# PV204 Security technologies

## Hardware Security Modules (HSM), PKCS#11

**Petr Švenda** *svenda@fi.muni.cz* *@rngsec*
Centre for Research on Cryptography and Security, Masaryk University

CROCS
Centre for Research on Cryptography and Security

$$\begin{tikzcd}
 & \text{Hardware Security Modules (HSMs)} \arrow[dl] \arrow[dr] & \\
\text{Difference to smartcards} \arrow[d] & & \text{Use-cases} \arrow[d] \\
\text{HSM security} \arrow[d] & & \text{Usage of HSM in cloud} \arrow[d] \\
\text{Certifications (FIPS140-2, CC EAL)} \arrow[d] & & \text{Cryptography as a service} \arrow[ddl] \\
\text{Steps of HSM operations} \arrow[dr] & & \\
 & \text{API} &
\end{tikzcd}$$

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

## 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…

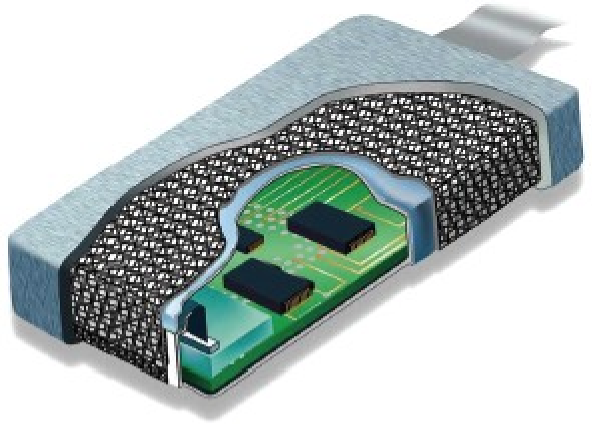
## Hardware Security Module - protection

- 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
- …

FIPS VALIDATED 140-2 TM

Common Criteria

PCI

## CERTIFICATIONS

## 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 (2013, but still draft, now abandoned)
    - Additional focus on software security and non-invasive attacks
- List of validated devices https://csrc.nist.gov/projects/cryptographic-module-validation-program

## Certifications: Common Criteria EAL 4-5+

- 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

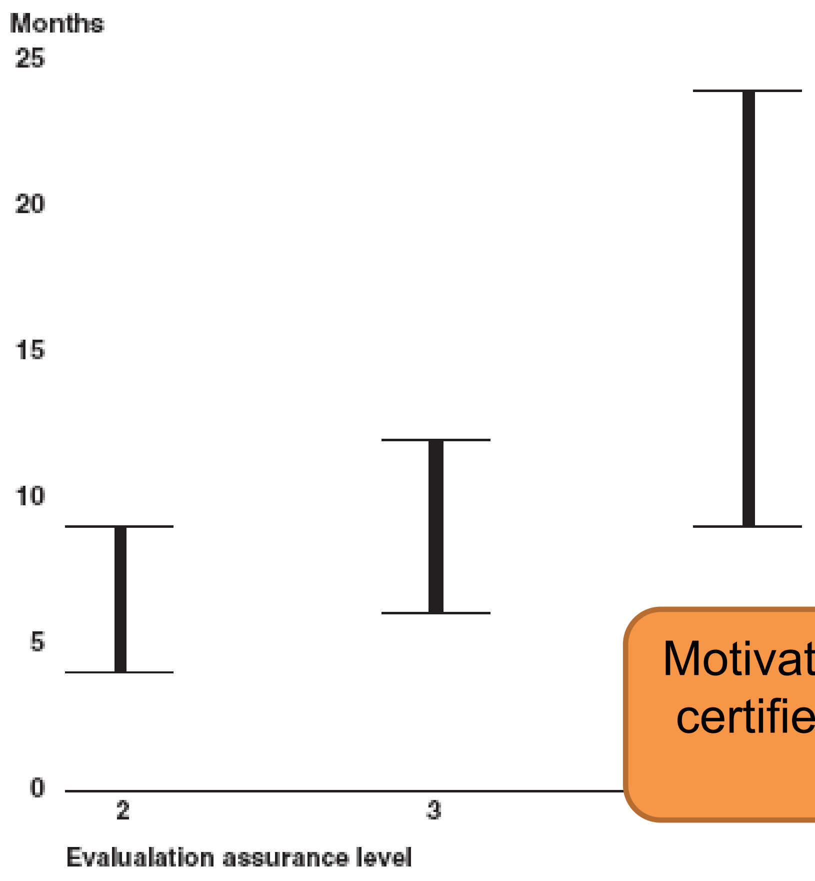
- PCI HSM v1 (2009), v2 (2012), v3 (2016)
  - https://www.pcisecuritystandards.org/security_standards/documents.php
- Focused on area of payment transactions
  - Payment terminals, backend HSMs…
  - Payment transaction processing
  - Cardholder authentication
  - Card issues procedure
- Set of logical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

## 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)



Source: GAO analysis of data provided by laboratories.

