional metadata, summary documents
    - automatically generated by CC portal, Cert. Auth. Members…
- *(Additional confidential documents shared between vendor and lab)*

What: Category of certified devices over the years



Other devices

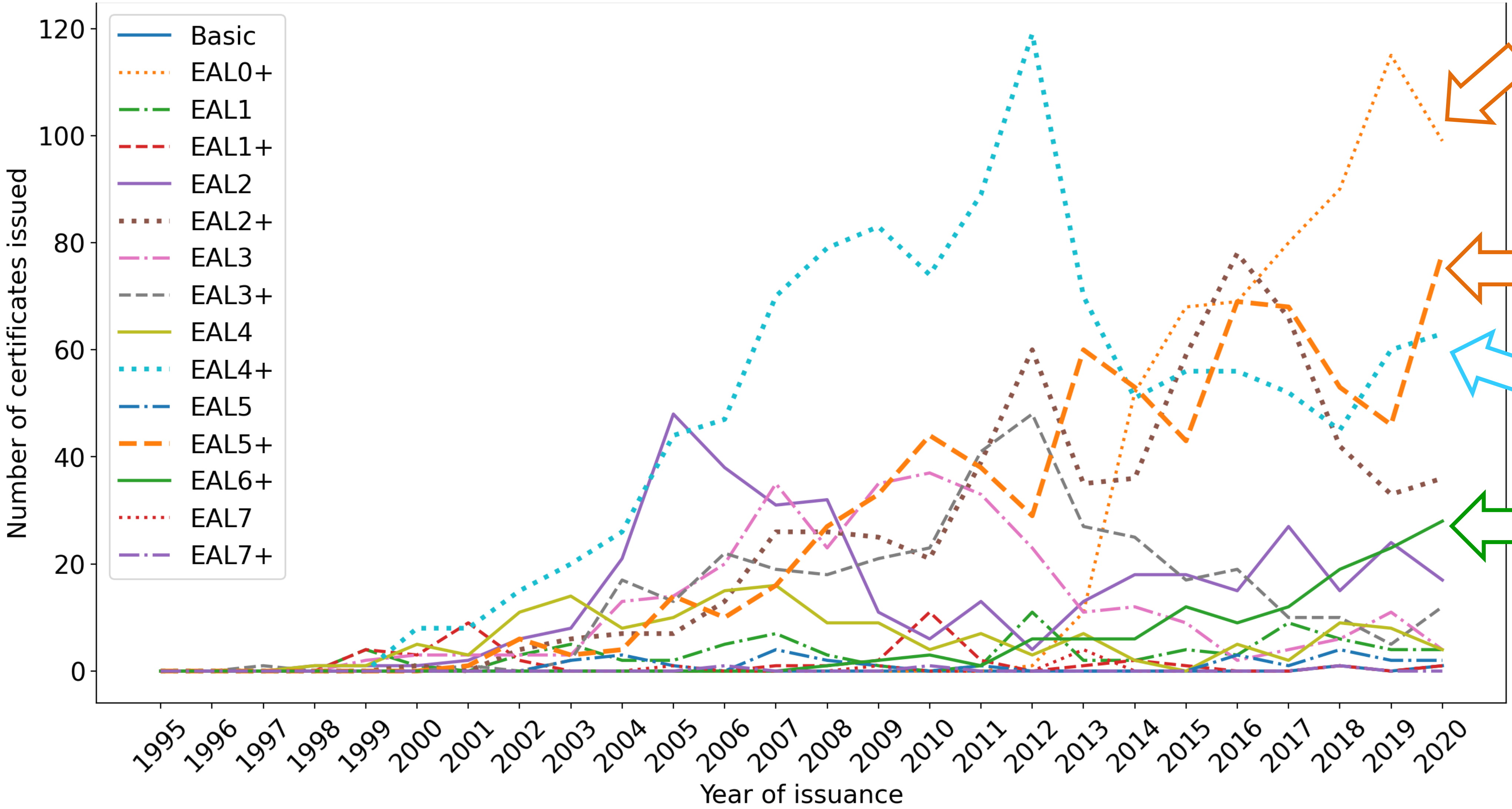
Operating Systems

Network-related

Multi-function

ICs, Smart Cards

Security level frequency per year



As by specific Protection Profile

EAL5+

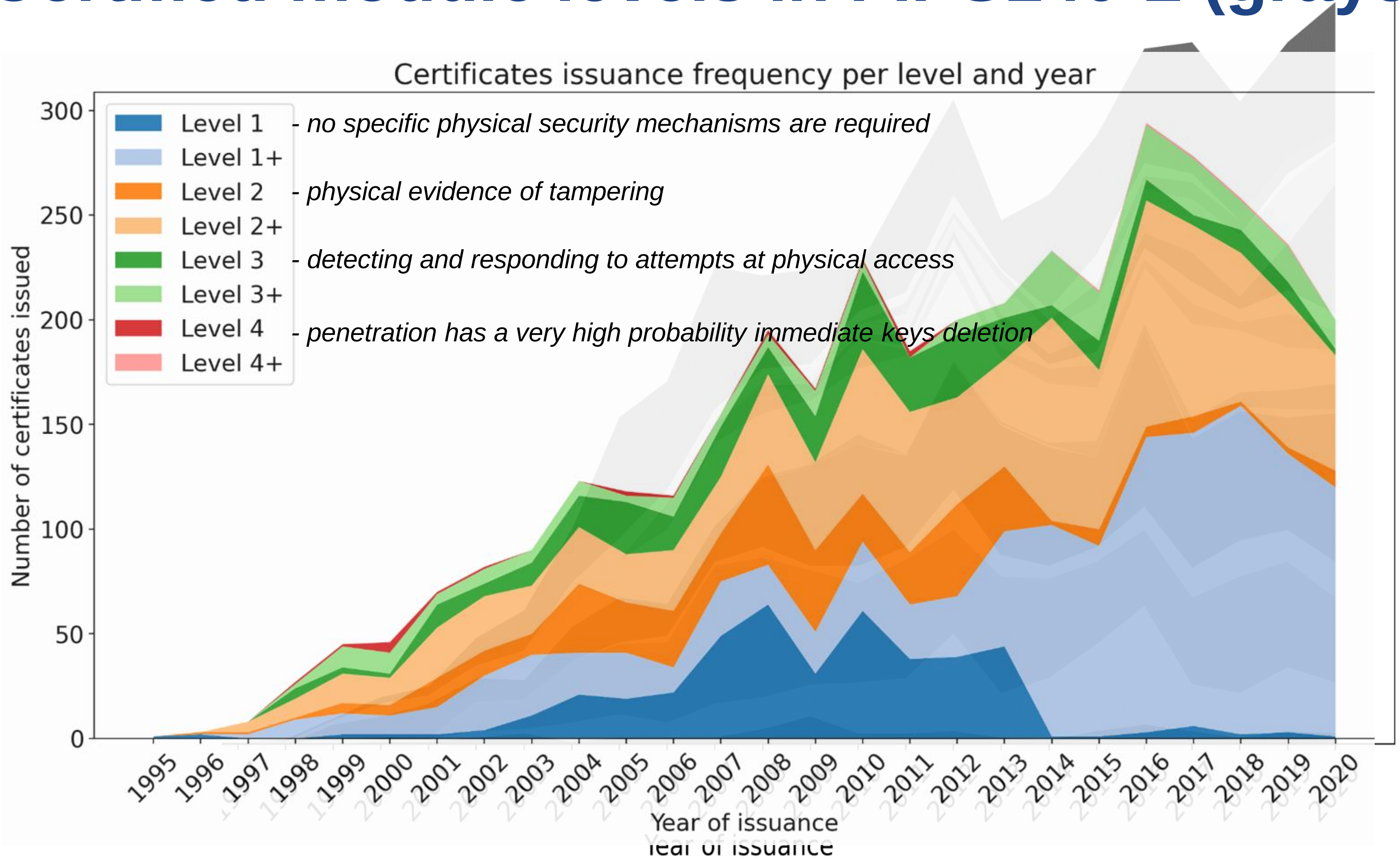
EAL4+

EAL6+

NIST FIPS140-2 certification primer

- Security Requirements for Cryptographic Modules
  - More specific domain than Common Criteria - both hardware and software
- Module – evaluated item with some security/cryptographic functionality
  - Certificate #3820
- Algorithm - implementation of security algorithm by given module
  - List of approved algorithms
    - e.g., AES in GCM mode, RSA key wrapping, SHA2 hash function...
  - Other algorithms possibly available in non-FIPS mode
- Public documents: Security Policy document, certificate web page

Certified module levels in FIPS140-2 (grayed are CC results)



- no specific physical security mechanisms are required

- physical evidence of tampering

- detecting and responding to attempts at physical access

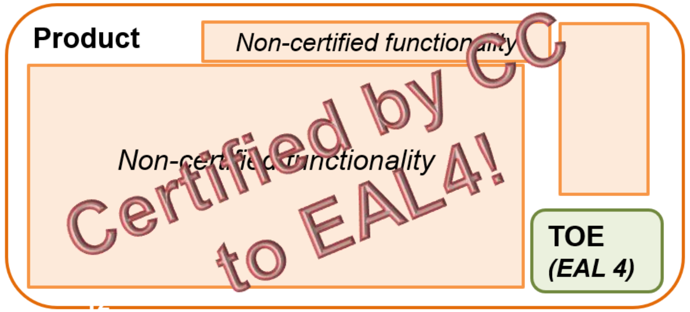
- penetration has a very high probability immediate keys deletion

Certifications: FIPS140-2

- NIST FIPS 140-2
  - Verified under Cryptographic Module Validation Program (CMVP)
  - NIST FIPS 140-2 Level 1+2 – basic levels, tamper evidence (broken shell, epoxy), role-based authentication (user/admin))
  - NIST FIPS 140-2 Level 3 – addition of physical tamper-resistance, identity-based auth, separation of interfaces with different sensitivity
  - NIST FIPS 140-2 Level 4 + additional physical security requirements, environmental attacks (very few devices certified)
- NIST FIPS 140-3
  - for a long time only draft, then abandoned, then in March 2019 somewhat surprisingly approved
    - Additional focus on software security and non-invasive attacks
    - Testing from September 2020, supersedes FIPS140-2 but in parallel till 2026
- List of validated devices https://csrc.nist.gov/projects/cryptographic-module-validation-program

Some problems…

- CC certification is costly and takes long time (>$100k, >>3 months)
  - Works well for static, long-time usable products (hardware, smartcards…)
  - CC generally not suitable for quickly changing products (software in cloud with daily updates…)
- Hard to interpret actual security by end-users
  - Evaluation only with respect to ToE (crucial parts can be put out-of-scope by vendor)
    - Marketing claims like “Common Criteria certified” (important is ToE details, achieved EAL, PP conformance, laboratory used…) or “Common Criteria ready”
  - Product is changing (sw/hw updates) – what is actually certified?
- How well was product scrutinized by testing laboratory?
  - Lack of public details, tools used, configurations and results…
  - Exact procedures under NDA and IP of labs/vendors



Cost of certification

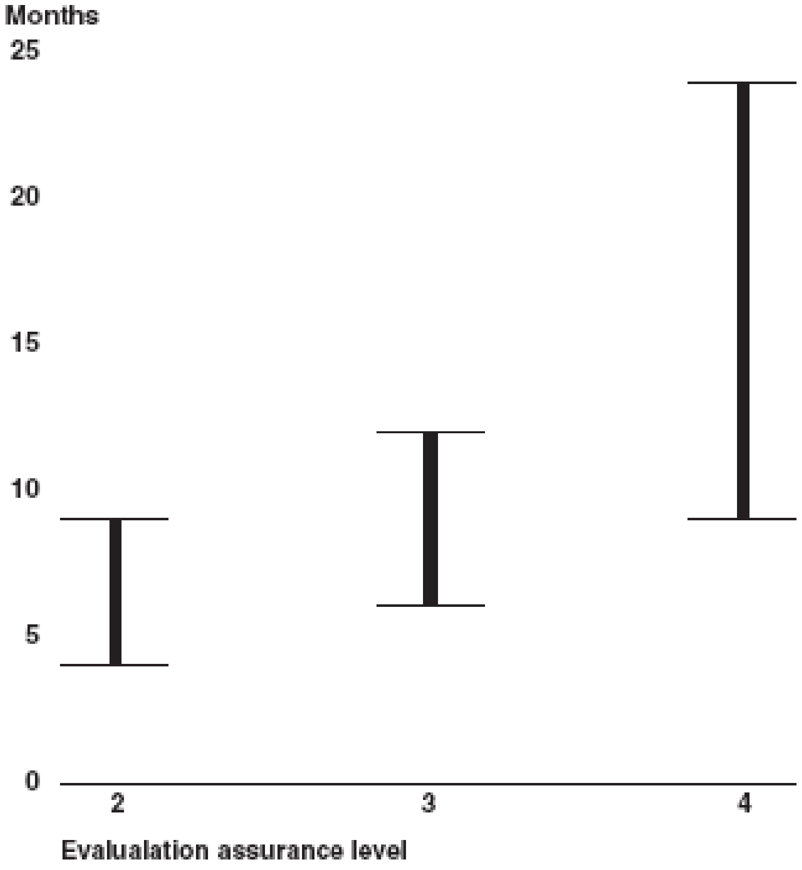
- Certification is usually done by commercial “independent” laboratories
  - Laboratories are certified by governing body
  - Quality and price differ
  - Usually payed for by device manufacturer

