PV204 Security technologies

Hardware Security Modules (HSM), PKCS#11

Petr Švenda svenda@fi.muni.cz
Faculty of Informatics, Masaryk University

CRoCS
Centre for Research on
Cryptography and Security

Overview

- Usage scenarios for HSMs
- Available hardware, security certifications
- Available security APIs (PKCS#11…)
- Known API-level attacks

Motivation usage scenarios

- Protection against trusted insiders
  - bank PIN processor
- Device with high impact of compromise
  - Private key of root certification authority
- Device in untrusted environment (ATM, PoS)
- DRM application (paid satellite TV)
- Smart grids (privacy of users)
- Intelligent transport systems…

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 generation (and entry)
  - Secure storage (and backup)
  - Secure use (cryptographic algorithms)
- Should never export sensitive data in plaintext
  - Especially keys = Critical Security Parameters (CSP)

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

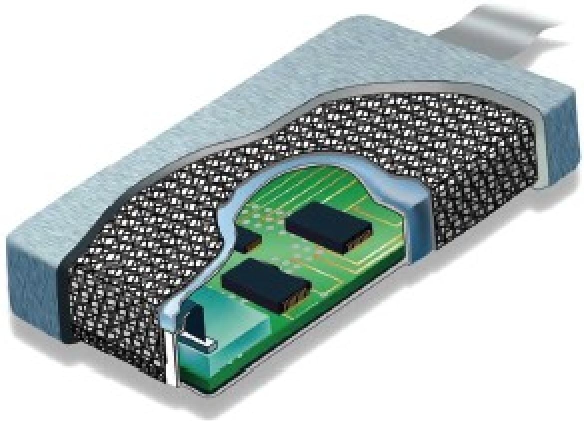
Hardware Security Module - specification

- 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 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 updates
- Logical separation of multiple users (memory)
  - Additional protection logic for separate memory regions
- Audit trails
- …

CERTIFICATIONS

Certifications: NIST FIPS 140-2

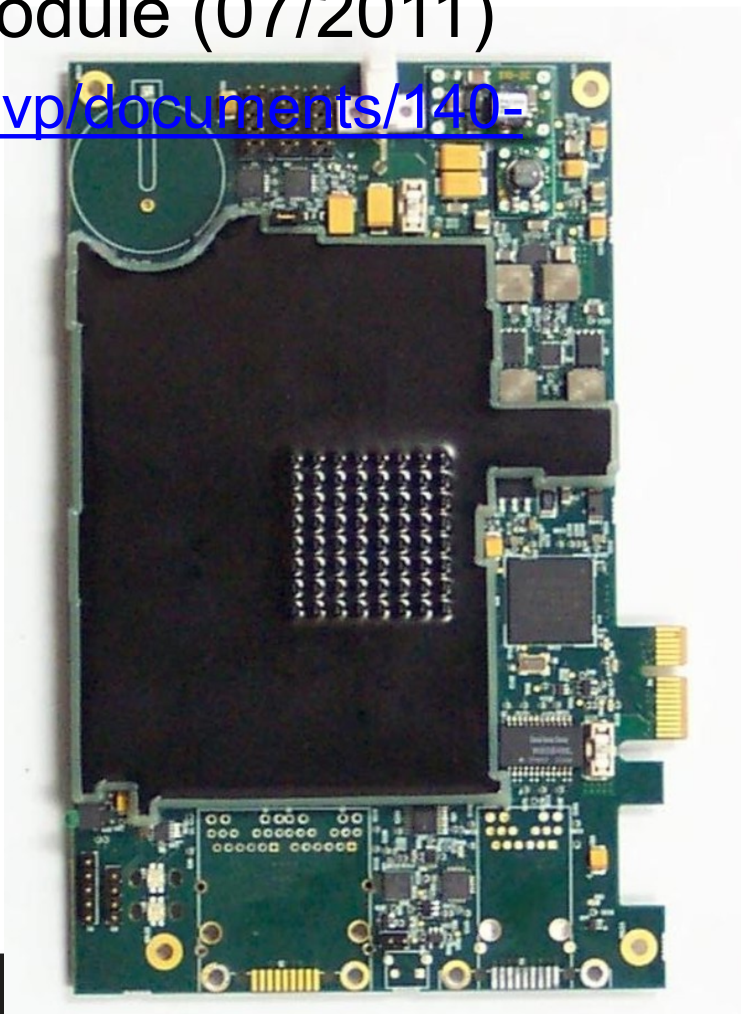
- Requirements on hardware and software components of security modules to be used by US government
  - Verified under Cryptographic Module Validation Program (CMVP)
  - Testing against a defined cryptographic module, provides a suite of conformance tests to required security level
  - List of validated 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 (2013, but still draft)
  - Additional focus on software security and non-invasive attacks

"Random" FIPS 140-2 example

- EXP9000 Hardware Security Module (07/2011)
  - http://csrc.nist.gov/groups/STM/cmvp/documents/140-1/140sp/140sp1577.pdf
  - FIPS140-2, security level 3
  - Approved algorithms
  - Non approved algorithms
  - Roles and authentication
  - Critical Security Parameters (CSP)
  - Physical security mechanisms
  - …



Common Criteria

Certifications: Common Criteria (cont.)