Motivation to keep already certified version longer in production

## Be aware what is actually certified

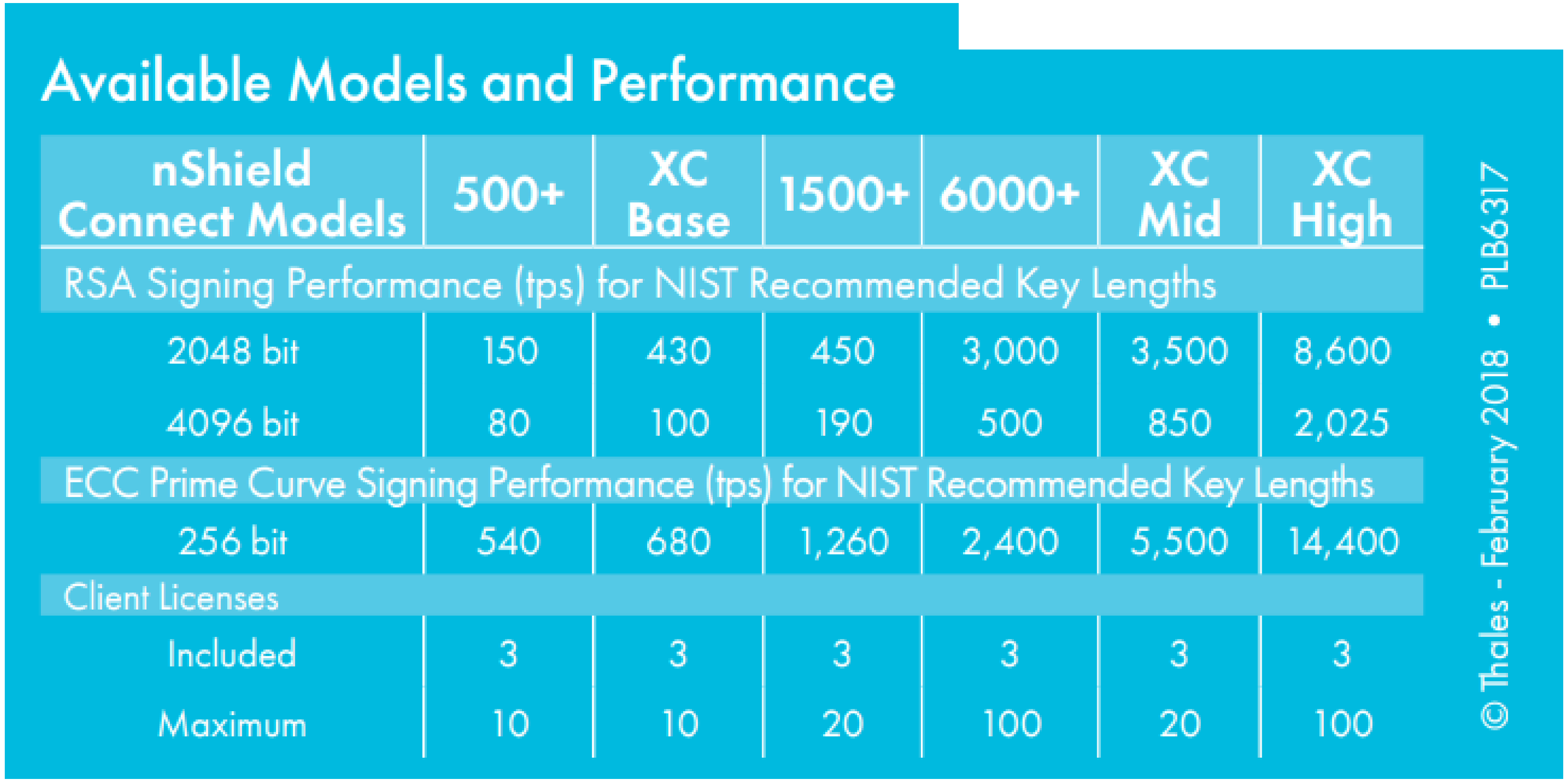
- Certified != secure
  - Satisfies defined criteria, producer claims were verified to be valid
  - Infineon’s RSA prime generation algorithm (BSI, CVE-2017-15361)
- Usually certified bundle of hardware and software
  - Concrete underlying hardware
  - Concrete version of firmware, OS and pre-loaded 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”

## 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)
  - Internal vs. external speed (data in/out excluded)
  - Measurements in “optimal” situations (single pre-prepared key, large data blocks…)
  - …