1. Certification pre-study
   - Verify if product is ready for certification
2. Full certification
   - Checklist if all required procedures were followed

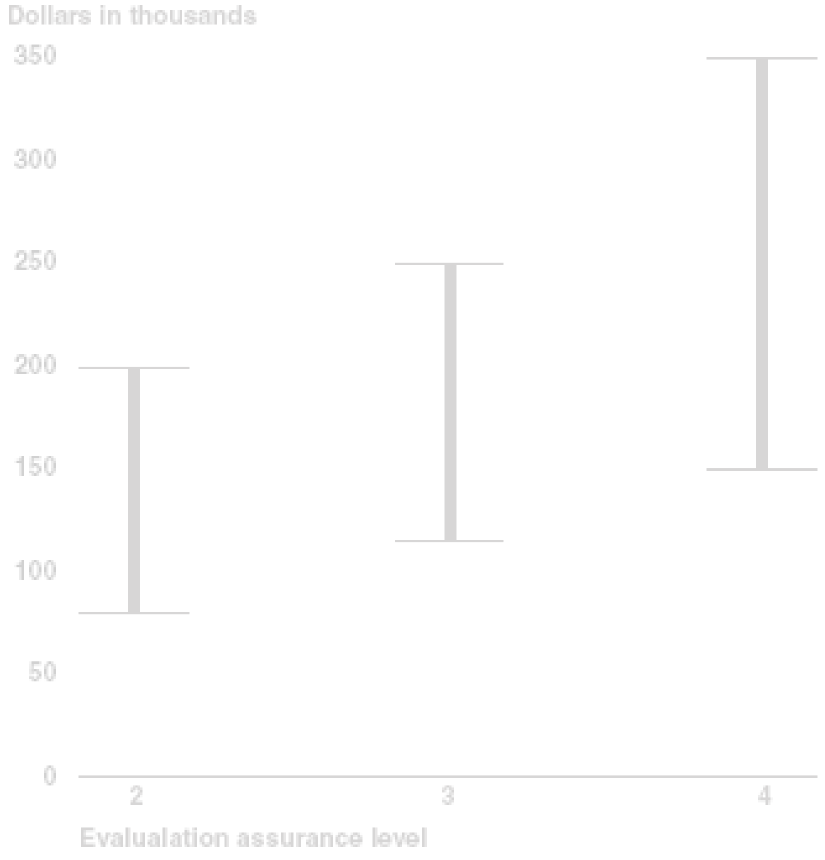
Cost of CC EAL (US GAO, 2006)

Motivation to keep already certified version longer in production

- Still true today (2021) (years to certify, $250k+ cost)



Source: GAO analysis of data provided by laboratories.



Source: GAO analysis of data provided by laboratories.

Be aware what is actually certified

- Certified != secure
  - Satisfies defined criteria, producer claims were verified to be valid
  - Infineon’s RSA prime generation algorithm (BSI, CVE-2017-15361)
- Typically, bundle of hardware and software is certified
  - Concrete underlying hardware
  - Concrete version of firmware, OS and pre-loaded application
- Certification usually invalidated when:
  - New hardware revision used (less common)
  - New version of firmware, OS, application (common)
  - Any customization, e.g., user firmware module (very common)
- Pragmatic result
  - “I’m using product that was certified at some point in time”

How is certified product used?

- Trade-off between security functionality and required data centre operations
- Certification FIPS 140-2
  - users usually turn FIPS mode off (want use additional functionality)
- “Almost” FIPS 140-2 mode
  - Everything FIPS except what user added (custom module)

HSM PERFORMANCE

HSM – performance I.

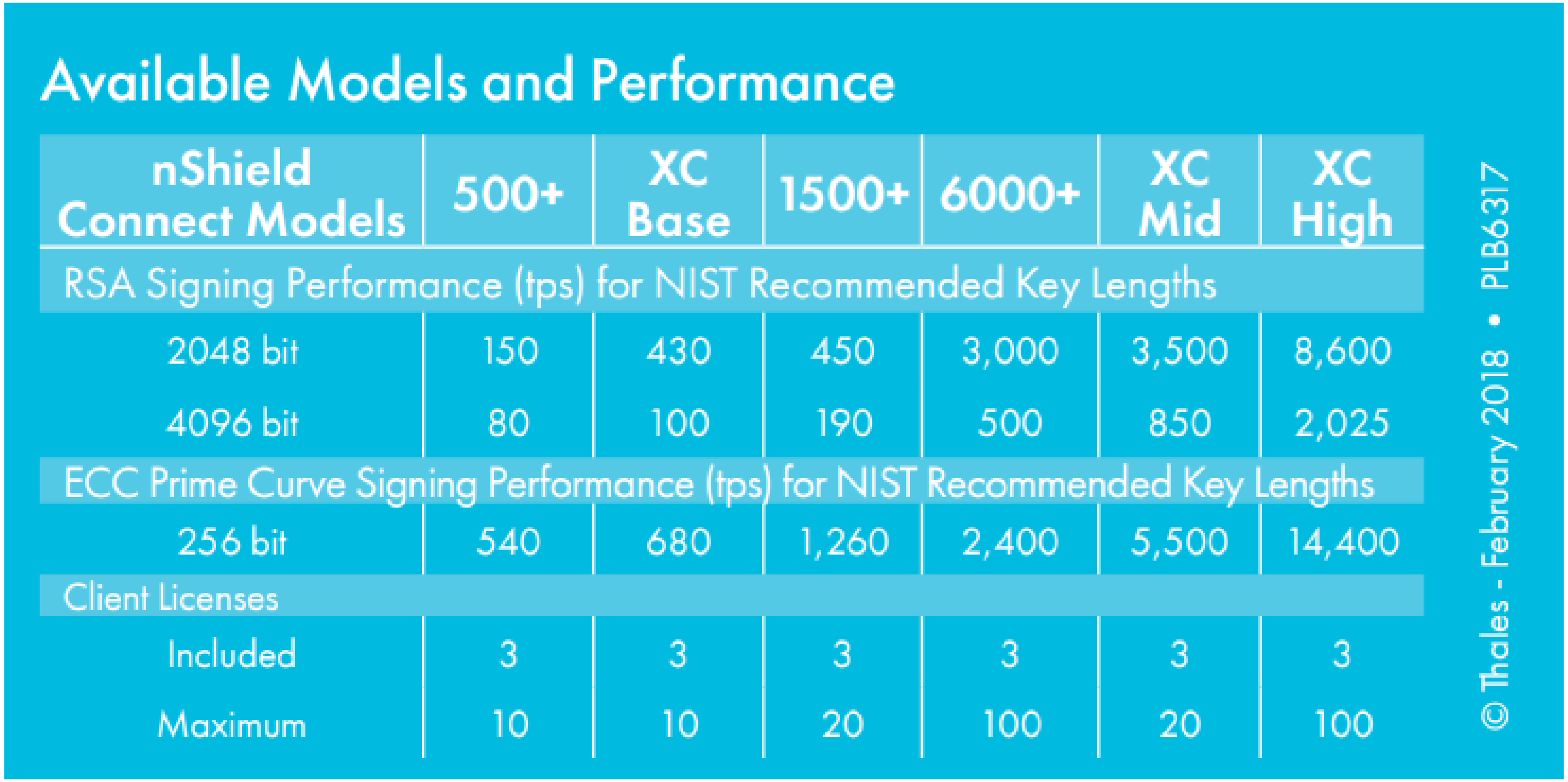
- Limited independent public information available
  - Claim: “up to 9000 RSA-1024b operations / second”
- But…
  - Real operations are not just raw crypto (formatting of messages…)
  - Longer key length may be needed (RSA-2048b and longer)
  - Internal vs. external speed (data in/out excluded)
  - Measurements in “optimal” situations (single pre-prepared key, large data blocks…)
  - …

HSM – performance II.

- Relatively difficult to obtain fair comparison
- F. Demaertelaere (2010)
  - https://handouts.secappdev.org/handouts/2010/Filip%20Demaertelaere/HSM.pdf
- RSA 1024 bit private key operation: 100 – 7000 ops/sec
- ECC 160 bit ECDSA signatures: 250 – 2500 ops/sec
- 3DES: 2 - 8 Mbytes/sec
- AES: 6 - 40 Mbytes/sec (256 bit key)

- No significant breakthrough in technology since 2010
- Higher throughput achieved by multiple HSMs

Recent update (Feb 2018)



Available Models and Performance

nShield Connect Models	500+	XC Base	1500+	6000+	XC Mid	XC High
RSA Signing Performance (tps) for NIST Recommended Key Lengths						
2048 bit	150	430	450	3,000	3,500	8,600
4096 bit	80	100	190	500	850	2,025
ECC Prime Curve Signing Performance (tps) for NIST Recommended Key Lengths						
256 bit	540	680	1,260	2,400	5,500	14,400
Client Licenses						
Included	3	3	3	3	3	3
Maximum	10	10	20	100	20	100

© Thales - February 2018 • PLB6317

http://go.thalesesecurity.com/rs/480-LWA-970/images/ThalesEsecurity_nShield_Connect_ds.pdf

HSM - load balancing, failover

- HSMs often used in business critical scenarios
  - Authorization of payment transaction
  - TLS accelerator for internet banking
  - …
- Redundancy and load-balancing required
- Single HSM is not enough
  - At least two in production for failover
  - At least one or two for development and test

Hardware Security Module

STEPS OF CRYPTO OPERATION

Steps of cryptographic operation

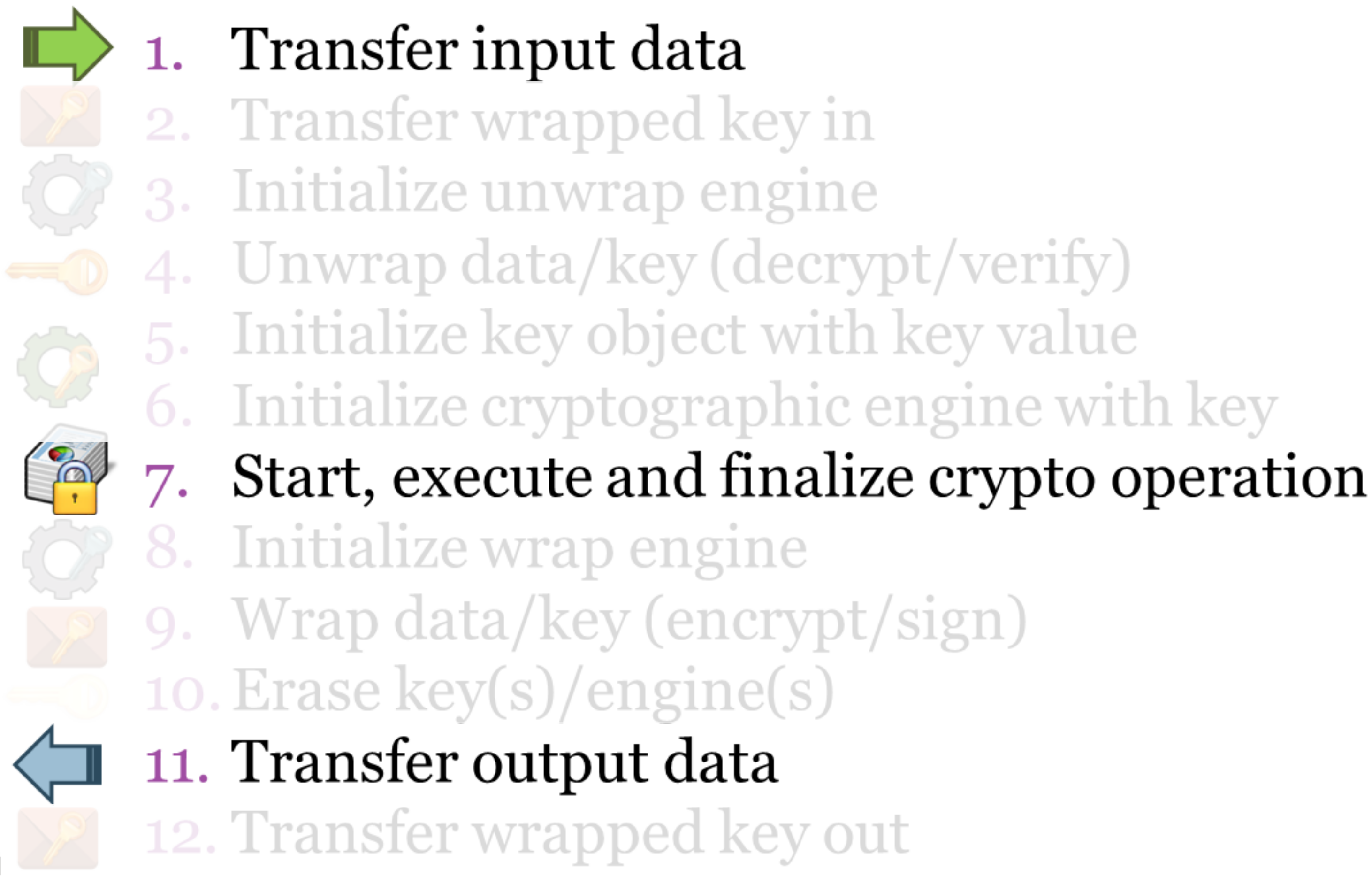
1. Transfer input data
2. Transfer wrapped key in
3. Initialize unwrap engine
4. Unwrap data/key (decrypt/verify)
5. Initialize key object with key value
6. Initialize cryptographic engine with key
7. Start, execute and finalize crypto operation
8. Initialize wrap engine
9. Wrap data/key (encrypt/sign)
10. Erase key(s)/engine(s)
11. Transfer output data
12. Transfer wrapped key out