- 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
  - https://www.bsi.bund.de/cae/servlet/contentblob/480256/publicationFile/29291/pp0045b_pdf.pdf
- + means “augmented” version (current version + additional requirements, e.g., EAL4+)

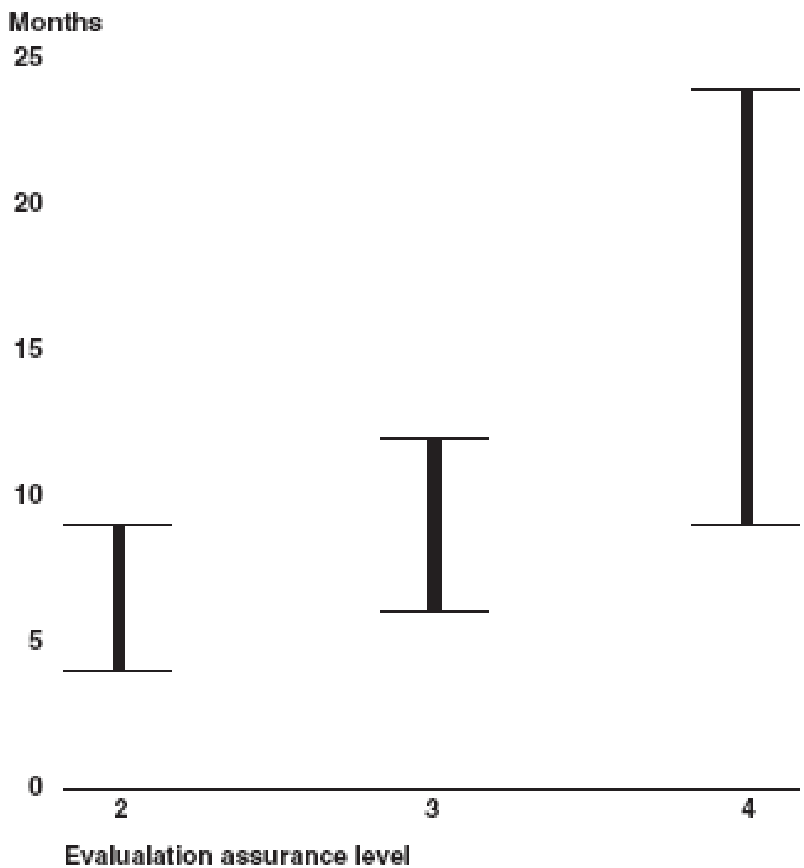
Certifications: PCI HSM version 1,2

- PCI HSM v1 (2009), v2 (2012)
  - 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
- Certification pre-study
  - Verify if product is ready for certification
- Full certification
  - Checklist if all required procedures were followed

Cost of CC EAL (US GAO, 2006)



Source: GAO analysis of data provided by laboratories.

Be aware what is actually certified

- Certified != secure
  - Satisfies defined criteria, producer claims are verified to be valid
- 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…)
  - …

HSM – performance II.

- 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

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

S4:Few users, many keys

- Limited 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

S5: Many users, many keys

- High sharing, frequent crypto context change

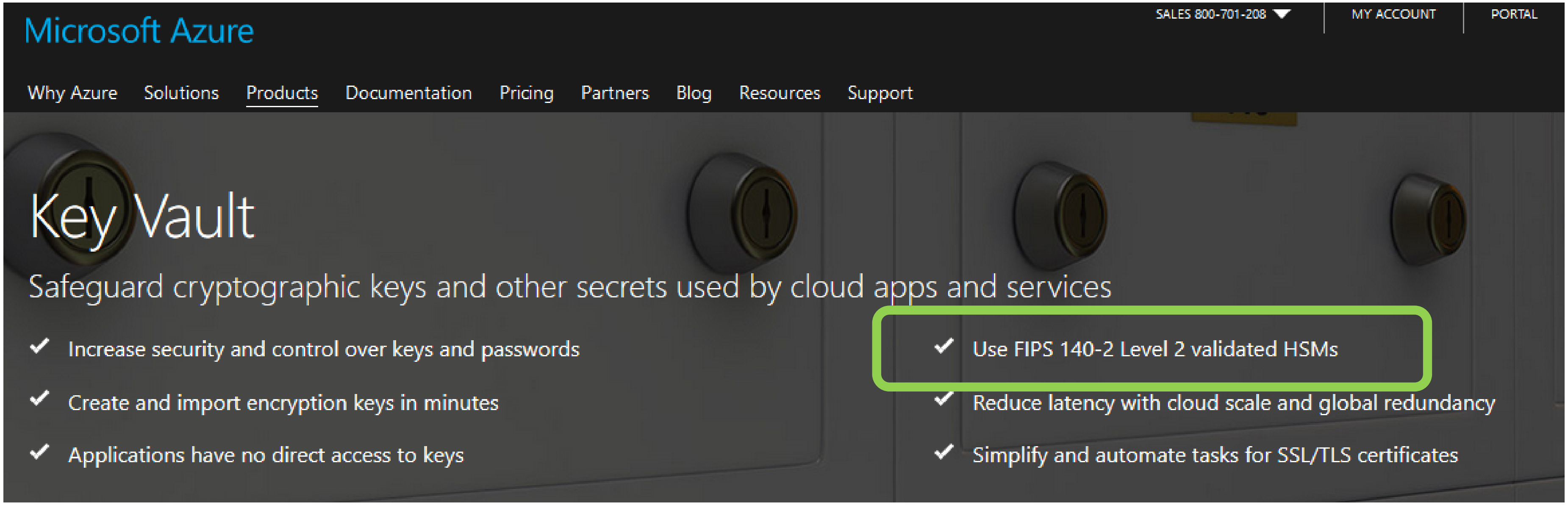
1. Transfer input data
2. Transfer wrapped key in
3. Initialize unwrap engine
4. Unwrap data/key (decrypt/verify)
5. Initialize key object with key value
6. Initialize cryptographic engine with key
7. Start, execute and finalize crypto operation
8. Initialize wrap engine
9. Wrap data/key (encrypt/sign)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