## 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

## S1: One user, few keys

- No sharing, all engines fully prepared

1. Transfer input data

7. Start, execute and finalize crypto operation

11. Transfer output data

## S2: One user, many keys

- No sharing, frequent crypto context change

1. Transfer input data
2. Transfer wrapped key in

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

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

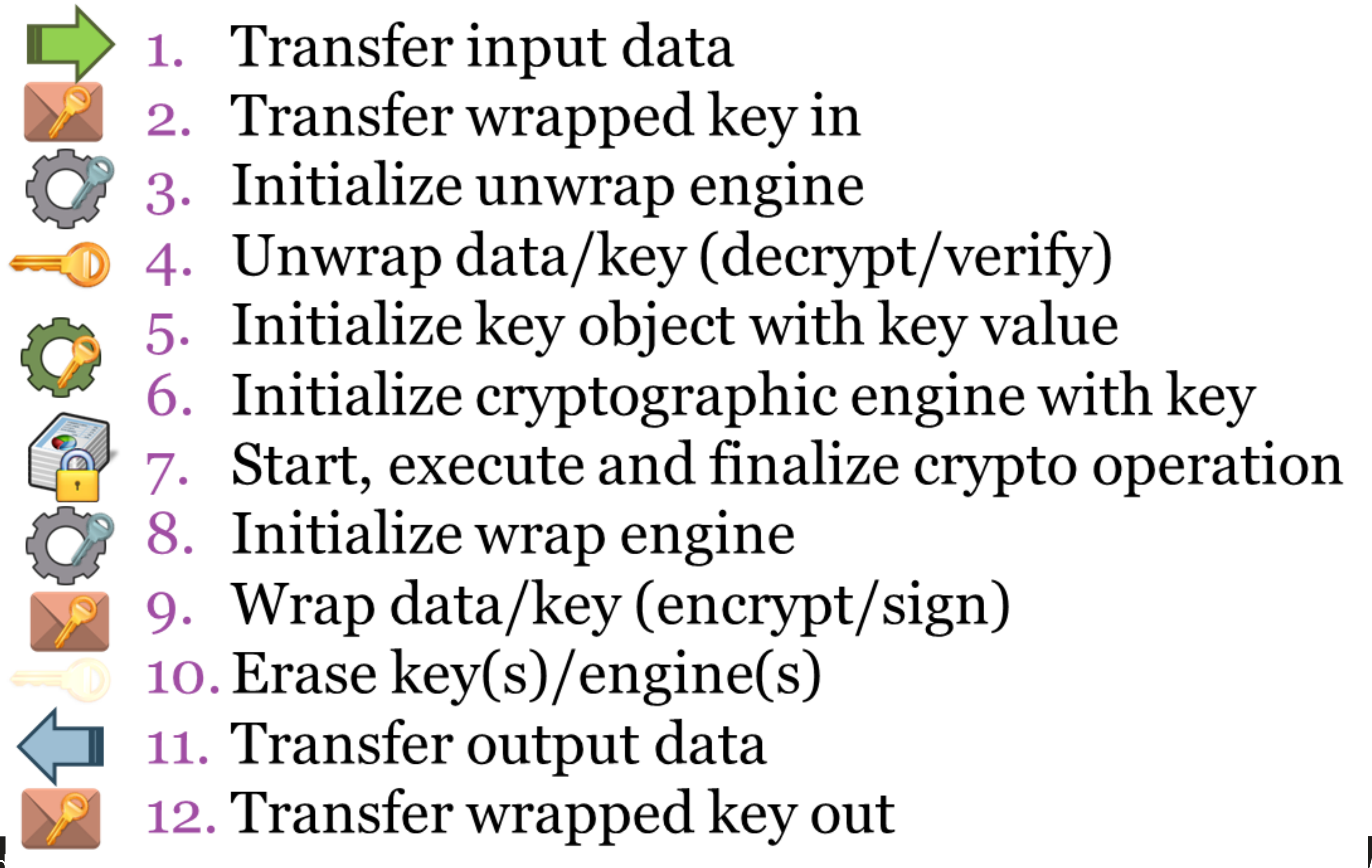
1. Transfer input data

6. Initialize cryptographic engine with key
7. Start, execute and finalize crypto operation

10. Erase key(s)/engine(s)
11. Transfer output data

## S5: Many users, many keys

- High sharing, frequent crypto context change

