RSA CONFERENCE 2009

Encryption and Key Management Tutorials

Part II: PKCS #11 Enhancements and Opportunities

Robert Griffin
RSA, The Security Division of EMC
4/20/09 | Session ID: TUT-M51

Session Classification: Tutorial

Today's Agenda

9 – 10:45 am: Learning to Speak Crypto

11 am – 12:15 pm: PKCS #11 V2.30

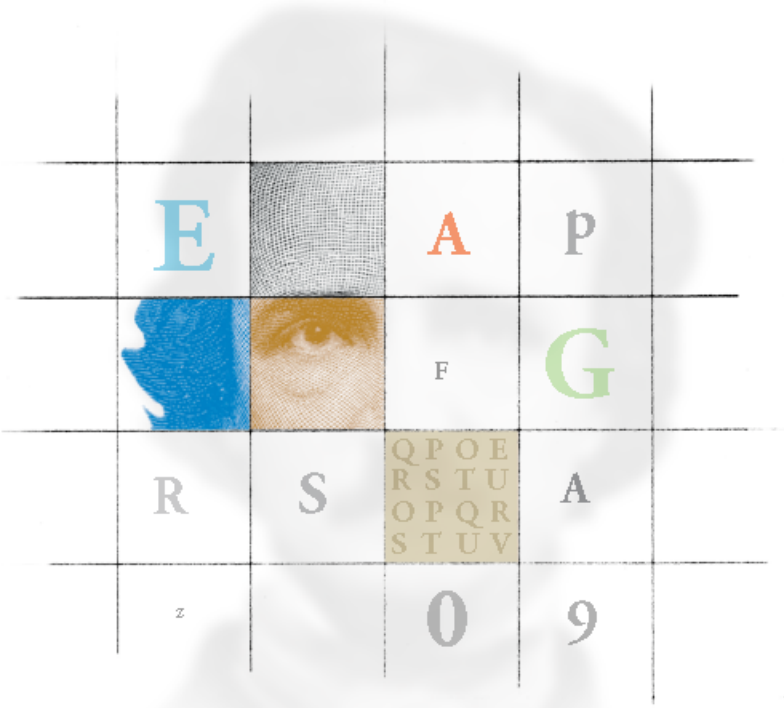
1 – 2:30 pm: Key Management

3:15- 5 pm: KMIP

Agenda for this Session

- Overview of PKCS #11
- PKCS #11 use cases
- PKCS #11 V2.30 enhancements
- PKCS #11 V2.30 use cases