HSM IN CLOUD

Security topics in cloud environment

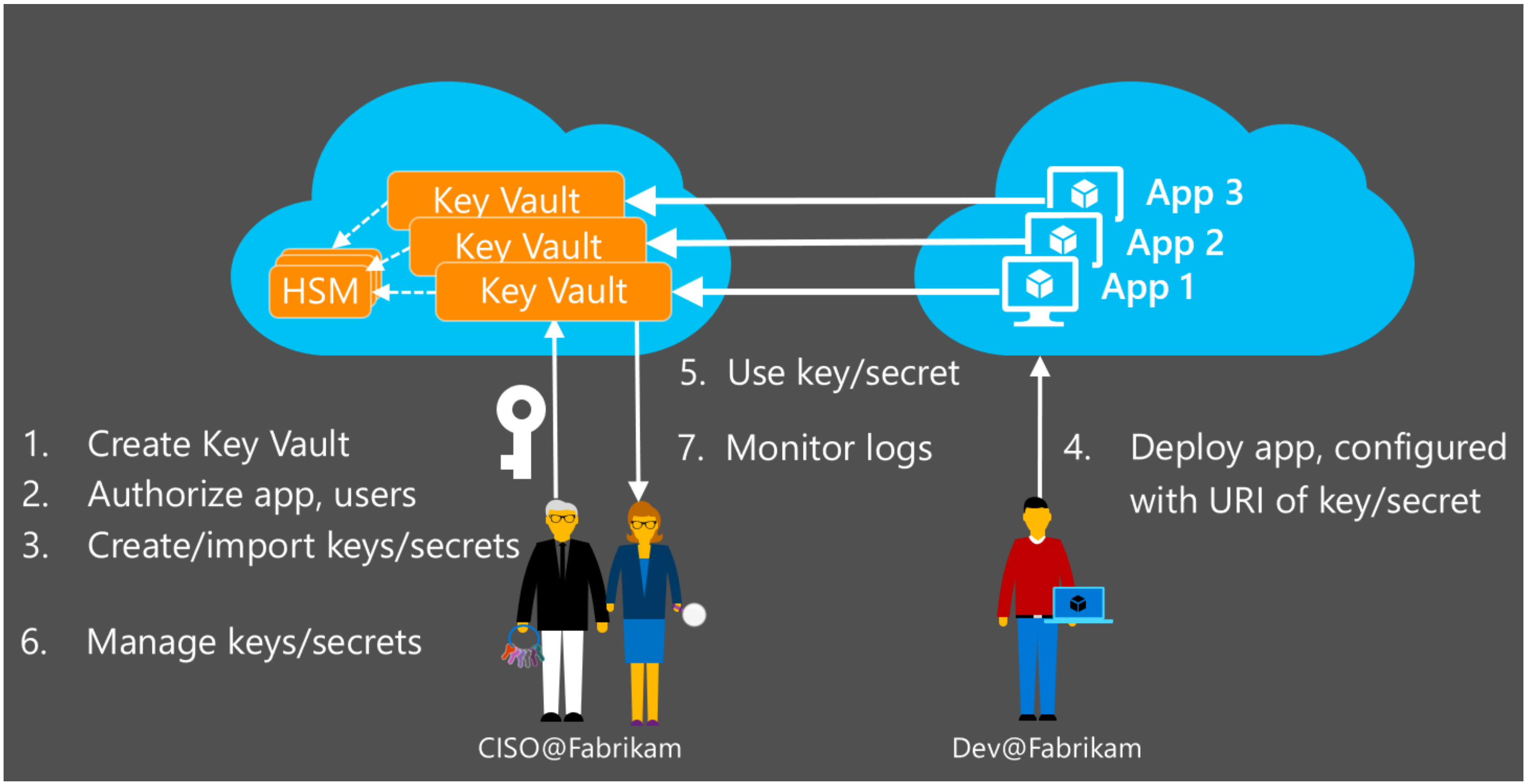
1. Move of legacy application into cloud
   – Previously used locally connected HSMs
2. Protection of messages exchanged between multiple cloud-based applications
   – Key exchange of used key without pre-distribution?
3. Volume encryption in cloud
   – Encrypted block mounted after application request (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