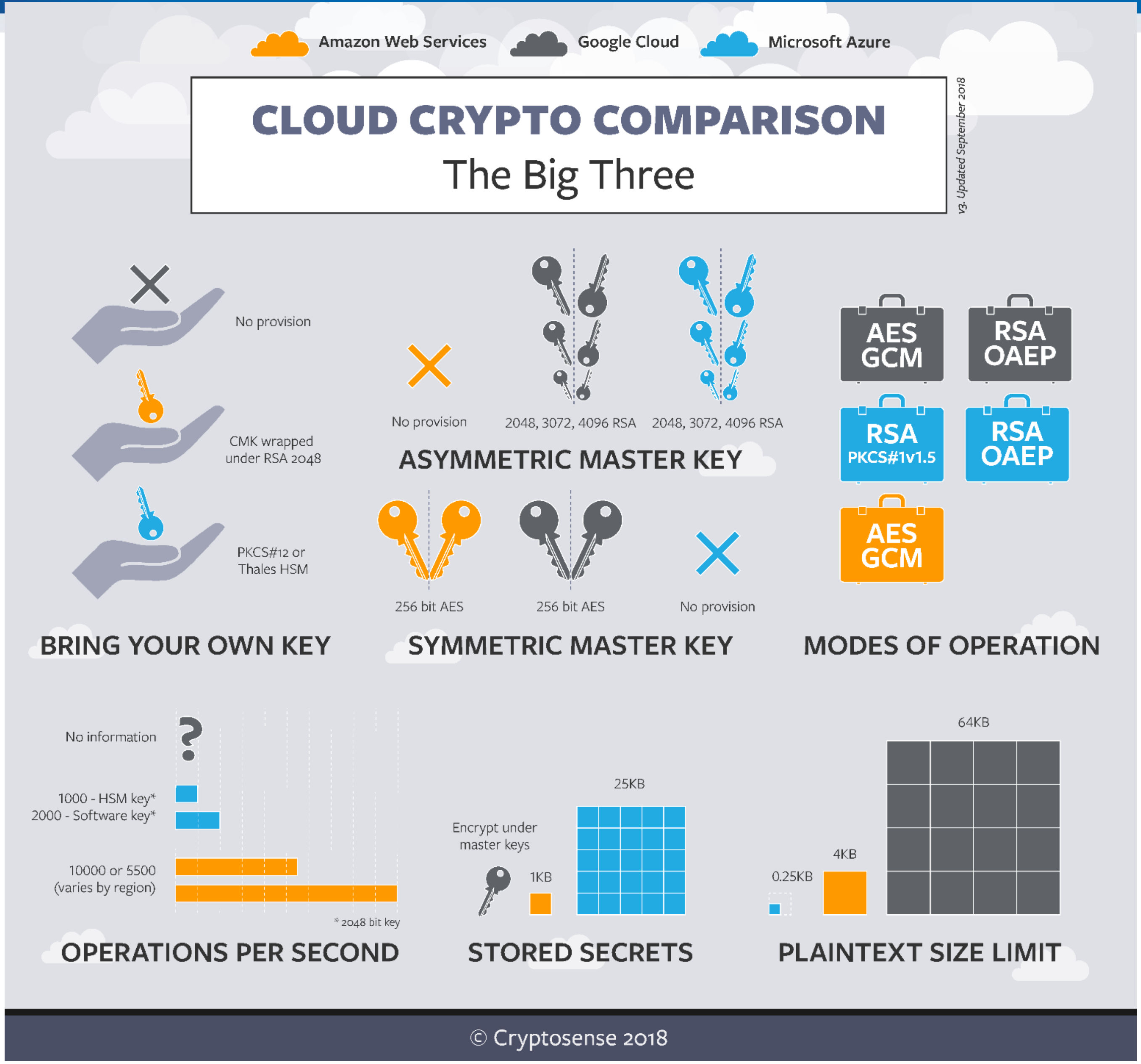


1. Transfer input data
2. Transfer wrapped key in
3. Initialize unwrap engine
4. Unwrap data/key (decrypt/verify)
5. Initialize key object with key value
6. Initialize cryptographic engine with key
7. Start, execute and finalize crypto operation
8. Initialize wrap engine
9. Wrap data/key (encrypt/sign)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

## HSM IN CLOUD

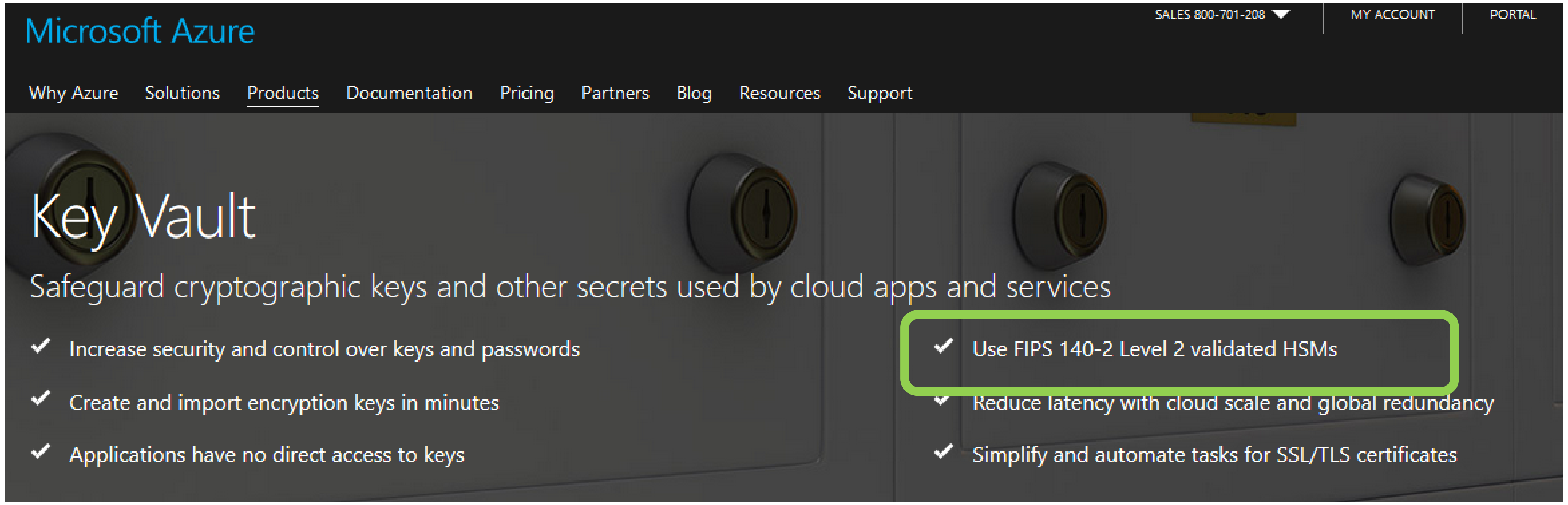
## Security topics in cloud environment

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



Microsoft Azure

SALES 800-701-208 | MY ACCOUNT | PORTAL

Why Azure | Solutions | Products | Documentation | Pricing | Partners | Blog | Resources | Support