Overview of PKCS #11



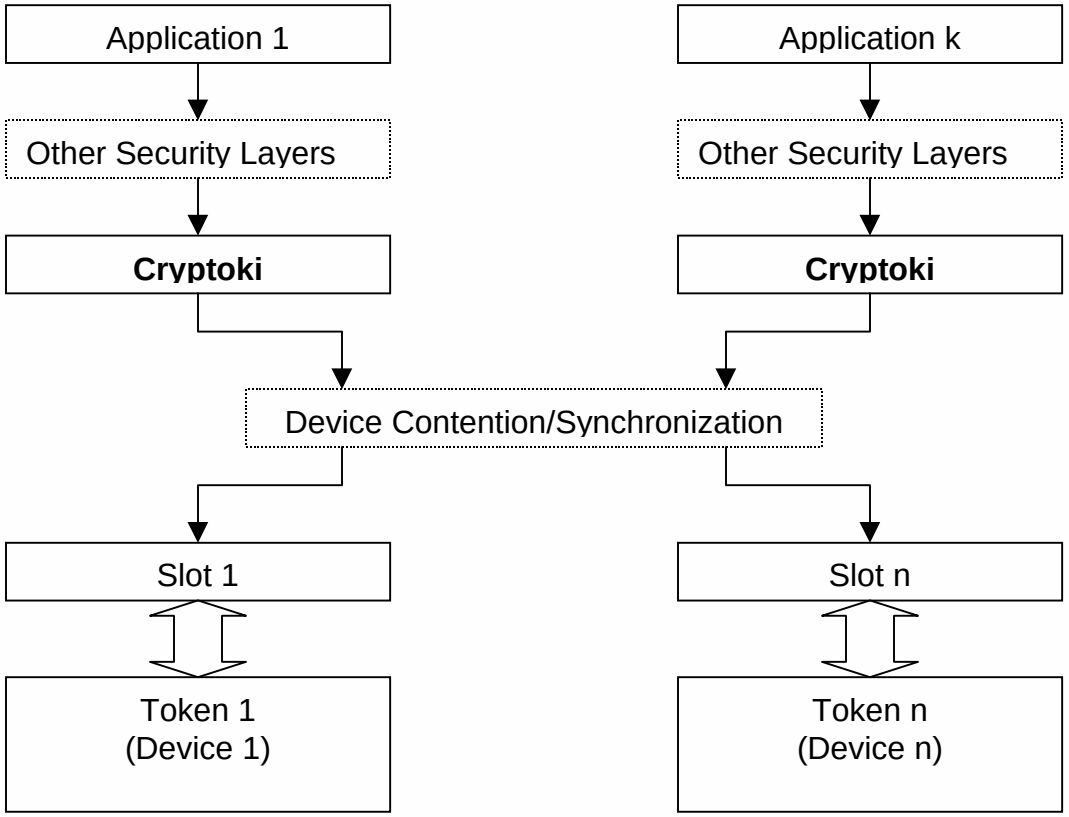
PKCS #11 Standard

- PKCS #11 is a widely-accepted standard for interfacing with devices that store keys (e.g. Hardware Security Module, smartcard)
  - Specifies application programming interface (“Cryptoki”) in C
  - Does not specify storage format
- History
  - 1/94: project launched
  - 4/95: v1.0 published
  - 12/97: v2.01 published
  - 12/99: v2.10 published
  - 6/04: v2.20 published
  - 12/05: amendments 1 & 2 (one-time password tokens, CTKIP)
  - 1/07: amendment 3 (additional mechanisms)
- PKCS #11 home
  - http://www.rsa.com/rsalabs/node.asp?id=2133