Modes for sharing of hardware security module

- S1: One user, few keys
- S2: One user, many keys
- S3: Few users, few keys
- S4: Few users, many keys
- S5: Many users, many keys

S1: One user, few keys

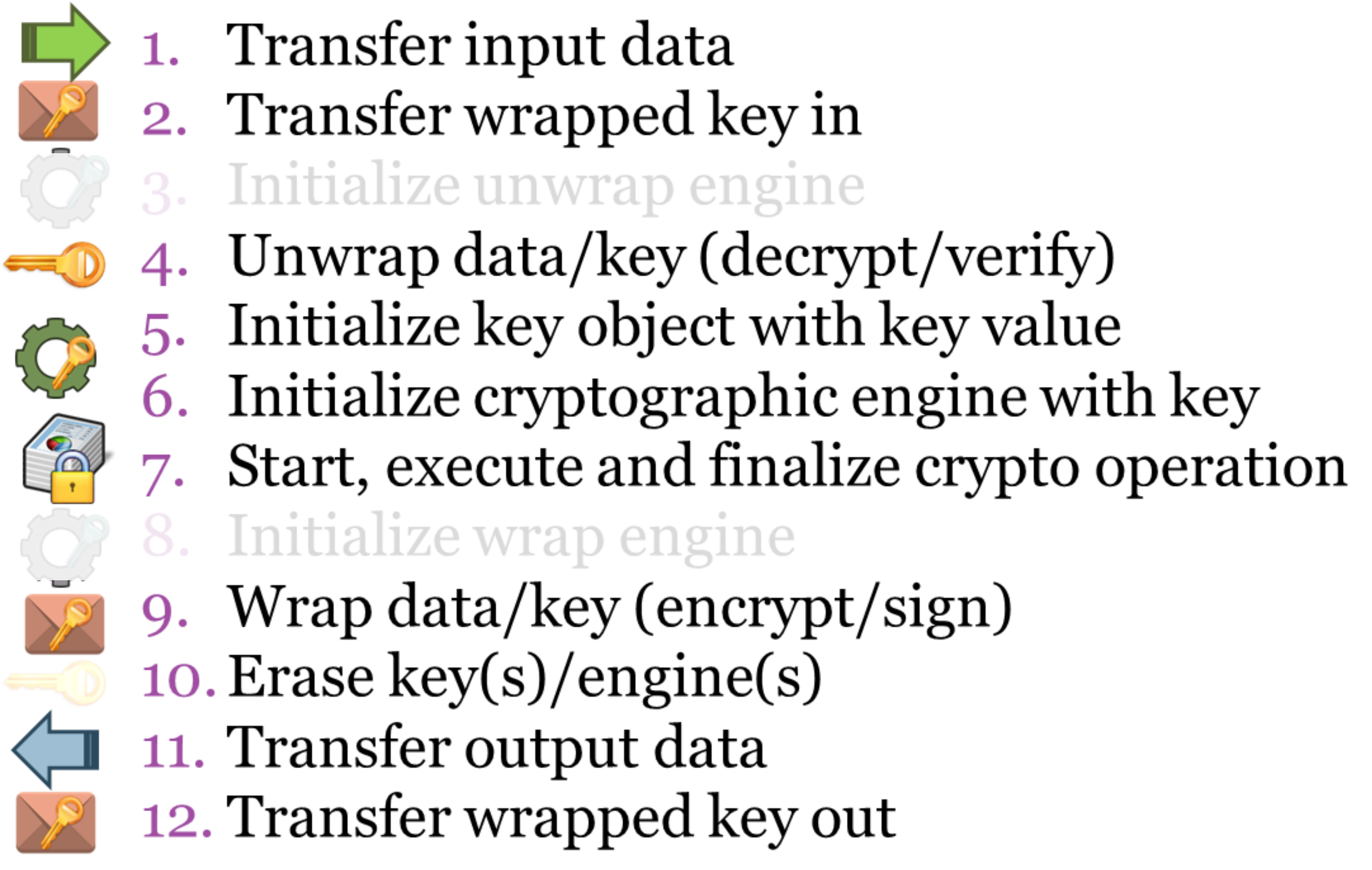
- No sharing, all engines fully prepared



1. Transfer input data
2. Transfer wrapped key in
3. Initialize unwrap engine
4. Unwrap data/key (decrypt/verify)
5. Initialize key object with key value
6. Initialize cryptographic engine with key
7. Start, execute and finalize crypto operation
8. Initialize wrap engine
9. Wrap data/key (encrypt/sign)
10. Erase key(s)/engine(s)
11. Transfer output data
12. Transfer wrapped key out

S2: One user, many keys

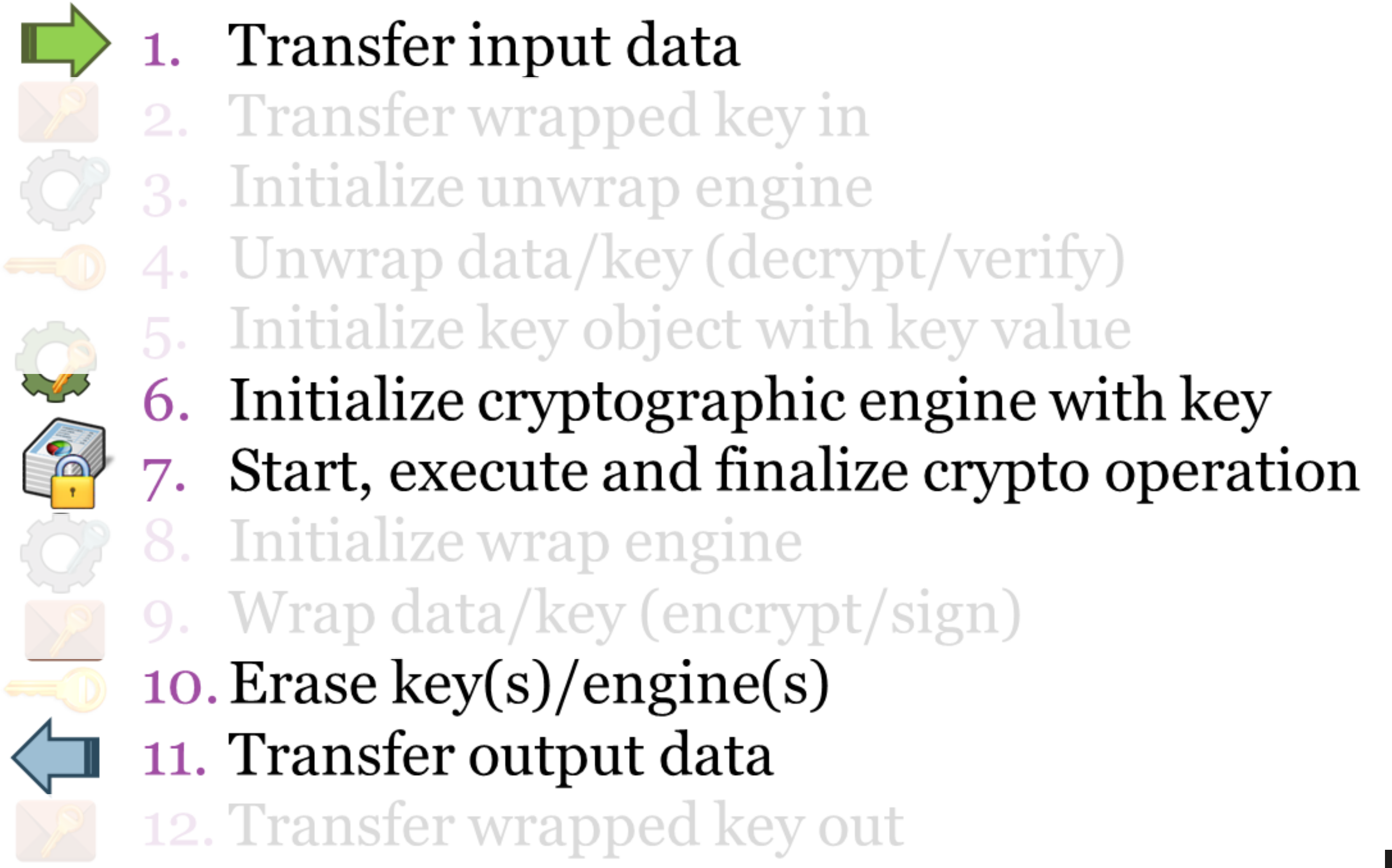
- No sharing, frequent crypto context change



1. Transfer input data
2. Transfer wrapped key in
3. Initialize unwrap engine
4. Unwrap data/key (decrypt/verify)
5. Initialize key object with key value
6. Initialize cryptographic engine with key
7. Start, execute and finalize crypto operation
8. Initialize wrap engine
9. Wrap data/key (encrypt/sign)
10. Erase key(s)/engine(s)
11. Transfer output data
12. Transfer wrapped key out

S3: Few users, few keys

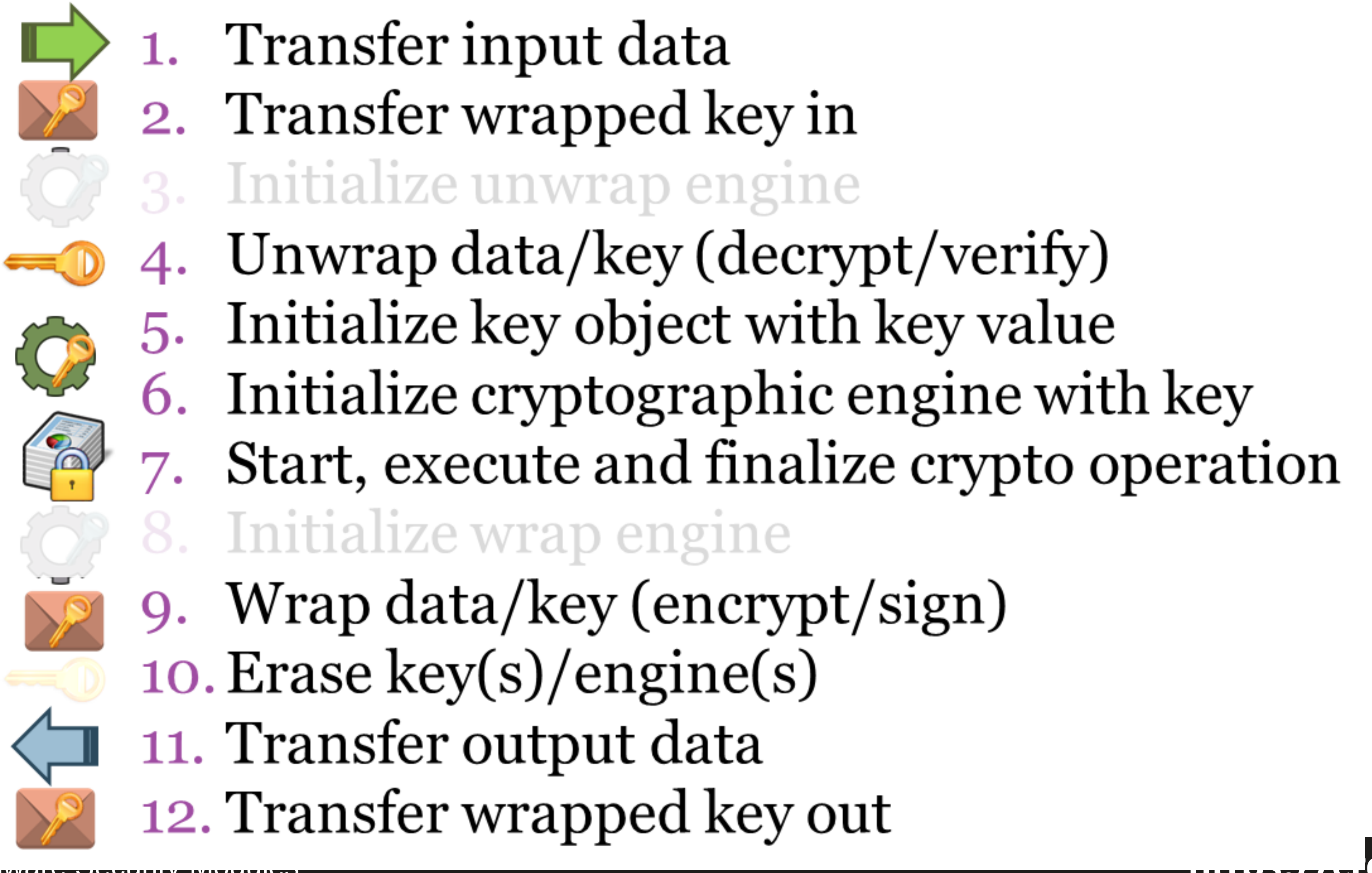
- Device is shared → isolation of users



1. Transfer input data
2. Transfer wrapped key in
3. Initialize unwrap engine
4. Unwrap data/key (decrypt/verify)
5. Initialize key object with key value
6. Initialize cryptographic engine with key
7. Start, execute and finalize crypto operation
8. Initialize wrap engine
9. Wrap data/key (encrypt/sign)
10. Erase key(s)/engine(s)
11. Transfer output data
12. Transfer wrapped key out