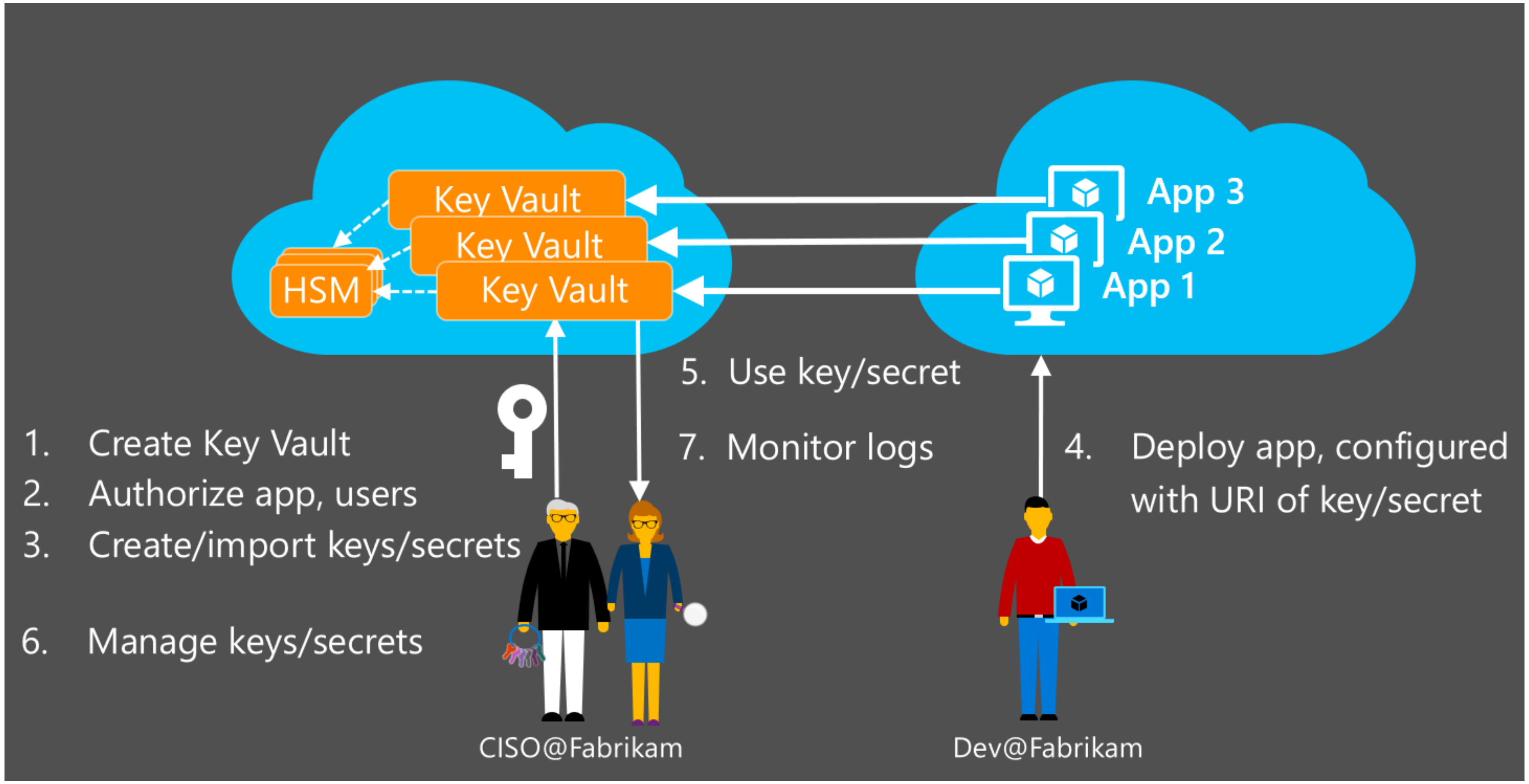
Key Vault

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/](https://docs.microsoft.com/en-us/rest/api/keyvault/)
- Language bindings (language specific wrappers)
  - JS, PowerShell, C#…