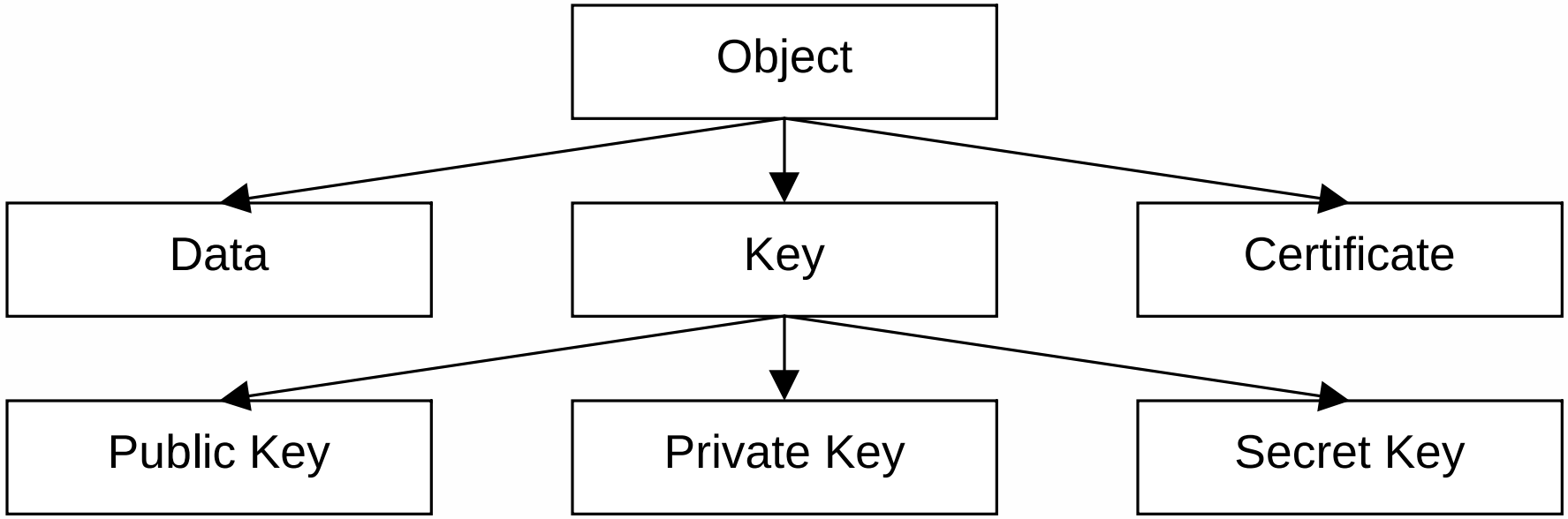
PKCS#11 Core Concepts

- Vendor neutral, cross-platform, industry standard
  - Security Object Life cycle
  - Security Attributes for all Objects
  - Secure by Design
  - Cryptographic Services
  - Extensible architecture
  - Arbitrary Objects
  - Arbitrary Attributes
- Supports wide range of devices
  - simple tokens
  - complex hardware security modules