Use case: Microsoft Azure KeyVault



- 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



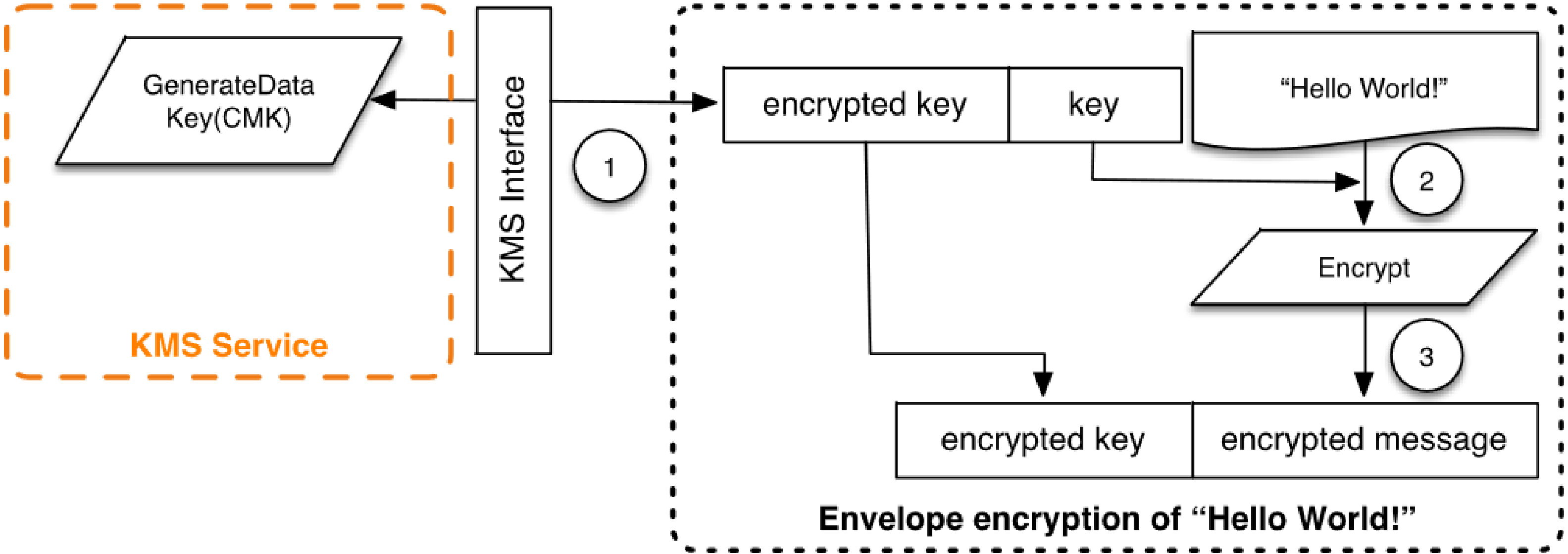
https://channel9.msdn.com/Events/Ignite/2015/BRK2706