S4: Few users, many keys

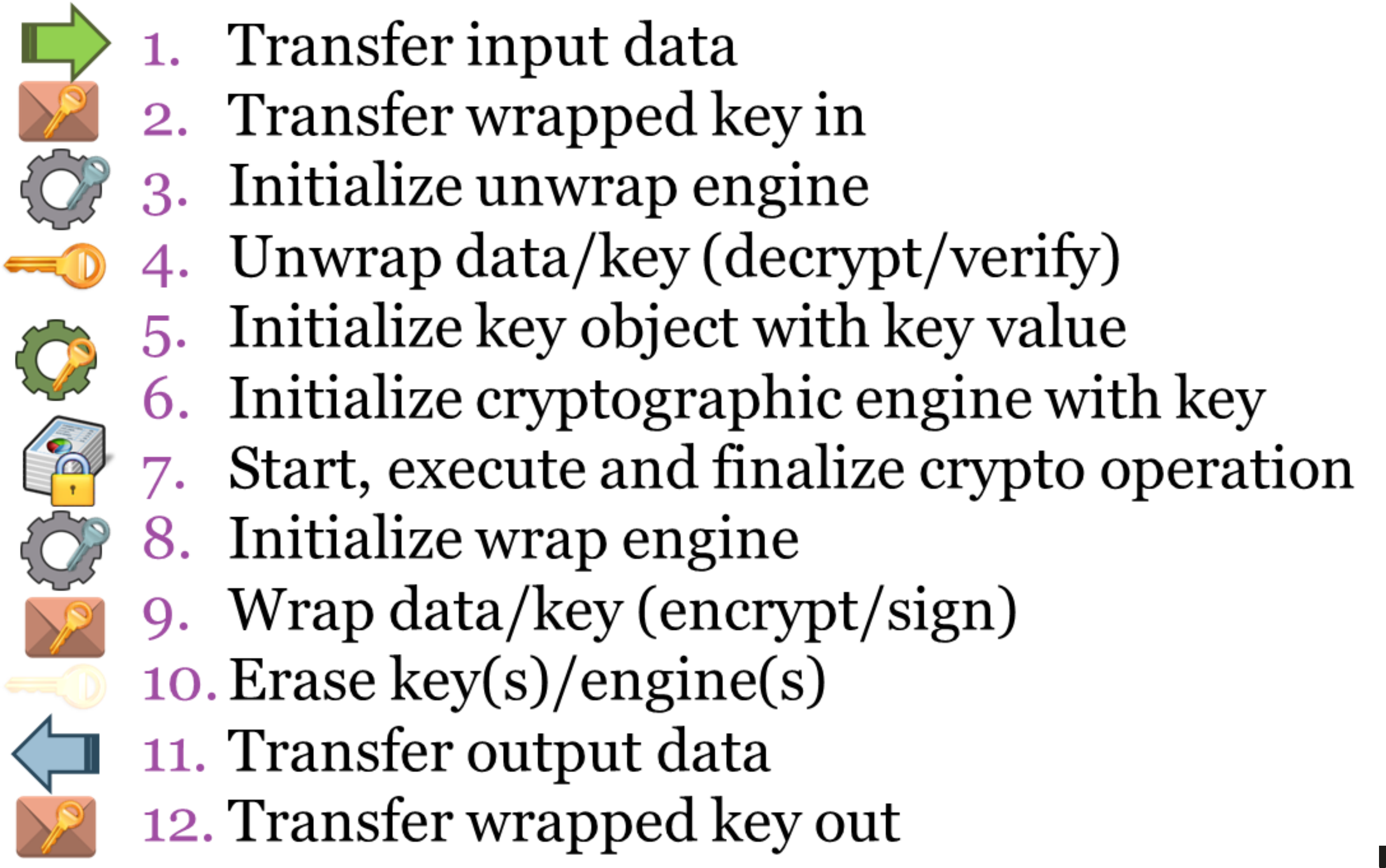
- Limited sharing, frequent crypto context change



1. Transfer input data
2. Transfer wrapped key in
3. Initialize unwrap engine
4. Unwrap data/key (decrypt/verify)
5. Initialize key object with key value
6. Initialize cryptographic engine with key
7. Start, execute and finalize crypto operation
8. Initialize wrap engine
9. Wrap data/key (encrypt/sign)
10. Erase key(s)/engine(s)
11. Transfer output data
12. Transfer wrapped key out

S5: Many users, many keys

- High sharing, frequent crypto context change



1. Transfer input data
2. Transfer wrapped key in
3. Initialize unwrap engine
4. Unwrap data/key (decrypt/verify)
5. Initialize key object with key value
6. Initialize cryptographic engine with key
7. Start, execute and finalize crypto operation
8. Initialize wrap engine
9. Wrap data/key (encrypt/sign)
10. Erase key(s)/engine(s)
11. Transfer output data
12. Transfer wrapped key out

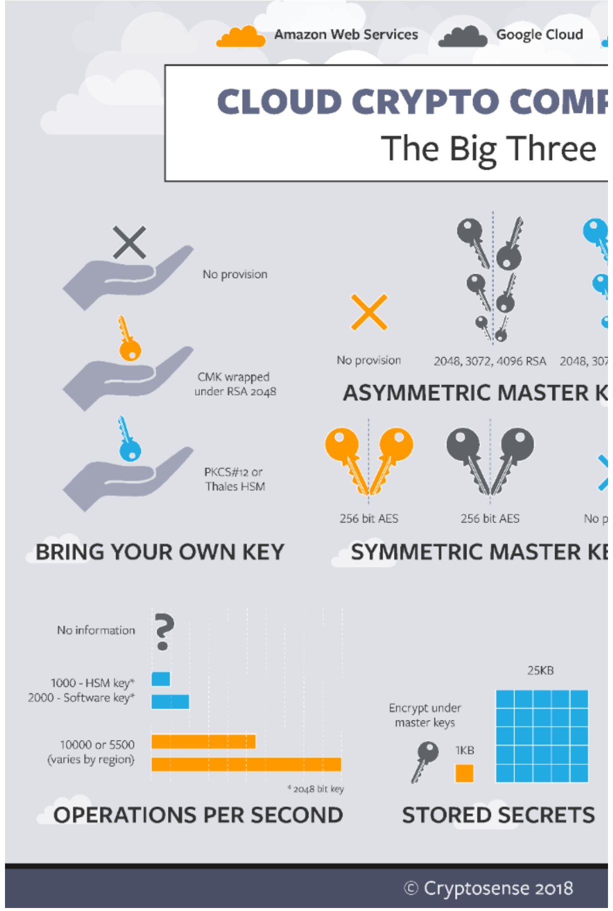
Application Programming Interfaces (API)

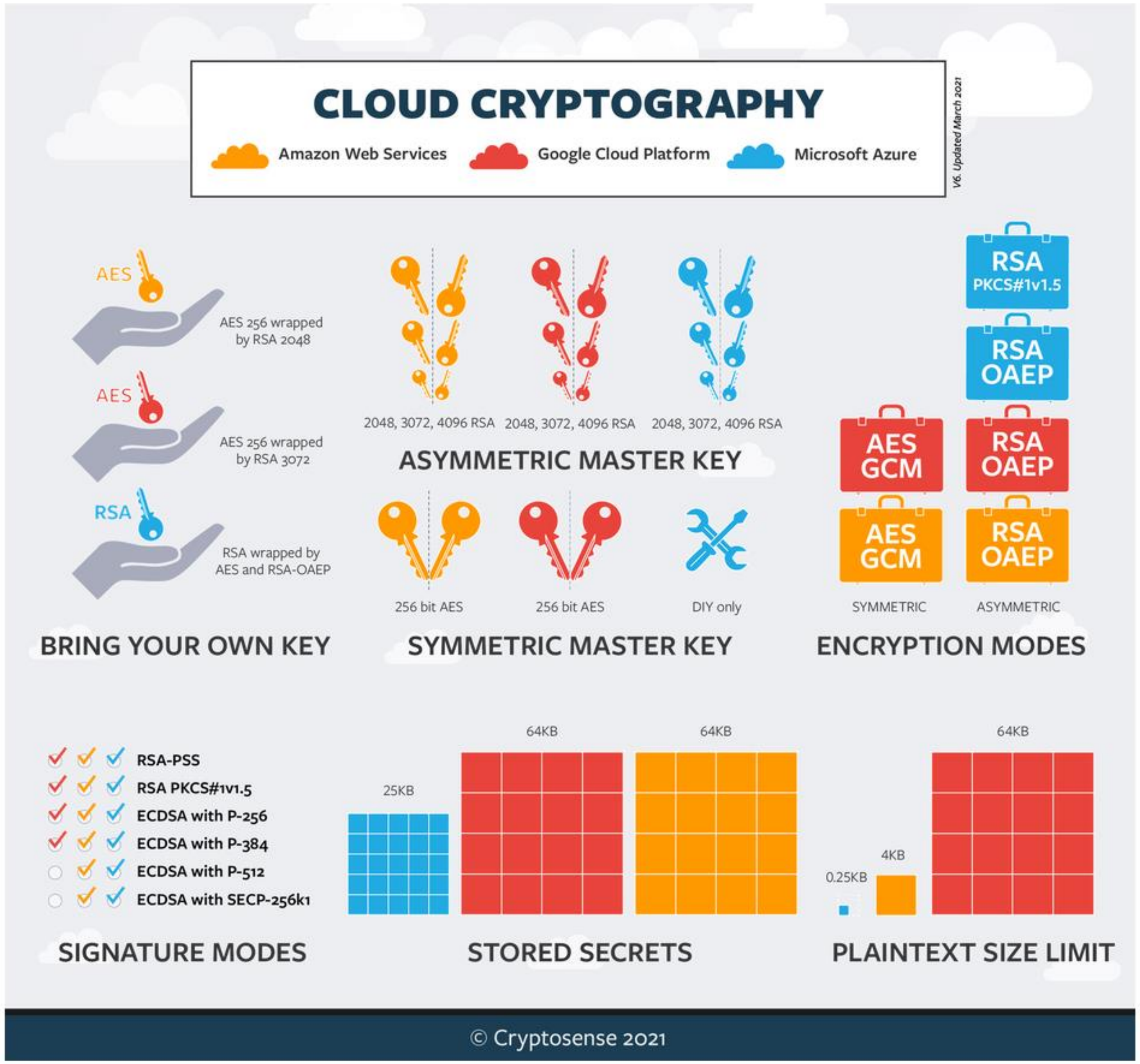
1. Proprietary API (legacy or custom functions)
2. Standardized API - but proprietary library required (PKCS#11)
3. Cryptographic service providers – plugin into standardized API (CNG, CSP…)
4. Standardized API - no proprietary component (PIV, EMV CAP…)
5. Proprietary (service-specific), but public API (MS KeyVault, AWS..)