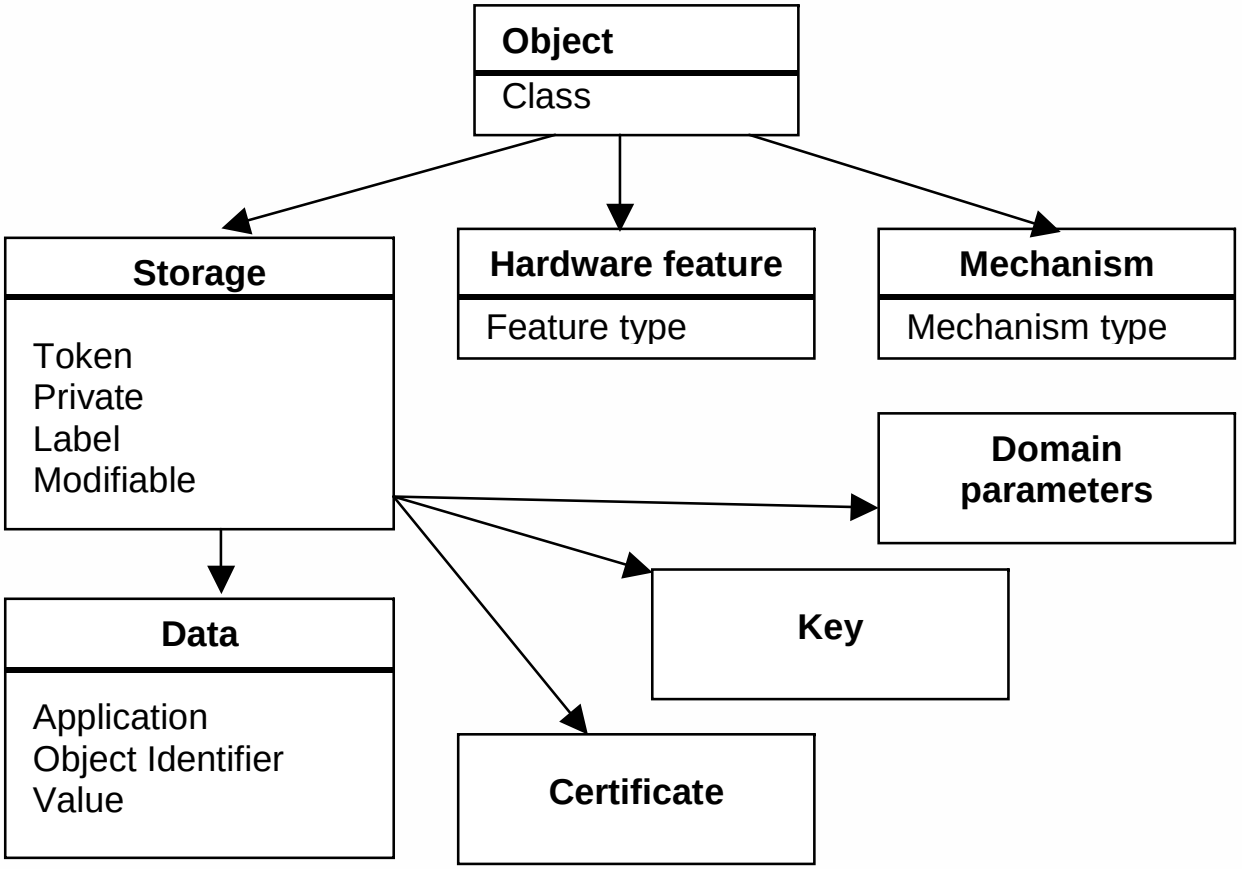
General Cryptoki Model *



PKCS #11 Object Hierarchy *



Object Attribute Hierarchy *



Security Object Life Cycle Services

- Create *object* on device
  - In volatile (session) or persistent (token) form
  - With security attributes (to protect migration or usage of object)
- Destroy *object* on device
- Import *object* into the device
  - Via secure (wrapped) or insecure (plaintext) approach
- Export *object* from the device
  - Via secure (wrapped) or insecure (plaintext) approach
- Perform operation on *object* on device (cryptographic services)
- Locate *object* on device
  - The search criteria is specified in terms of attribute values
- Set attributes against an *object* on device
- Get attributes from an *object* on device

Security Object Allowed Operations

- CKA_ENCRYPT
  - CK_TRUE if the security object supports encryption
- CKA_DECRYPT
  - CK_TRUE if the security object supports decryption
- CKA_SIGN
  - CK_TRUE if the security object supports signing
- CKA_VERIFY
  - CK_TRUE if the security object supports verification where the signature is an appendix to the data
- CKA_VERIFY_RECOVER
  - CK_TRUE if the security object supports verification where the data is recovered from the signature
- CKA_DERIVE
  - CK_TRUE if the security object key supports key derivation (*i.e.*, if other keys can be derived from this one)
- CKA_ALLOWED_MECHANISMS
  - A list of mechanisms allowed to be used with this security object

Security Object Basic Permissions

- CKA_TOKEN
  - CK_TRUE if security object is a token object
  - CK_FALSE if security object is a session object
- CKA_MODIFIABLE
  - CK_TRUE if object can be modified
- CKA_SENSITIVE
  - Security sensitive attributes non-readable
- CKA_PRIVATE
  - Authentication required prior to security object being *visible*
- CKA_TRUSTED
  - For certificates – can be trusted for verification
  - For keys – can be used as the wrapping key for *wrap-trusted* operations
- CKA_LOCAL
  - Security object was created on the device (not imported)

Security Object Import/Export

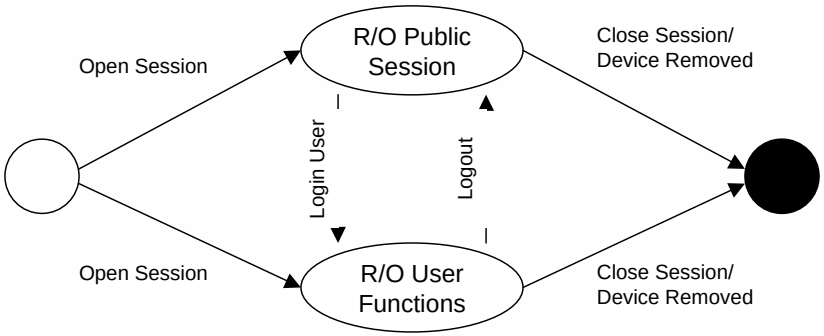
- CKA_WRAP
  - CK_TRUE if the security object supports wrapping (*i.e.*, can be used to wrap other security objects)
- CKA_WRAP_WITH_TRUSTED
  - CK_TRUE if the security object can only be wrapped with a wrapping security object that has CKA_TRUSTED set to CK_TRUE
- CKA_UNWRAP
  - CK_TRUE if the security object supports unwrapping (*i.e.*, can be used to unwrap other security objects)
- CKA_EXTRACTABLE
  - CK_TRUE if the security object is extractable and can be wrapped
- CKA_WRAP_TEMPLATE
  - The attribute template to match against any security objects wrapped using this wrapping security object. Security objects that do not match cannot be wrapped
- CKA_UNWRAP_TEMPLATE
  - The attribute template to apply to any security objects unwrapped using this wrapping security object. Any user supplied template is applied after this template as if the object has already been created.