Use case: AWS Key Management Service

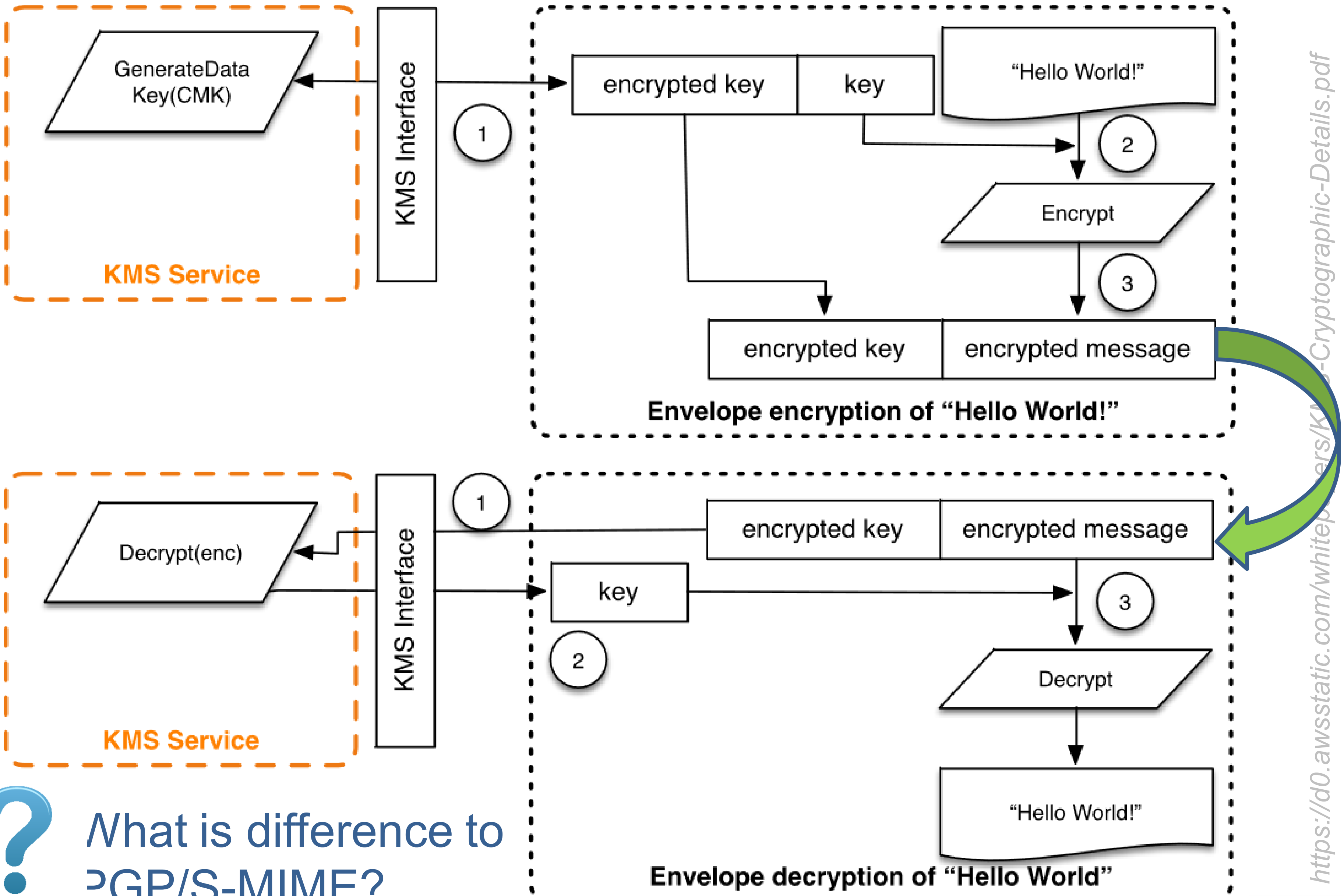
- AWS Key Management Service Cryptographic Details, M. Campagna (2015)
  - https://d0.awsstatic.com/whitepapers/KMS-Cryptographic-Details.pdf
- Centralized key management
  - Used by cloud-based applications
  - Used by any client application
  - Replication of wrapping keys into HSMs in different datacenters

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



What is difference to PGP/S-MIME?

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

Who is trusted?

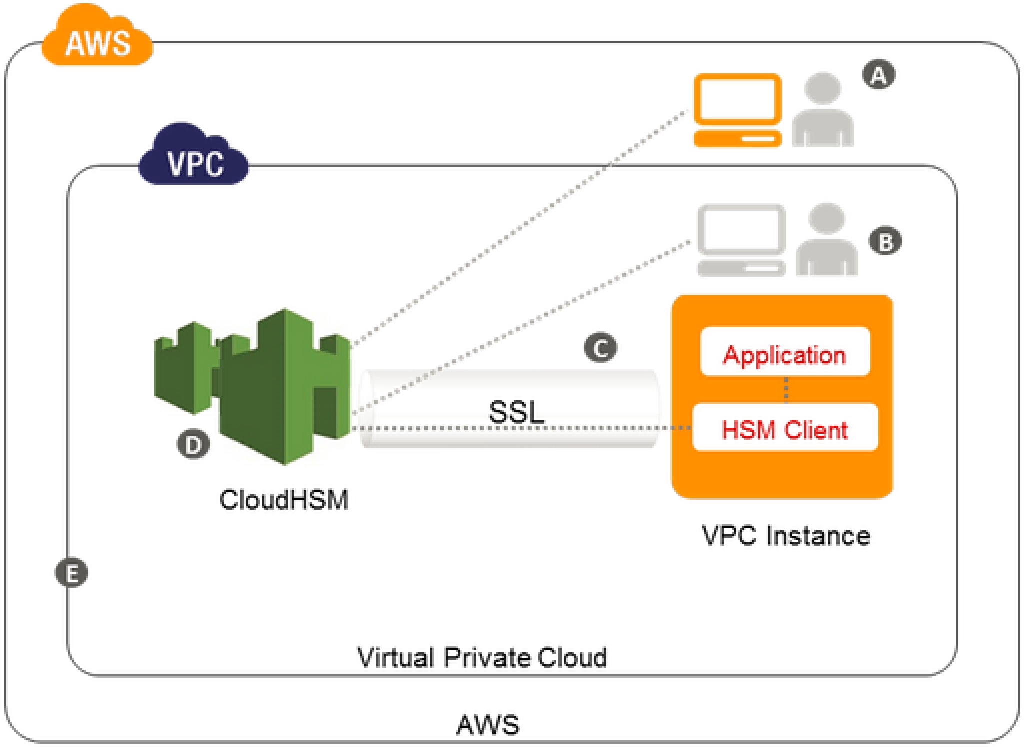
- KMS Service to wrap envelope keys properly
- KMS Service not to leak wrapping key
- Cloud operator not to read unwrapped keys from memory