HSM IN CLOUD

Security topics in cloud environment

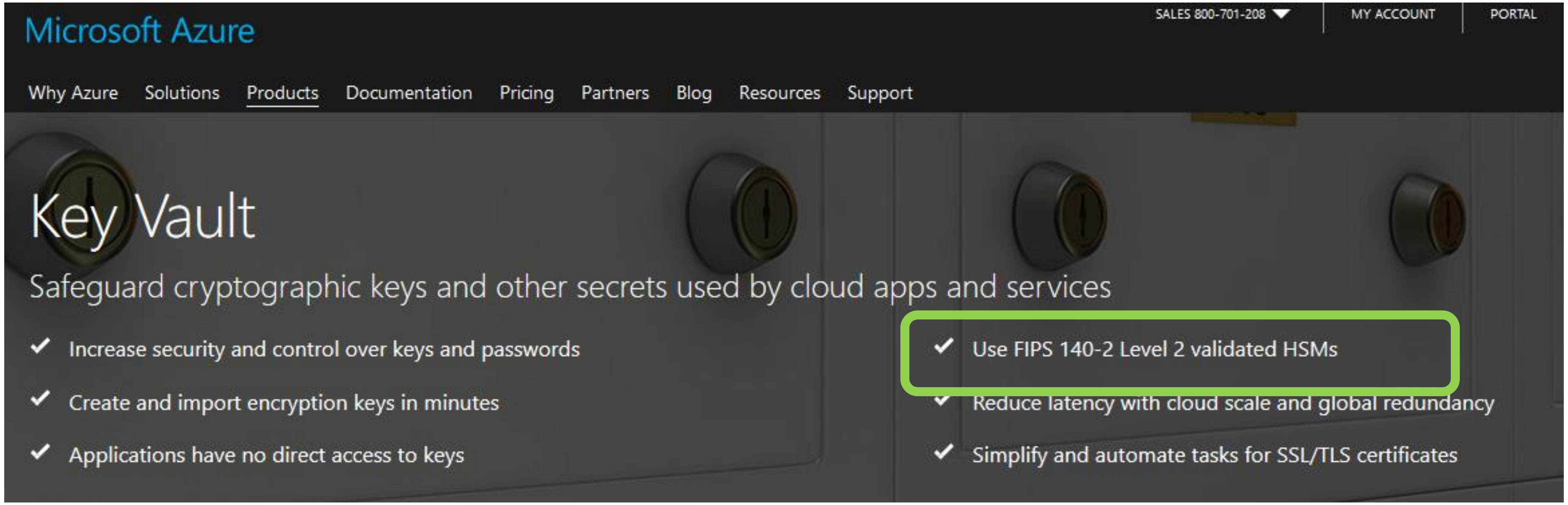
1. Move of legacy applications into cloud
   - Previously used locally connected HSMs
2. Protection of messages exchanged between multiple cloud-based applications
   - Key exchange of used key without pre-distribution?
3. Volume encryption in cloud
   - Encrypted block mounted after application request (e.g., Amazon’s Elastic Block Storage)
4. Encrypted databases
   - Block encryption of database storage, encryption of rows/cells
5. Cryptography as a Service
   - Not only key management, also other cryptographic functionality





https://cryptosense.com/cloud-cryptography-comparison/

Use case: Microsoft Azure KeyVault



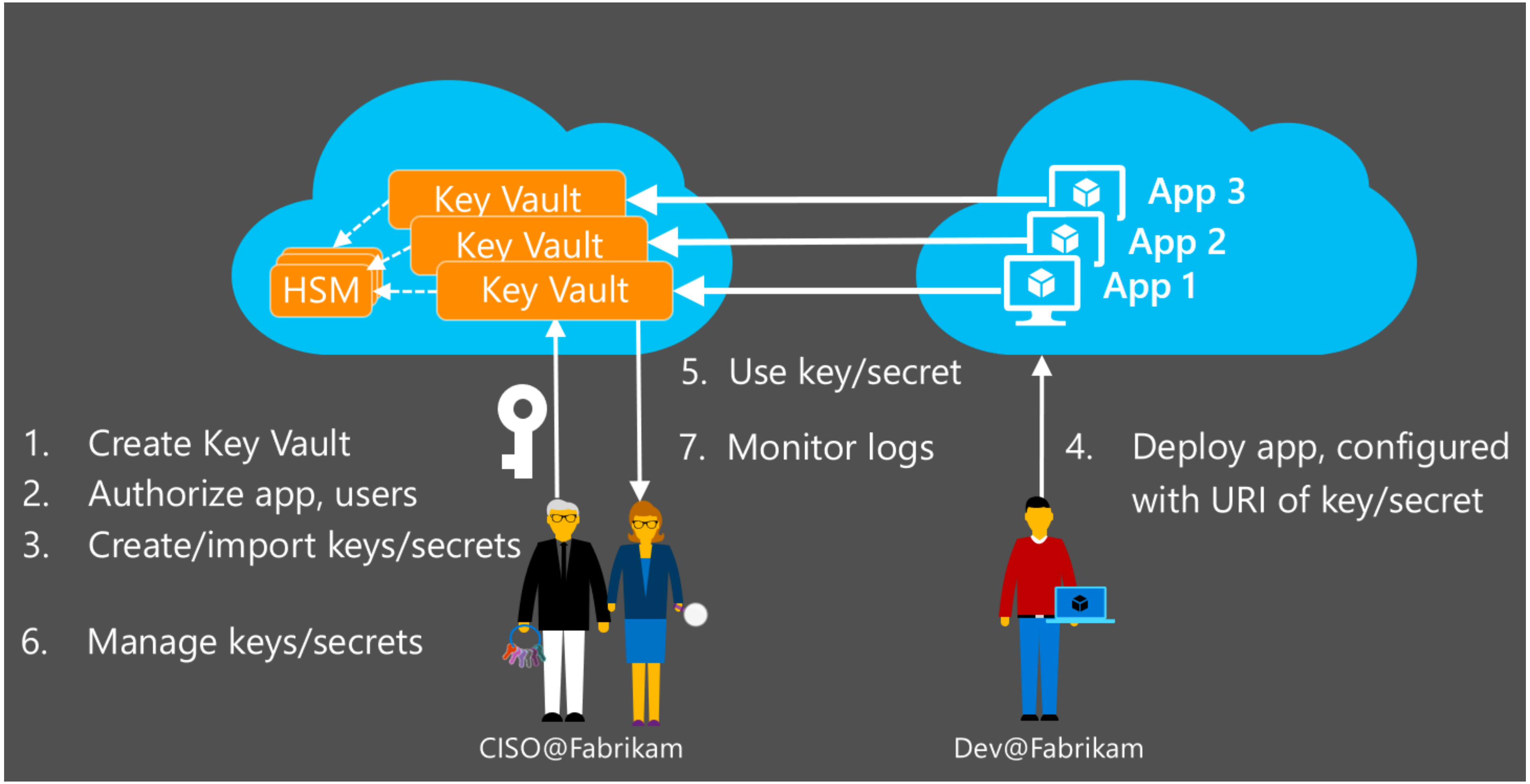
Key Vault

Safeguard cryptographic keys and other secrets used by cloud apps and services

- ✓ Increase security and control over keys and passwords
- ✓ Create and import encryption keys in minutes
- ✓ Applications have no direct access to keys
- ✓ Use FIPS 140-2 Level 2 validated HSMs
- ✓ Reduce latency with cloud scale and global redundancy
- ✓ Simplify and automate tasks for SSL/TLS certificates

- REST API to generate keys, export pub, use keys…
  - – https://docs.microsoft.com/en-us/rest/api/keyvault/
- Language bindings (language specific wrappers)
  - – JS, PowerShell, C#…

Microsoft Azure KeyVault



1. Create Key Vault
2. Authorize app, users
3. Create/import keys/secrets
4. Deploy app, configured with URI of key/secret
5. Use key/secret
6. Manage keys/secrets
7. Monitor logs

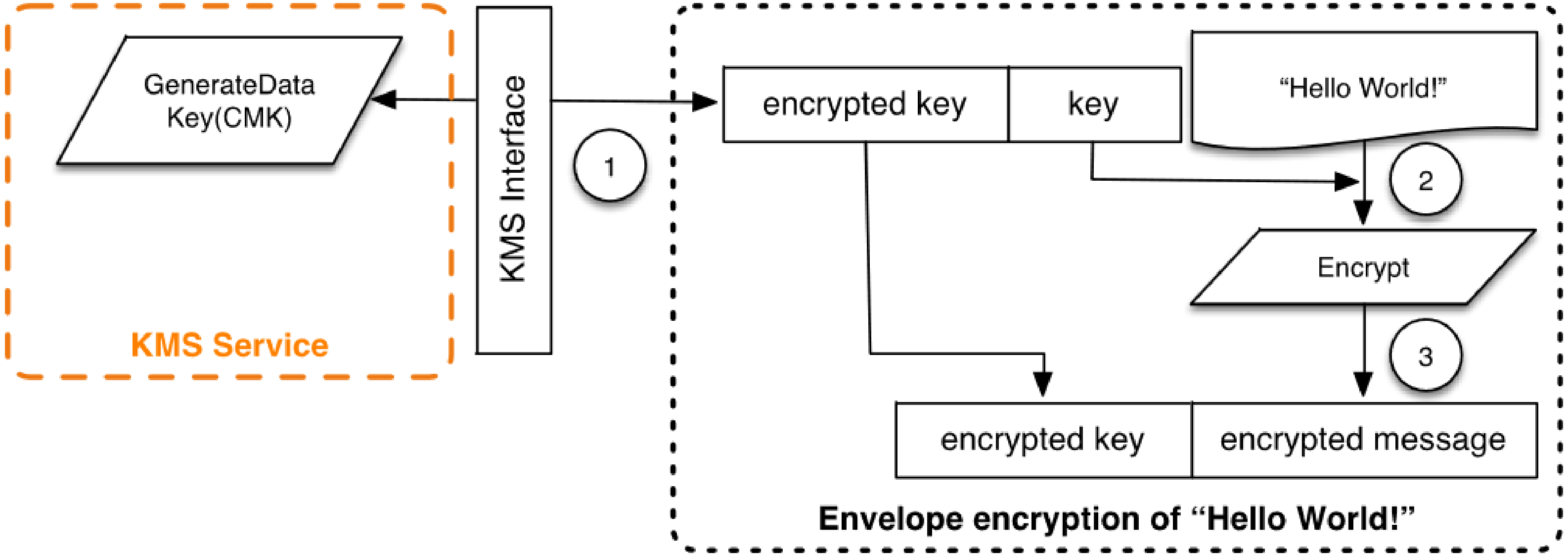
https://channel9.msdn.com/Events/Ignite/2015/BRK2706

Use case: AWS Key Management Service

- AWS Key Management Service Cryptographic Details (2015)
  - https://docs.aws.amazon.com/kms/latest/cryptographic-details/kms-crypto-details.pdf
  - Centralized key management
  - Used by cloud-based applications
  - Used by any client application
  - Replication of wrapping keys into HSMs in different datacenters