Security Object Historical State

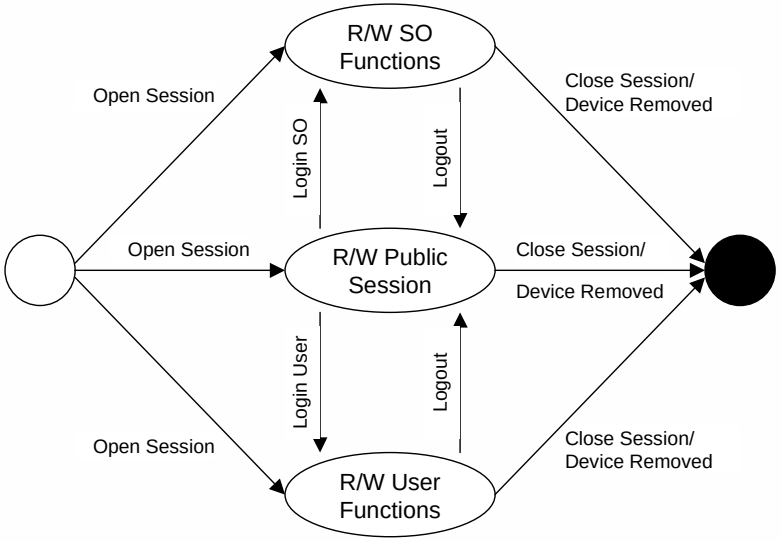
- CKA_ALWAYS_SENSITIVE
  - CK_TRUE if the security object has *always* had the CKA_SENSITIVE attribute set to CK_TRUE
- CKA_NEVER_EXTRACTABLE
  - CK_TRUE if the security object has *never* had the CKA_EXTRACTABLE attribute set to CK_TRUE
- CKA_START_DATE
  - Start date for the security object (day/month/year)
- CKA_END_DATE
  - End date for the security object (day/month/year)

Session States

*



Read-Only Session States



Read-Write Session States

Session Events

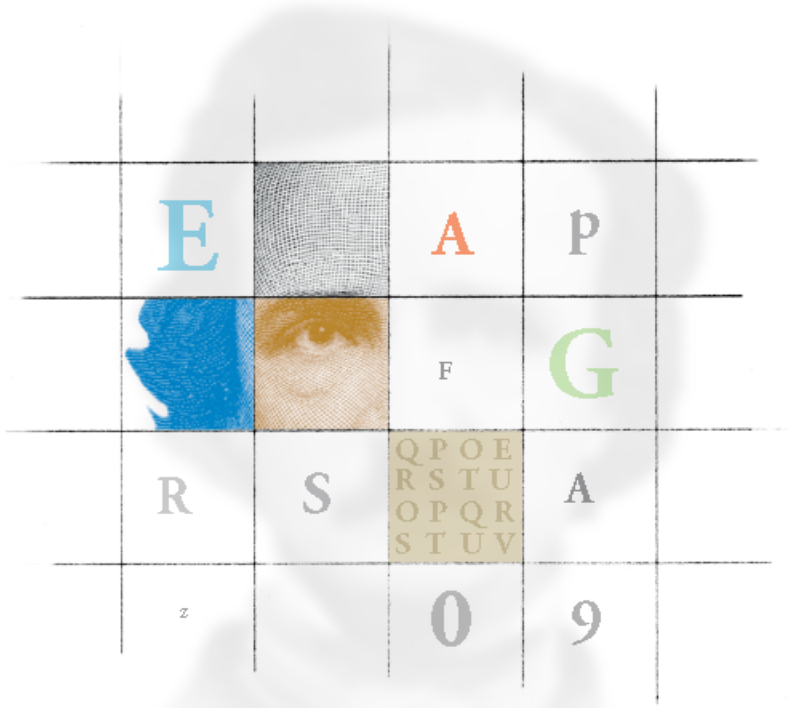
*

Event	Occurs when...
Log In SO	the SO is authenticated to the token.
Log In User	the normal user is authenticated to the token.
Log Out	the application logs out the current user (SO or normal user).
Close Session	the application closes the session or closes all sessions.
Device Removed	the device underlying the token has been removed from its slot.