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

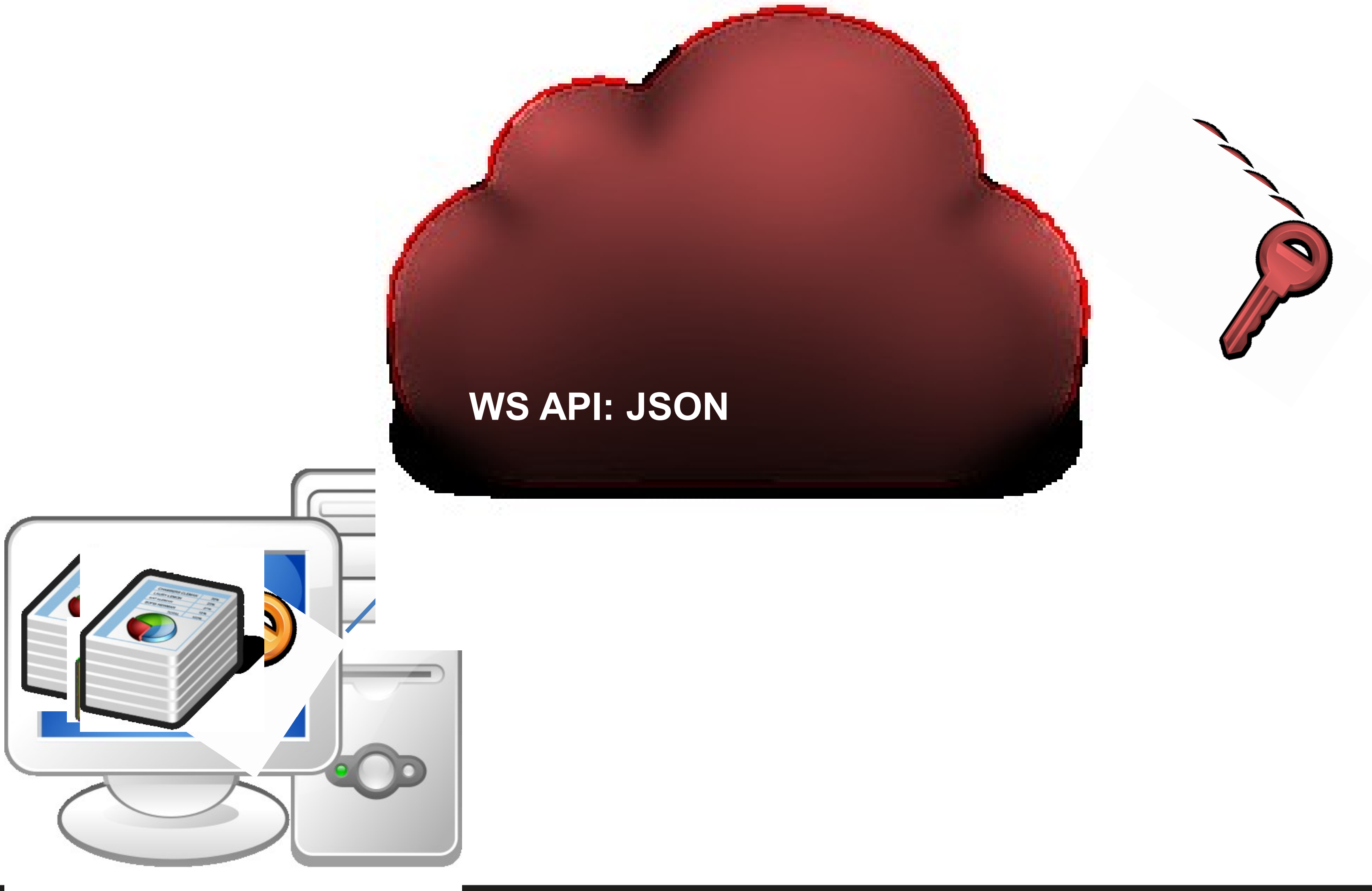
Use case: Amazon AWS CloudHSM

- (A) AWS manages the HSM appliance but does not have access to your keys
- (B) You control and manage your own keys
- (C) Application performance improves (due to close proximity with AWS workloads)
- (D) Secure key storage in tamper-resistant hardware available in multiple regions and AZs
- (E) CloudHSMs are in your VPC and isolated from other AWS networks