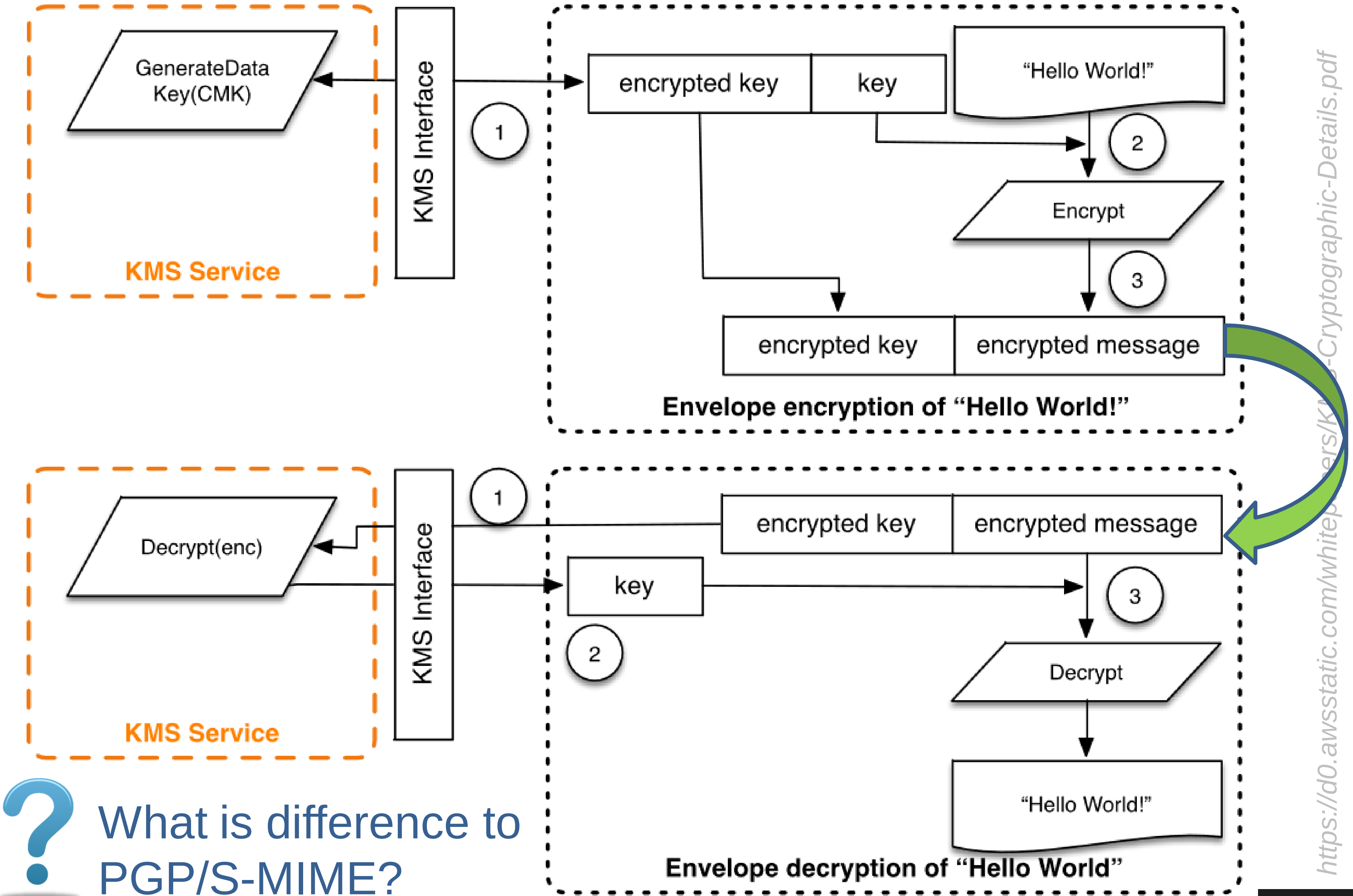
Usage scenario: envelope encryption

- Protected message exchange between multiple (cloud-based) application
  1. Random key generated in one application
  2. Key protected (wrap) using trusted element (HSM)
  3. Wrapped key appended to message
  4. Key unwrapped in second application (via HSM)



Envelope encryption of "Hello World!"

https://d0.awsstatic.com/whitepapers/KMS-Cryptographic-Details.pdf



https://d0.awsstatic.com/whitepapers/KMS-Cryptographic-Details.pdf

What is difference to PGP/S-MIME?

Who is trusted?

- KMS Service to wrap envelope keys properly
- KMS Service not to leak wrapping key
- Cloud operator not to read unwrapped keys from memory

Use case: Amazon AWS CloudHSM

- Amazon’s AWS CloudHSM
  - Based on SafeNet’s Luna HSM
  - Only few users can share one HSM (probably no sharing)
  - => High initial cost (~$5000 + $1.88 per hour)
- Note: significantly different service from AWS KMS
  - “Whole” HSM is available to single user/application, not only key (un)wrapping functionality
  - Suitable for legacy apps, compliancy requirements

Use case: Amazon AWS CloudHSM

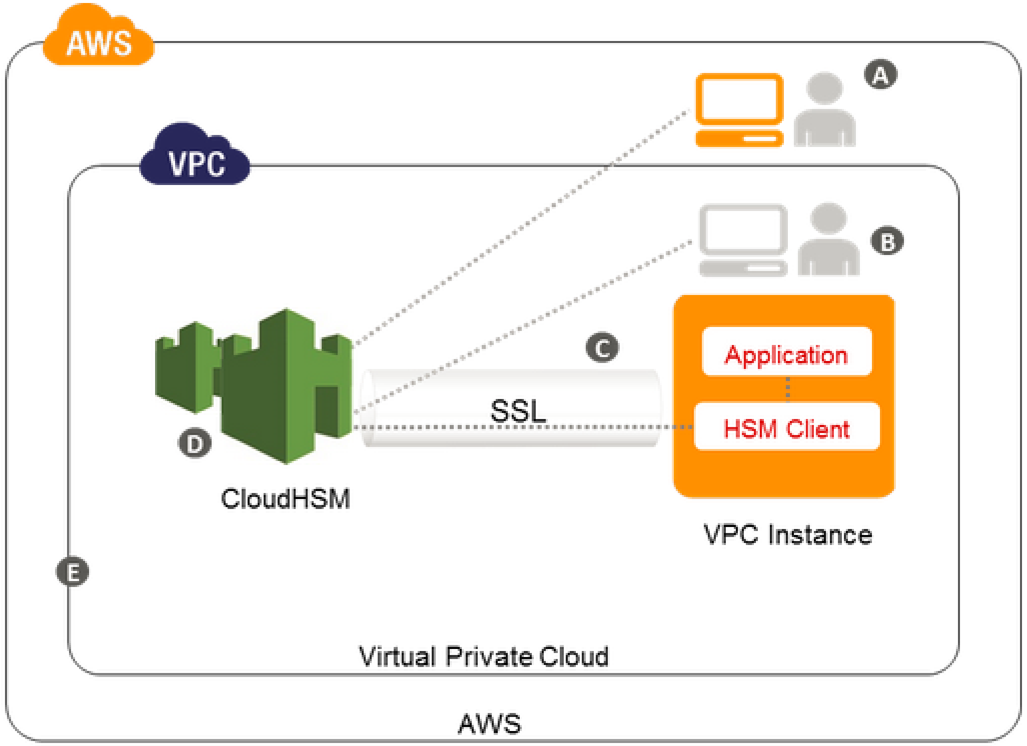
Ⓐ AWS manages the HSM appliance but does not have access to your keys

Ⓑ You control and manage your own keys

Ⓒ Application performance improves (due to close proximity with AWS workloads)

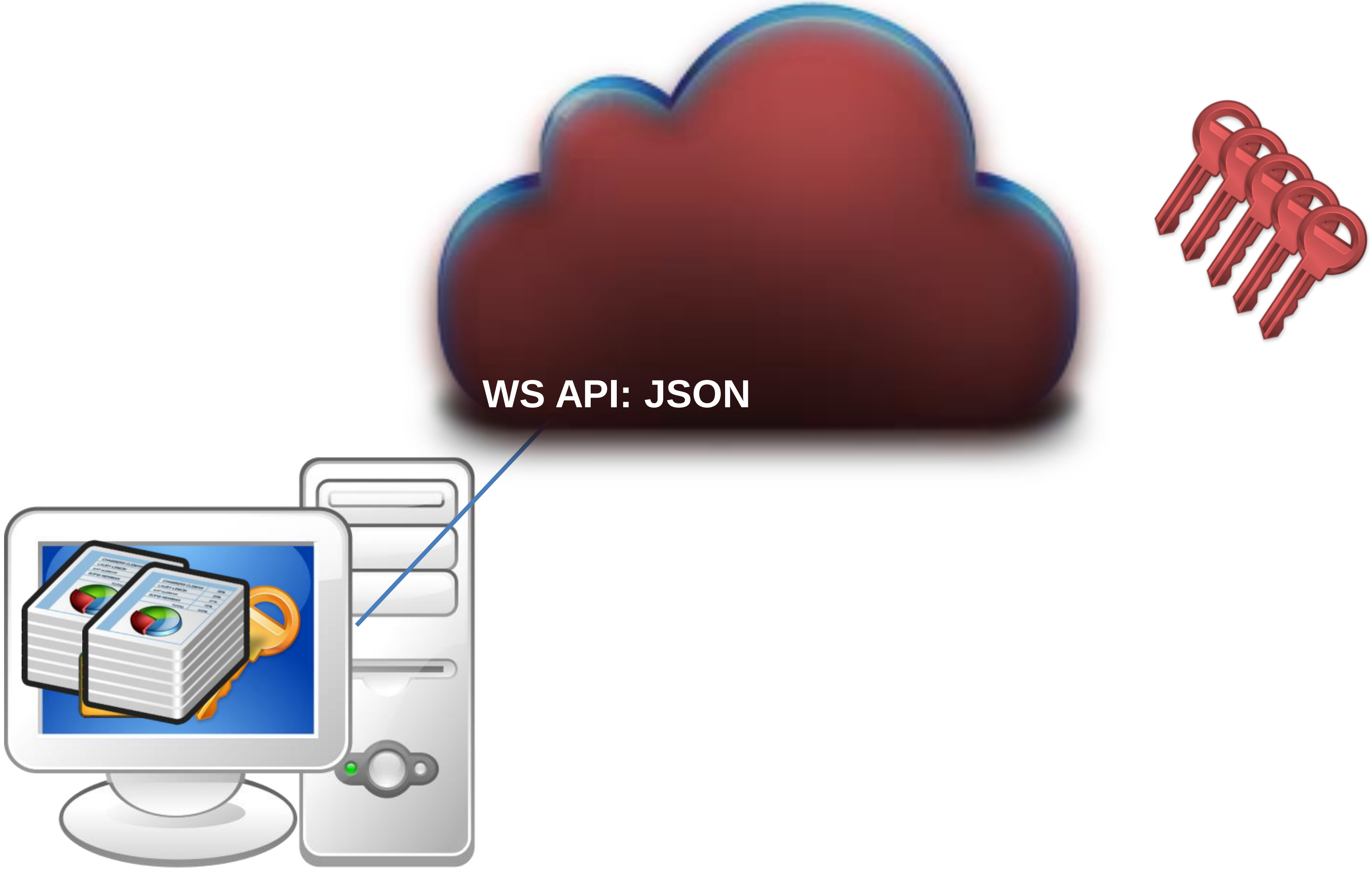
Ⓓ Secure key storage in tamper-resistant hardware available in multiple regions and AZs

Ⓔ CloudHSMs are in your VPC and isolated from other AWS networks

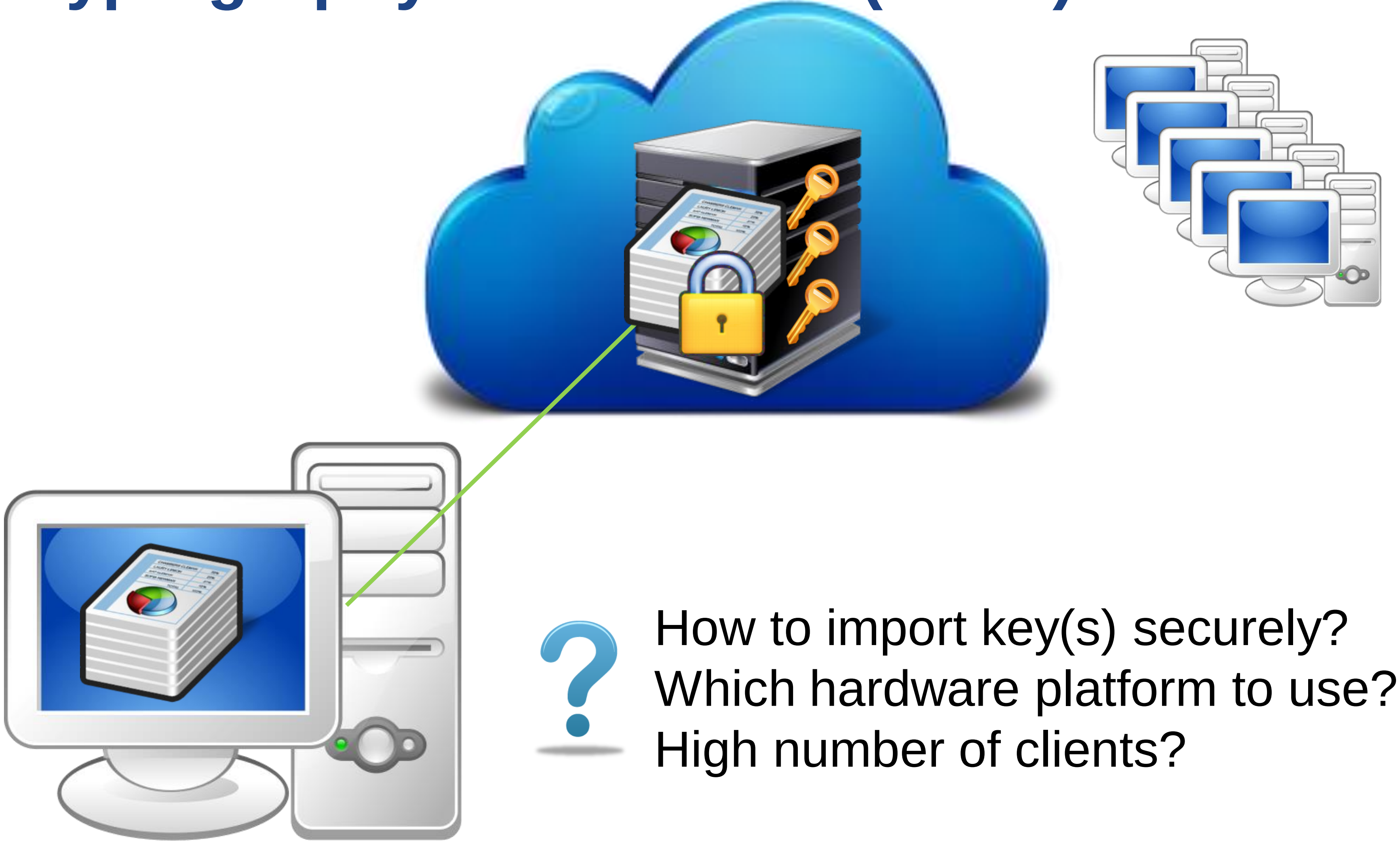


CRYPTOGRAPHY AS A SERVICE

Offloading security operations…



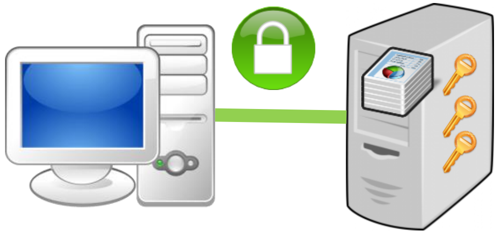
… into secured environment
Cryptography as a Service (CaaS)



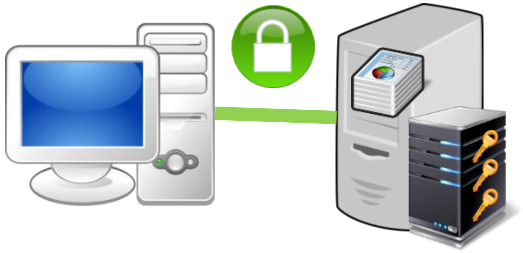
How to import key(s) securely?
Which hardware platform to use?
High number of clients?