## Microsoft Azure KeyVault



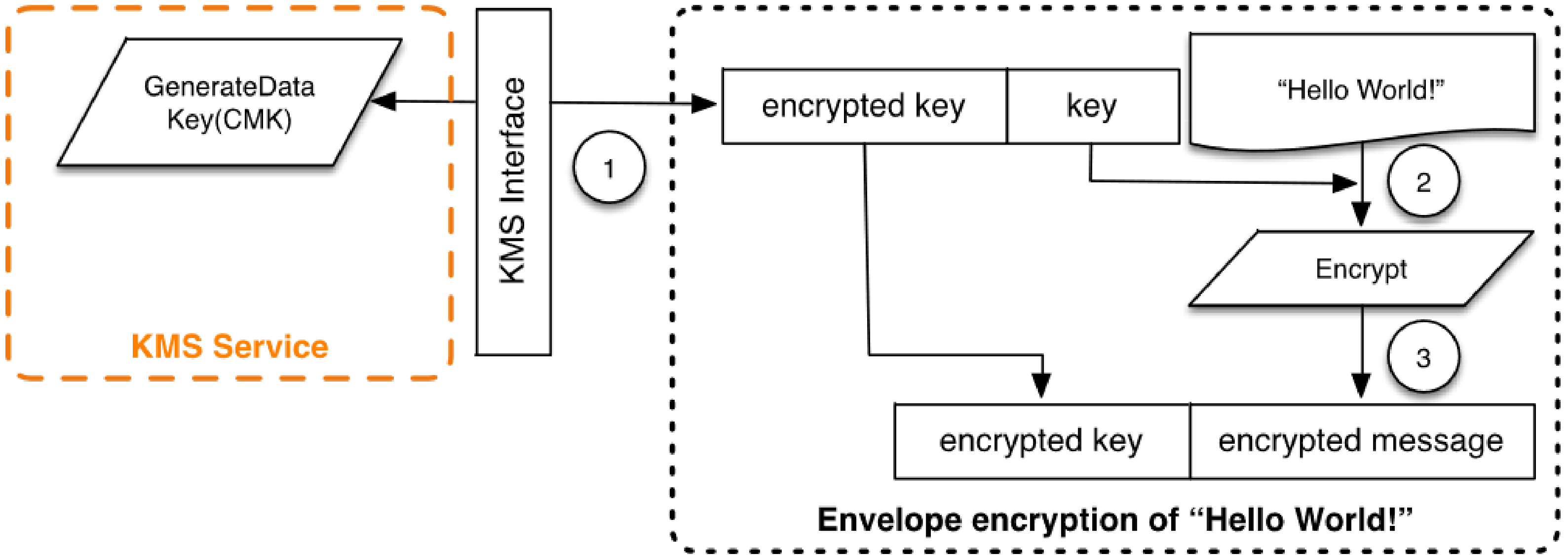
*https://channel9.msdn.com/Events/Ignite/2015/BRK2706*

## Use case: AWS Key Management Service

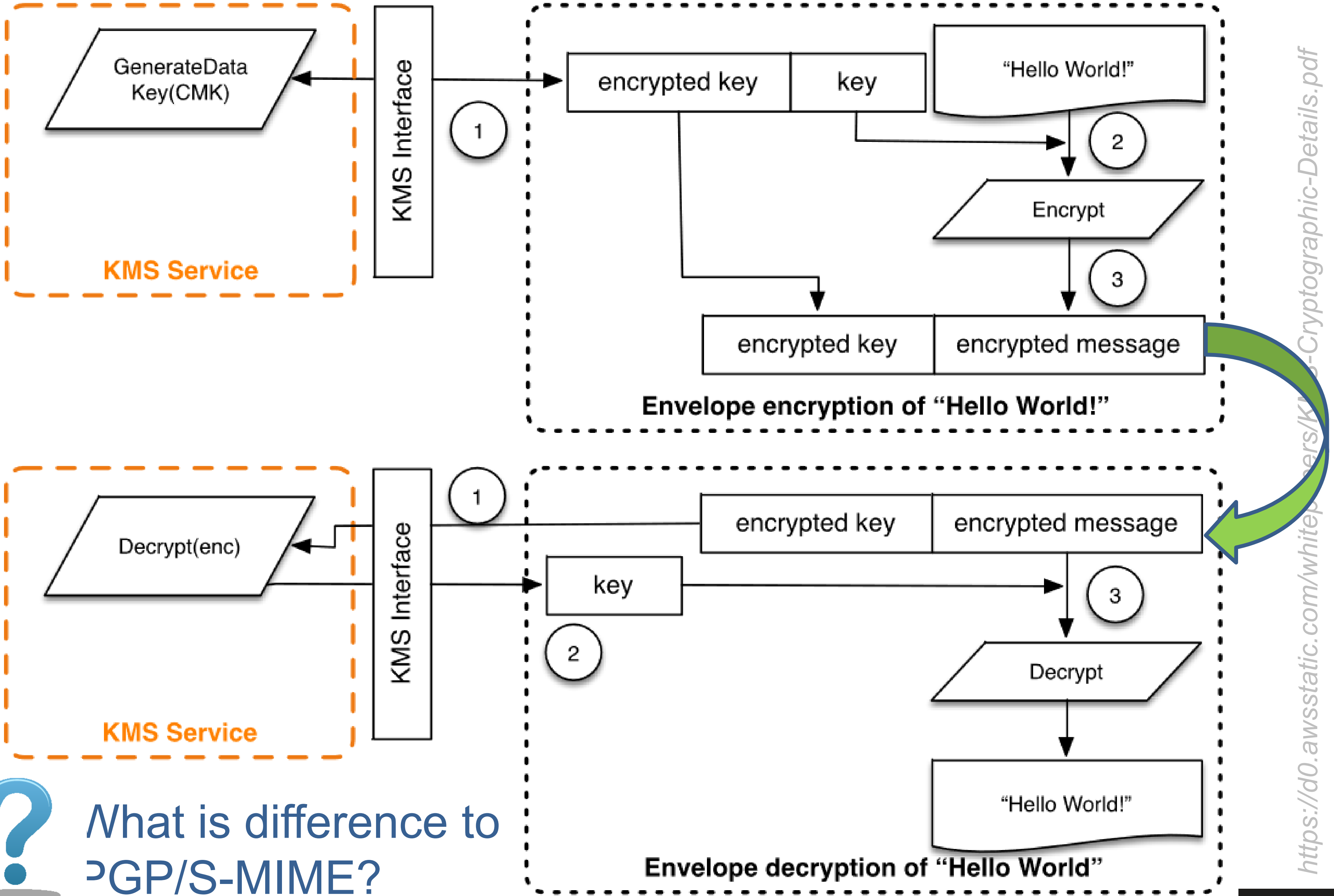
- AWS Key Management Service Cryptographic Details (2015)
  – https://d0.awsstatic.com/whitepapers/KMS-Cryptographic-Details.pdf
- Centralized key management
  – Used by cloud-based applications
  – Used by any client application
  – Replication of wrapping keys into HSMs in different datacenters