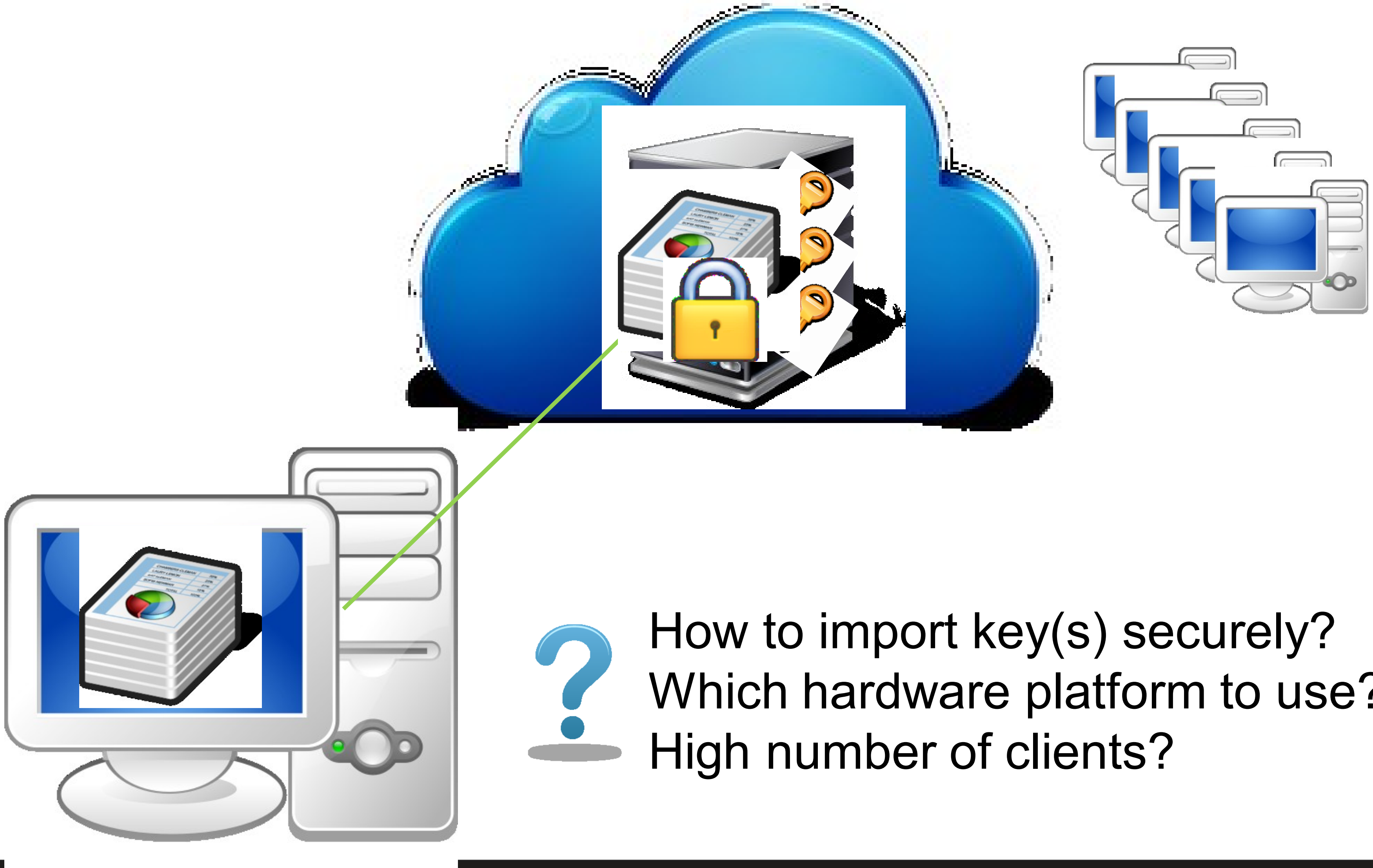


CRYPTOGRAPHY AS A SERVICE

Offloading s

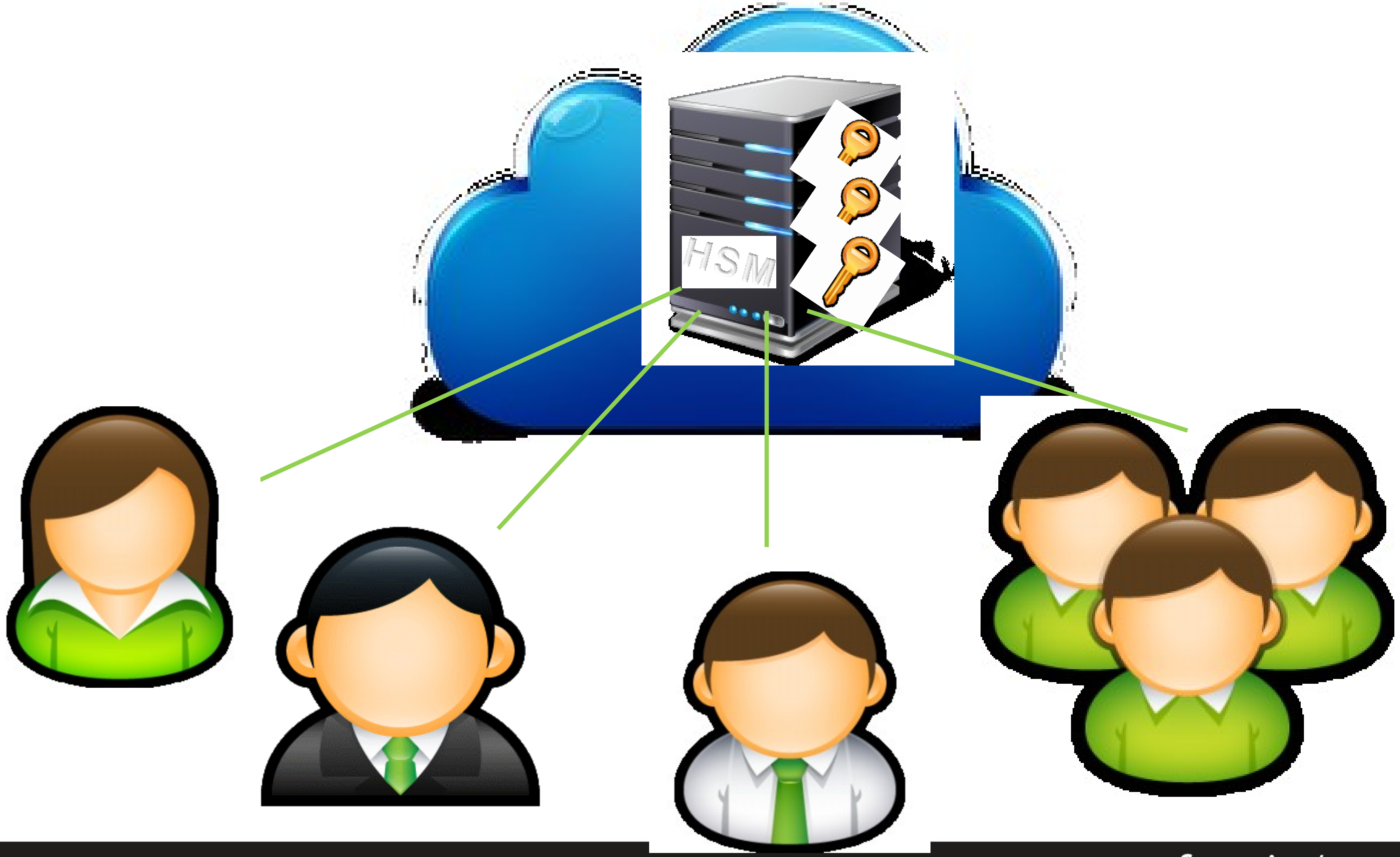


… into secured environment



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

Cryptography as a Service (CaaS)



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

CaaS - implementation issues

- Software-only CaaS more vulnerable to attacks
- Classic HSMs are not build for high-level of sharing
  - Performance degradation due to frequent context exchange
  - Logical separation only to few entities (16-32)
  - Physical separation on device-level
- If interested, read more at
  - Architecture Considerations for Massively Parallel Hardware Security Platform, D. Cvrcek, P. Svenda (2015) http://crcs.cz/papers/space2015

Hardware Security Module

HSM SECURITY API