Different levels of trust

- CaaS with trusted server
  - Software operation only, HTTPS for in/out
  - Trust to server, CaaS platform is target, insider attack



- CaaS with semi-trusted server
  - HTTPS for in/out, decrypted by server
  - Operation send into trusted hardware
  - CaaS platform still target



- CaaS with untrusted server
  - HTTPS for in/out, but inner protection
  - Data decrypted/processed/encrypted inside device

Requirements – client view

- Untrusted CaaS provider (handling secrets)
- Secure import of app’s secrets - enrollment
- Client<->CaaS communication security
  - Confidentiality/integrity of input and output data
  - Authentication of input/output requests
- Key use control
  - Use constraints – e.g., number of allowed ops
- Easy recovery from client-side compromise

Requirements – CaaS provider view

- Massive scalability
  - W.r.t. users, keys, transactions…
- Low latency of responses
- Robust audit trail of key usage
- Tolerance and recovery from failures
  - hardware/software failures
- Easy to use API
  - also easy to use securely

Hardware options for CaaS

- Use of general-purpose hardware (CPU, GPU)
- Use of generic programmable hardware (FPGA)
- Use of dedicated cryptographic circuits (ASICs)
- Use of secure processors (HSMs, smartcards)

- (use of additional tamper protection of device)
- (use of fully homomorphic encryption)

CaaS - implementation issues

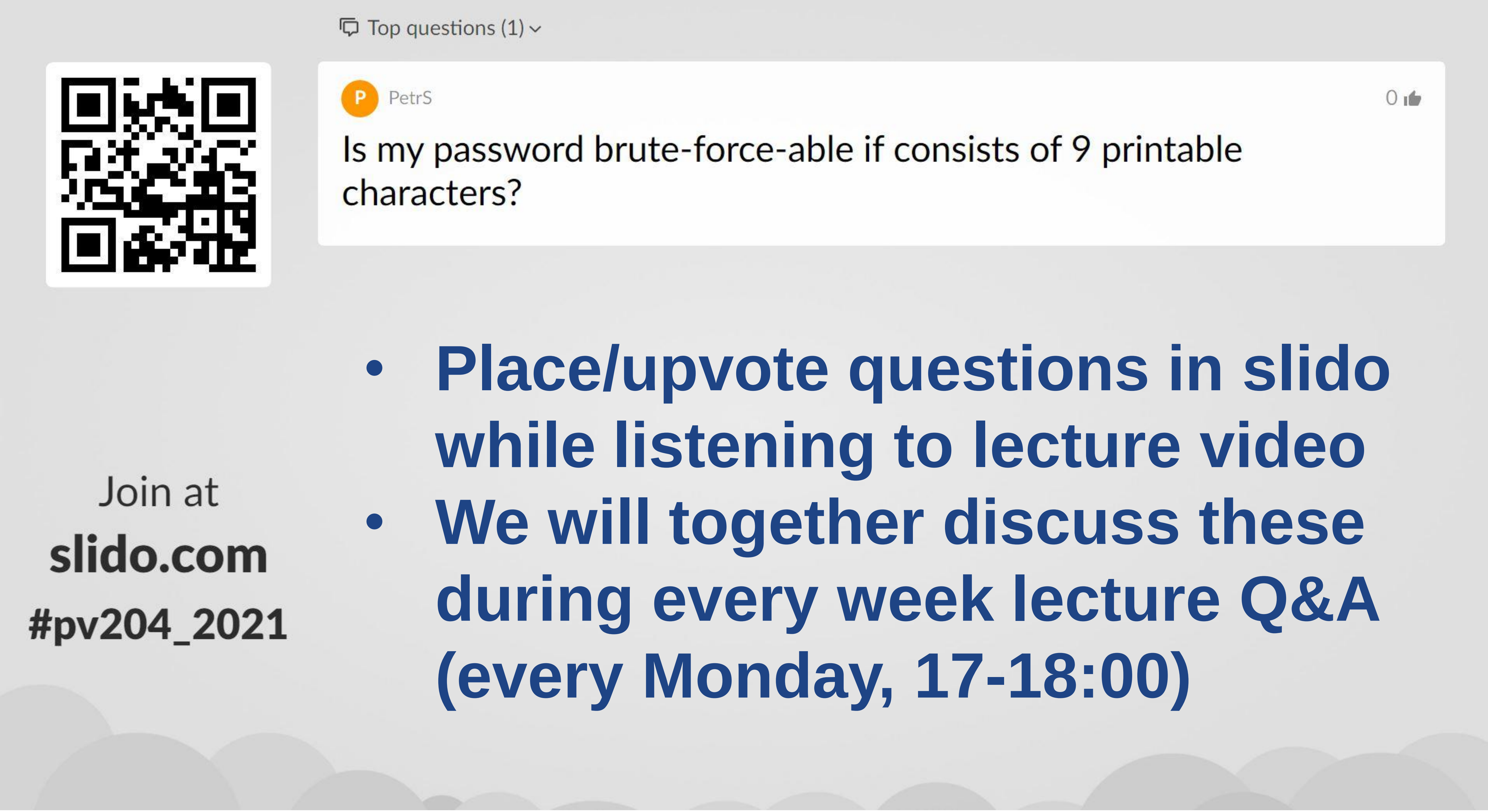
- Security
  - Software-only CaaS more vulnerable to attacks
  - Access control and operation authorization critical
- Performance
  - Classic HSMs are not build for high-level of sharing
  - Performance degradation due to frequent context exchange (key scheduling, engine preparation)
  - Logical separation only to few entities (16-32)
  - Physical separation on device-level (=> very limited)

Secure parallel multi-processor

- High number of secure processors
  - Secure memory, secure execution, crypto engines
  - FIPS140-2 Level 3/4, CC EAL 4+
- Secure channels between secure processors
- Untrusted controller
  - Small trusted computing base
  - Initialization/operational phase
- Restricted use and audit trail (=> state)
- High-speed I/O data interface
- High robustness due to high redundancy
  - If one card lock or die, other will serve a request
- Physical separation of secure processors

Conclusions

- Hardware Security Module is device build for security and performance of cryptographic operations
- Security certifications (but be aware of limits)
- Initially mostly for banking sector
  - Now more widespread (TLS, key management..)
- As applications are moving to cloud, so do HSMs
  - Full HSM (legacy apps), or HSM-backed functionality (e.g., KMS)
- Diverse APIs, potential logical attacks



Top questions (1)

PetrS 0

Is my password brute-force-able if consists of 9 printable characters?

Join at
slido.com
#pv204_2021

- **Place/upvote questions in slido while listening to lecture video**
- **We will together discuss these during every week lecture Q&A (every Monday, 17-18:00)**

THANK YOU FOR YOUR TIME!

PKCS#11 DETAILS

PKCS#11: Function prototypes

- GetProcAddress() returns untyped function pointer
- We need to cast this function pointer to known function type
- Function types for PKCS#11 are in pkcs11_ft.h

```
typedef CK_RV CK_ENTRY (*FT_C_Encrypt)(
  CK_SESSION_HANDLE hSession,
  CK_BYTE_PTR       pData,
  CK_ULONG          ulDataLen,
  CK_BYTE_PTR       pEncryptedData,
  CK_ULONG_PTR      pulEncryptedDataLen
);
```

PKCS#11: Load and init library

```
int LoadAndInitLibrary(const char* path, HINSTANCE* phLib) {
    CK_RV   status = CKR_OK;
    FT_C_Initialize fInitialize = NULL;

       if (phLib) {
        if ((*phLib = LoadLibrary(path)) != NULL) {
            // INITIALIZE LIBRARY
            fInitialize = NULL;
            if ((fInitialize = (FT_C_Initialize) GetProcAddress(*phLib, "C_Initialize")) != NULL) {
                (fInitialize)(NULL);
            }
            else status = GetLastError();
        }
        else status = GetLastError();
    }
    else status = -1;

    return status;
}
```

PKCS#11: Finalize and unload library

```
int FinalizeAndCloseLibrary(HINSTANCE hLib) {
    CK_RV   status = CKR_OK;
    FT_C_Finalize   fFinalize;
       if (hLib != NULL) {
        // UNINITIALIZE LIBRARY
        fFinalize = NULL;
        if ((fFinalize = (FT_C_Finalize) GetProcAddress(hLib, "C_Finalize")) != NULL) {
            (fFinalize)(NULL);
        }

        FreeLibrary(hLib);
    }
    else status = -1;

    return status;
}
```

PKCS#11: List tokens in system

- Slots in system are equivalent to readers
  - C_GetSlotList
  - C_GetSlotInfo
- Slot can be empty or with inserted token
  - C_GetTokenInfo

PKCS#11: Connect to token

- When slot with token is found
  - C_OpenSession
  - public session is opened
- Switch to private session by inserting PIN
  - C_Login
  - C_Logout
- C_CloseAllSessions

PKCS#11: arguments lists

- Most of the PKCS#11 functions accept parameters as CK_ATTRIBUTE[] array
- Every value is encoded in single CK_ATTRIBUTE
  - CK_ATTRIBUTE_TYPE type
  - CK_VOID_PTR pValue
  - CK_ULONG ulValueLen

```
CK_CHAR label_public[] = {"Test1_public"};      //label of data object
CK_CHAR data_public[] = {“PV204 Public"};
CK_ATTRIBUTE dataTemplate_public[] = {
	{CKA_CLASS, &dataClass, sizeof(dataClass)},
	{CKA_TOKEN, &ptrue, sizeof(ptrue)},
	{CKA_LABEL, label_public, sizeof(label_public)},
	{CKA_VALUE, (CK_VOID_PTR) data_public, sizeof(data_public)},
	{CKA_PRIVATE, &pfalse, sizeof(pfalse)}  // is NOT private object
};
BYTE    numAttributes_public = 5;
C_CreateObject(hSession, dataTemplate_public, numAttributes_public, &hObject);
```

PKCS#11: Store/search/get data

- Data created in public/private part of the token
  - CKA_PRIVATE attribute
  - C_CreateObject()
- User must be logged when creating/read private objects
- You must find target object
  - attribute template, must be logged when searching private objects
  - C_FindObjectsInit()
  - C_FindObjects()
  - C_FindObjectsFinal()
- Read data from object
  - C_GetAttributeValue()

HSM SECURITY API

Microsoft CNG

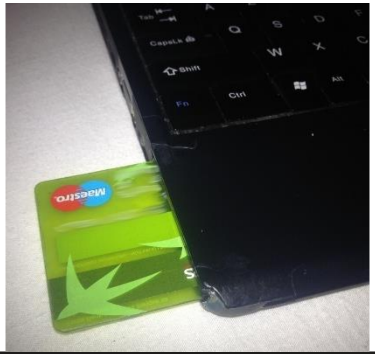
- Cryptography API: Next Generation (CNG API)
- Long-term replacement for CryptoAPI
- CNG API
  - Cryptographic Primitives
  - Key Storage and Retrieval
  - Key Import and Export
  - Data Protection API: Next Generation (CNG DPAPI)
- http://msdn.microsoft.com/en-us/library/windows/desktop/aa376210%28v=vs.85%29.aspx

Cryptographic Service Providers (CSP)