## 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)



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

CRoCS

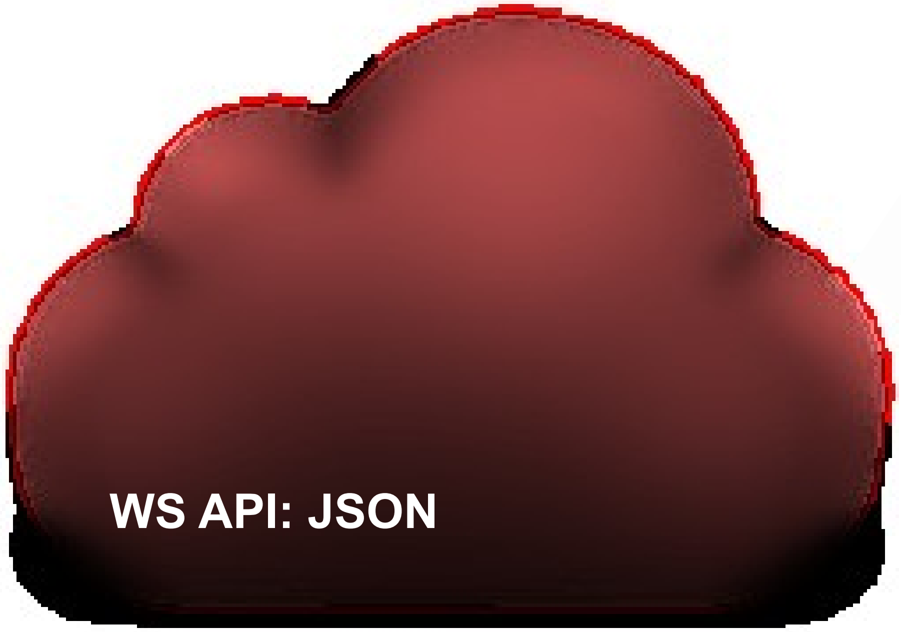
## Group activity: certification report (10 minutes)

- https://csrc.nist.gov/Projects/cryptographic-module-validation-program/Validated-Modules/Search
- ‘Show all’ option, pick any hardware module, quick read report
- What FIP140-2 level was achieved?
- What is approved cryptographic functionality?
- How is physical security protected? Side-channels?
- What kind of self-test are executed?
- Is the module also certified within Common Criteria?
- Interesting results (5 minutes)