Application Programming Interfaces (API)

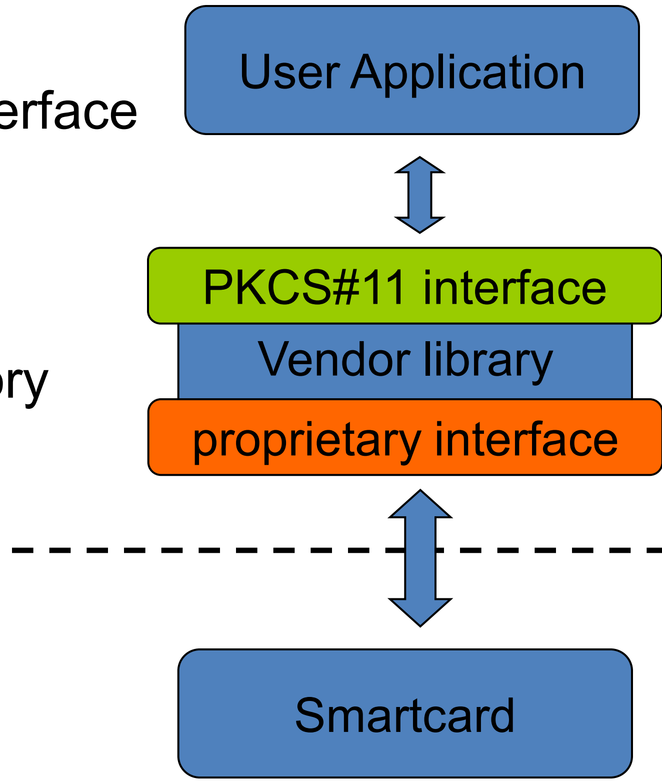
1. Proprietary API (legacy or custom functions)
2. Standardized API - but proprietary library required (PKCS#11)
3. Cryptographic service providers – plugin into standardized API (CNG, CSP…)
4. Standardized API - no proprietary component (PIV, EMV CAP…)
5. Proprietary (service-specific), but public API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

PKCS#11, (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

- 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)

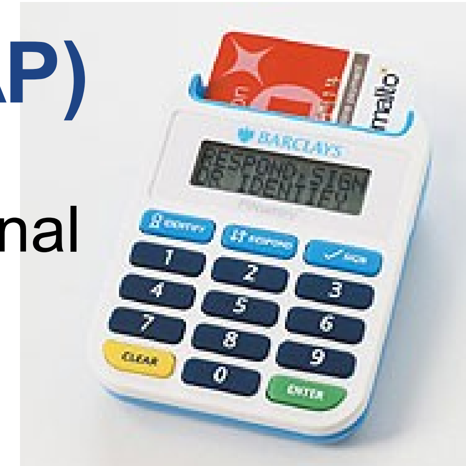
Microsoft CNG

- 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

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/

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..)
- Diverse APIs, potential logical attacks

PKCS#11 DETAILS