PKCS Mechanisms

- A mechanism specifies precisely how a certain cryptographic process is to be performed.
  - For example, **CKM_RSA_PKCS_KEY_PAIR_GEN** is a key pair generation mechanism based on the RSA public-key cryptosystem, as defined in PKCS #1. It does not support encrypt/decrypt, sign/verify, signRecovery/verifyRecover, digest, wrap/unwrap or derive key.
- For a particular token, a particular operation may support only a subset of the mechanisms listed.
  - For example, CKM_SECURID_KEY_GEN, defined in Amendment 1 to V2.20, generates RSA SecurID keys with a particular set of attributes as specified in the template for the key.
- There is no guarantee that a token which supports one mechanism for some operation supports any other mechanism for any other operation (or even supports that same mechanism for any other operation).
  - For example, even if a token is able to create RSA digital signatures with the **CKM_RSA_PKCS** mechanism, it may or may not be the case that the same token can also perform RSA encryption with **CKM_RSA_PKCS**.

PKCS #11 Use Cases

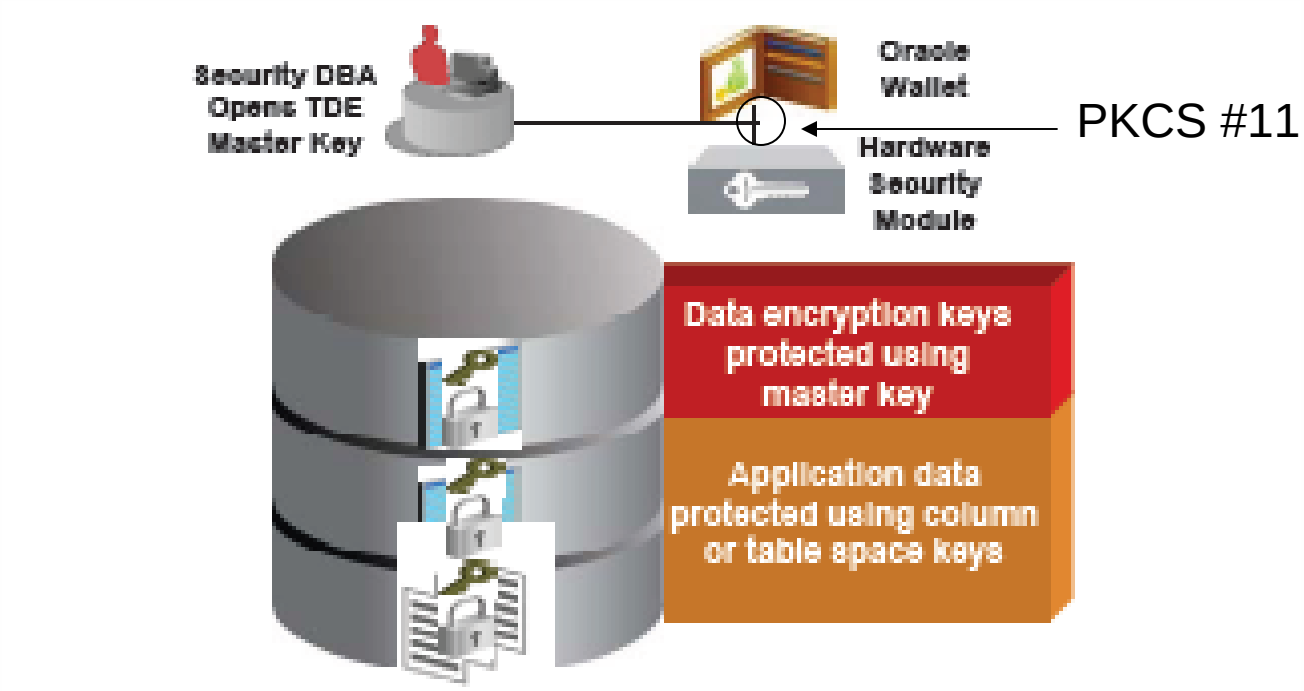


PKCS #11 Use Cases

- Hardware Security Module
- Smartcard Interface
- Certificate Distribution
- One-time Password