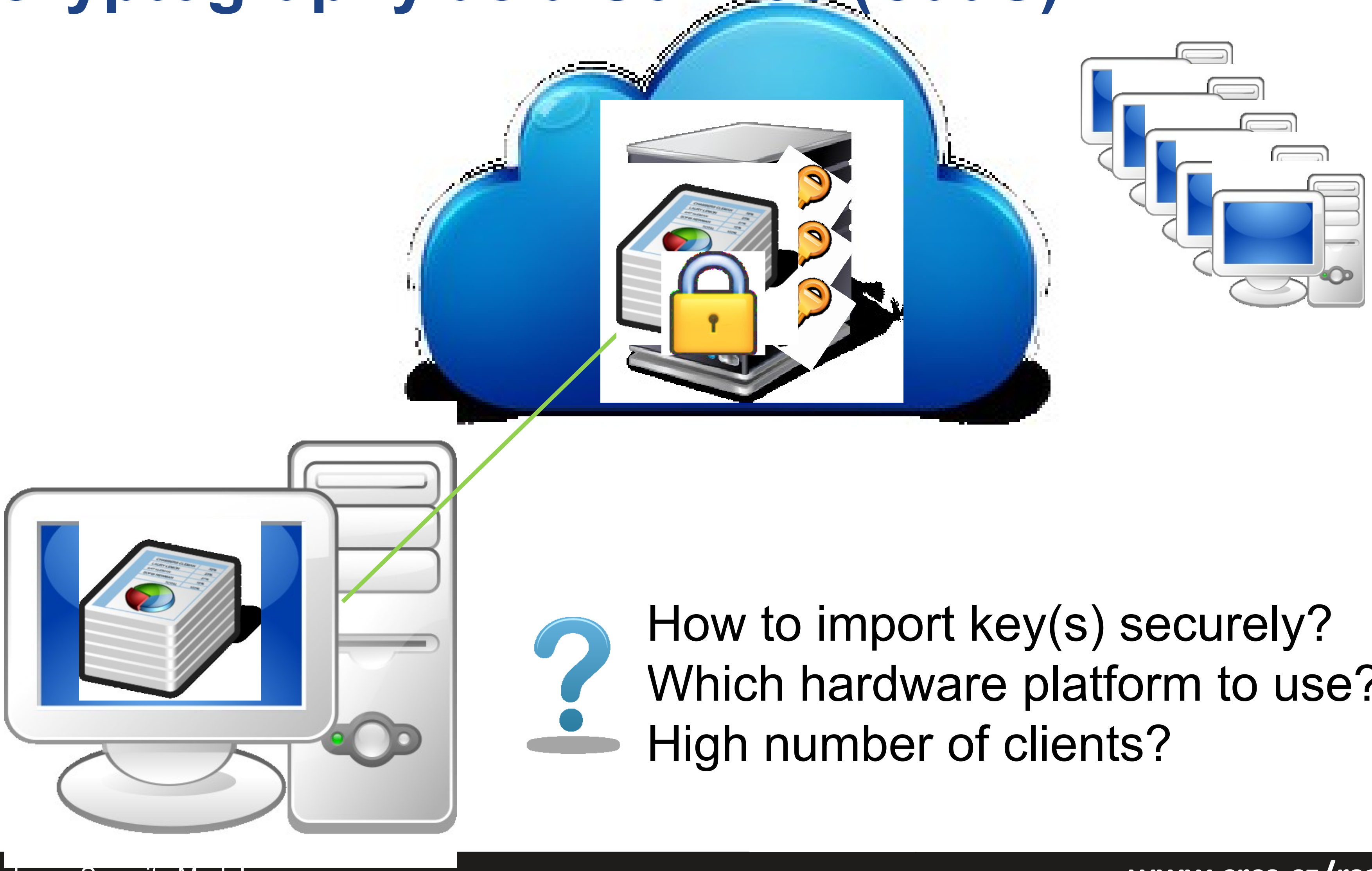
## CRYPTOGRAPHY AS A SERVICE

CR○CS

## Offloading s

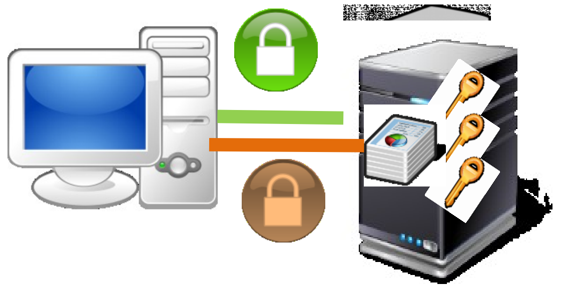


## … into secured environment Cryptography as a Service (CaaS)



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

## 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 Security Module

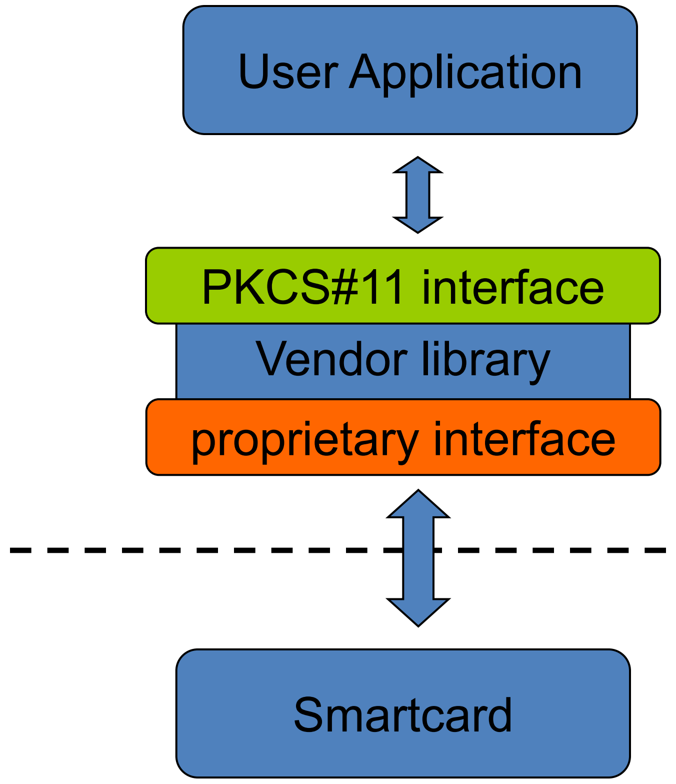
## HSM SECURITY API

## 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..)

## 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)

## 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
  - 
- Diverse APIs, potential logical attacks

## PKCS#11 DETAILS