- Generic framework with API for providers of cryptographic functionality
  - E.g., implementation of RSA
  - Different underlying storage (software vs. hardware-based)
- Allows for runtime selection
  - Connect to target provider (usually identification string)
  - E.g., “Microsoft Base Cryptographic Provider v1.0”
- Microsoft CSPs
  - http://msdn.microsoft.com/en-us/library/windows/desktop/aa386983%28v=vs.85%29.aspx
- Java CSPs (JCE)…

Chip Authentication Program (CAP)

- Usage of chip-based banking card for additional operations
- Designed for backward compatibility
  - existing cards can be used
  - Separate on-card applet is preferred, but not required
- Designed by MasterCard as EMV-CAP
  - https://en.wikipedia.org/wiki/Chip_Authentication_Program
  - Adopted by Visa as Dynamic Passcode Authentication (DPA)
- Hardware CAP readers available
- Python software implementation
  - http://sites.uclouvain.be/EMV-CAP/Application/



CAP – supported commands

- Supported operations
  - Code/identify
  - Response
  - Sign
- Variants:
  - Mode 1: amount included in computed cryptogram
  - Mode 2: no amount, used for logging into system
  - Mode 2 + TDS
    - With transaction data signing
    - Multiple data fields of the transaction

Custom API pro/cons

- Is design of own API better idea?
- Pros:
  - derive api in line with use
  - focused api, no overhead
  - highly efficient implementation
- Cons:
  - security holes by design
  - high effort
  - lost certification

PKCS#11, (PKCS#15), ISO/IEC 7816-15

- Standards for API of cryptographic tokens
- PKCS#11
  - http://www.rsa.com/rsalabs/node.asp?id=2133
  - software library on PC, rather low level functions
  - widely used, TrueCrypt, Mozilla FF/TB, OpenSSL, OpenVPN…
- PKCS#15
  - http://www.rsa.com/rsalabs/node.asp?id=2141
  - both hardware and software-only tokens, identity cards…
  - superseded by ISO/IEC 7816-15 standard

PKCS#11 v3.0

- Public Review Draft 01, 29 May2019
- https://docs.oasis-open.org/pkcs11/pkcs11-base/v3.0/csprd01/pkcs11-base-v3.0-csprd01.pdf

PKCS#15 (https://github.com/OpenSC)

- pkcs15-init

```
#!/bin/bash

sleep 5
pkcs15-init --create-pkcs15 --pin 12345678 --no-so-pin
sleep 5
pkcs15-init --store-pin --auth-id 01 --pin 12345678
            --puk 12345678 --label “PV204 Tutorial"
```

- pkcs15-tool --dump
- pkcs15-tool --list-keys

ATTACKS AGAINST API

Attacks against PKCS#11

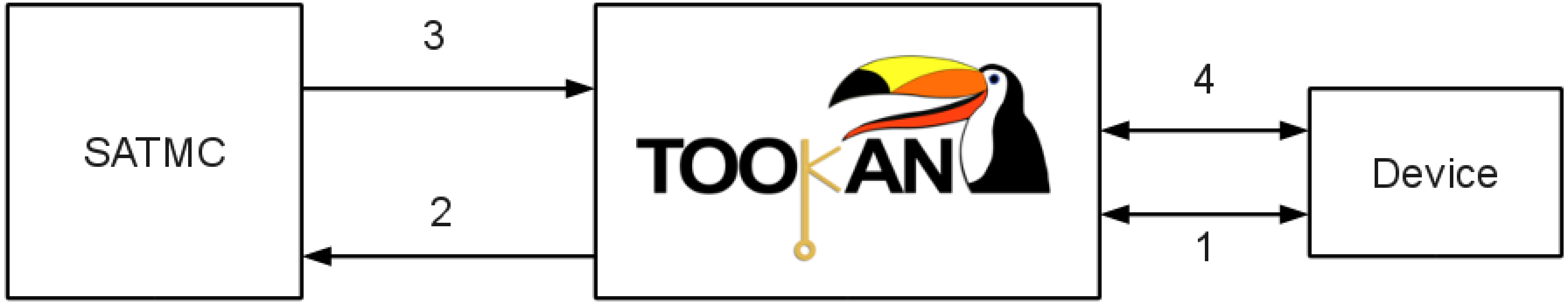
- Lack of policy for function calls
  - functions are too “low-level”
  - sensitive objects can be manipulated directly
- Key binding attack (C_WrapKey)
  - target key with double length is exported from SC
  - encrypted by unknown master key
  - attacker divide key into two parts and import them as wrapped key for ECB mode
  - perform brute-force search on each half separately
- Missing authentication of wrapped key
  - attacker can create its own wrapping key
  - and ask for export of unknown key under his own wrapping key
- Export of longer keys under shorter, …

RSA padding oracle attack

- Allows to recover content of encrypted message even when key is unknown
- Based on 1 bit leakage from correct/incorrect padding
  - Error status returned by device
- (cycle) mess with encrypted message, send to card, inspect error
- 30 minutes with HSM, hours/days with smart card
- See more at
  - http://secgroup.dais.unive.it/wp-content/uploads/2012/11/Practical-Padding-Oracle-Attacks-on-RSA.html

Tookan tool

- Formal verification with real device model
  - probe PKCS#11 token with multiple function calls
  - automatically create formal model for token
  - run model checker and find attack
  - try to execute attack against real token
- http://secgroup.dais.unive.it/projects/tookan/



CERTIFICATION: MORE DETAILS

Certifications: NIST FIPS 140-2

- Requirements on hardware and software components of security modules to be used by US government
  - Verified under Cryptographic Module Validation Program (CMVP)
  - Testing against a defined cryptographic module, provides a suite of conformance tests to required security level
  - List of validated devices http://csrc.nist.gov/groups/STM/cmvp/validation.html
- Common levels for HSMs
  - NIST FIPS 140-2 Level 1+2 – basic levels, tamper evidence (broken shell, epoxy), role-based authentication (user/admin))
  - NIST FIPS 140-2 Level 3 – addition of physical tamper-resistance, identity-based authentication, separation of interfaces with different sensitivity

Certifications: NIST FIPS 140-2 (cont.)

- Common levels for HSMs (cont.)
  - NIST FIPS 140-2 Level 4 + additional physical security requirements, environmental attacks
  - http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf
  - Only very few devices certified to FIPS 140-2 Level 4
- NIST FIPS 140-3 (since 2013, for long time only draft, abanded then accepted in 2019)
  - Additional focus on software security and non-invasive attacks
  - https://csrc.nist.gov/projects/fips-140-3-transition-effort
  - Testing shall begin in September 2020, till 2026 in parallel with FIPS 140-2

NIST FIPS 140-2 and RNG

- Truly random number generators (TRNG)
  - No approved FIPS 140-2 TRNG
- Pseudorandom number generators
  - ANSI X9.31 Appendix A.2.4, 3DES/AES-based
- FIPS 140-2 requires testing of RNG
  - Known-answer-tests (KAT), Diehard battery

"Random" FIPS 140-2 example

- https://csrc.nist.gov/projects/cryptographic-module-validation-program
- Futurex EXP9000 HSM (07/2011)
  - https://csrc.nist.gov/projects/cryptographic-module-validation-program/Certificate/1577
  - FIPS140-2, security level 3
  - Approved algorithms
  - Non approved algorithms
  - Roles and authentication
  - Critical Security Parameters (CSP)
  - Physical security mechanisms
  - …

Certifications: Common Criteria EAL 4-5+

- CC does not directly measure the security of the system/device itself
  - only states level on which the system/device was tested
  - and against what Security Target
- To achieve particular level, system must meet assurance requirements
  - Documentation, design analysis, functional/penetration testing
- CC certifies that system followed certain rules when implementing target goals
  - Broader than FIPS 140-2

Certifications: Common Criteria EAL 4-5+

Common Criteria

- Common levels for HSMs
  - EAL4: Methodically Designed, Tested and Reviewed
  - EAL5: Semi-formally Designed and Tested
- Protection profiles
  - Specifies generic security evaluation criteria to substantiate vendors' claims (more technical)
  - Crypto Module Protection Profile (BSI)
  - https://www.bsi.bund.de/cae/servlet/contentblob/480256/publicationFile/29291/pp0045b_pdf.pdf
- + means “augmented” version (current version + additional requirements, e.g., EAL4+)

Certifications: PCI HSM version 1,2,3

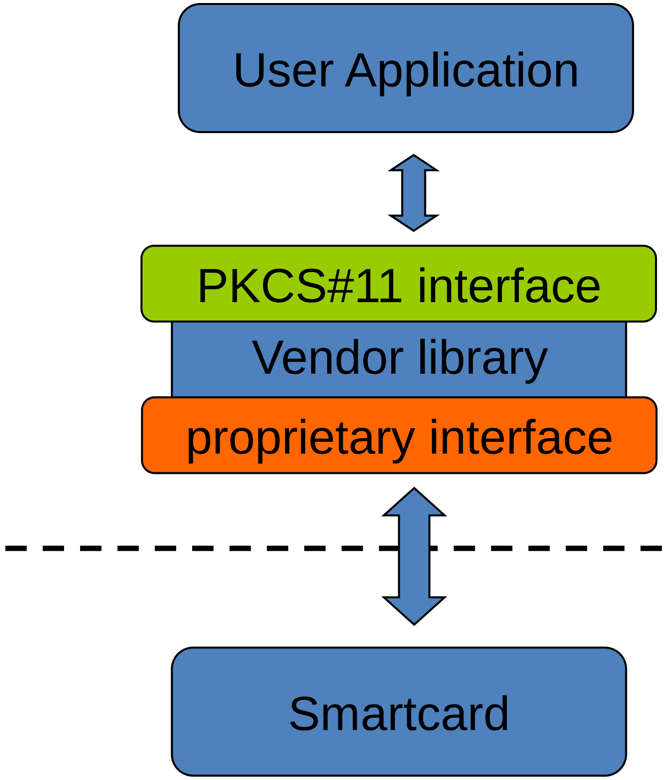
- PCI HSM v1 (2009), v2 (2012), v3 (2016)
  - https://www.pcisecuritystandards.org/security_standards/documents.php
- Focused on area of payment transactions
  - Payment terminals, backend HSMs…
  - Payment transaction processing
  - Cardholder authentication
  - Card issues procedure
- Set of logical and physical requirements relevant to payment industry
  - Closer to NIST FIPS 140-2 then to CC (more concrete requirements)

Hardware Security Module

HSM SECURITY API

PKCS#11

- Standardized interface of security-related functions
  - vendor-specific library in OS, often paid
  - communication library->card proprietary interface
- Functionality cover
  - slot and token management
  - session management
  - management of objects in smartcard memory
  - encryption/decryption functions
  - message digest
  - creation/verification of digital signature
  - random number generation
  - PIN management
- Secure channel not possible!
  - developer can control only App→PKCS#11 lib



PKCS#11 library

- API defined in PKCS#11 specification
  - – http://www.rsa.com/rsalabs/node.asp?id=2133
  - – functions with prefix ‘C_’ (e.g., C_EncryptFinal())
  - – header files pkcs11.h and pkcs11_ft.h
- Usually in the form of dynamically linked library
  - – cryptoki.dll, opensc-pkcs11.dll, dkck232.dll…
  - – different filenames, same API functions (PKCS#11)
- Virtual token with storage in file possible
  - – suitable for easy testing (no need for hardware reader)
  - – Mozilla NSS, SoftHSM…

PKCS#11: role model

- Functions for token initialization
  - outside scope of the specification
  - usually implemented (proprietary function call), but erase all data on token
- Public part of token
  - data accessible without login by PIN
- Private part of token
  - data visible/accessible only when PIN is entered

PKCS#11: Cryptographic functionality

- C_GetMechanismList to obtain supported cryptographic mechanisms (algorithms)
- Many possible mechanisms defined (pkcs11t.h)
  - CK_MECHANISM_TYPE, not all supported
  - (compare to JavaCard API)
- C_Encrypt, C_Decrypt, C_Digest, C_Sign, C_Verify, C_VerifyRecover, C_GenerateKey, C_GenerateKeyPair, C_WrapKey, C_UnwrapKey, C_DeriveKey, C_SeedRandom, C_GenerateRandom…

PKCS#11 - conclusions

- Wide support in existing applications
- Low-level API
- Difficult to start with
- Requires proprietary library by token manufacturer
- Complex standard with vague specification => security problems
  - Hard to implement properly

Play with HSM (without HSM ☺)

- SoftHSM
  - Software-only HSM
  - Open-source implementation of cryptographic store
  - Botan library for cryptographic operations
  - https://www.opendnssec.org/softhsm/
  - https://github.com/disig/SoftHSM2-for-Windows
- Utimaco HSM simulator
  - https://hsm.utimaco.com/download/
  - Simulator of physical HSM (with PKCS#11 and other interfaces)