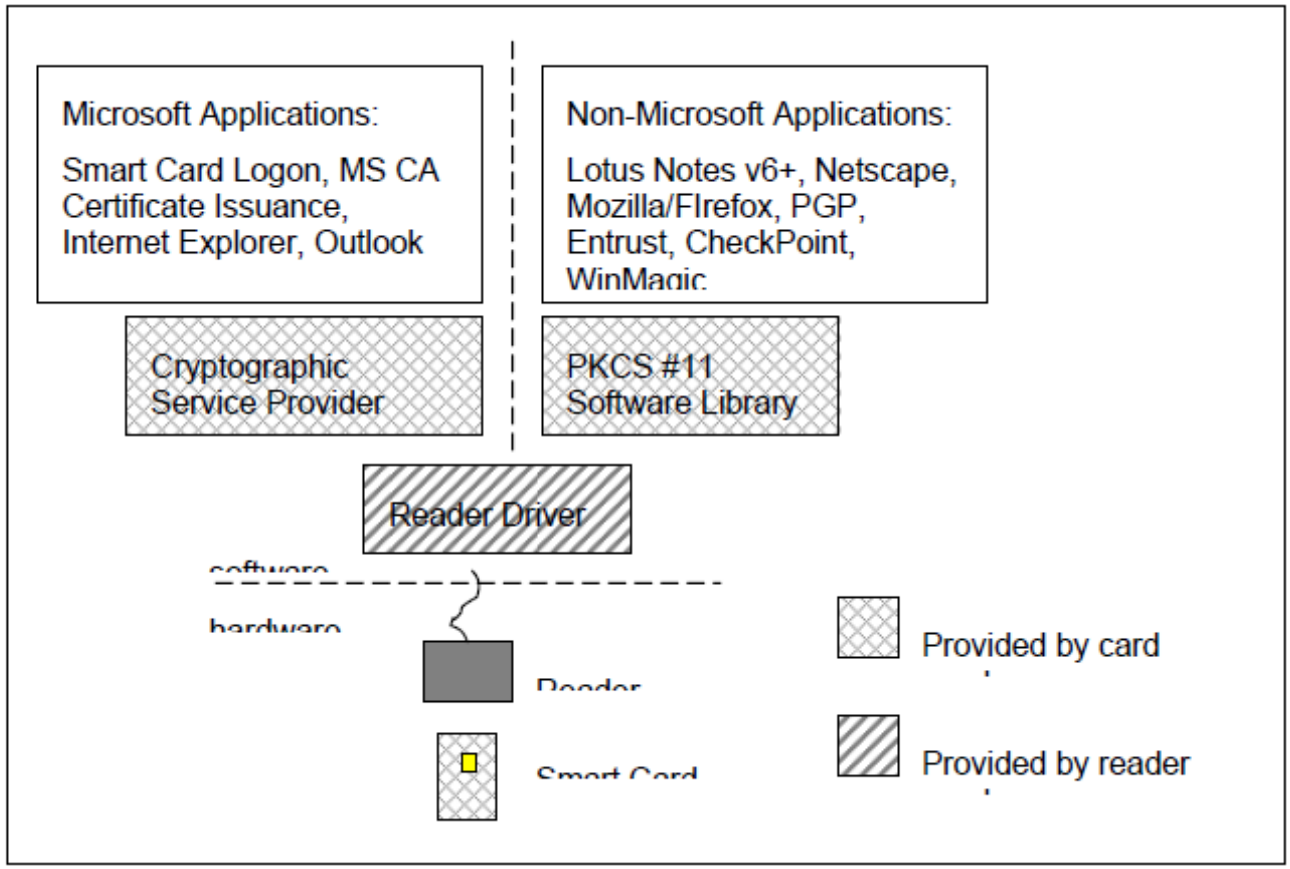
Oracle PKCS #11 Interface to HSM

Oracle Database 11g Transparent Data Encryption (TDE) communicates with the HSM device using the PKCS#11 interface.

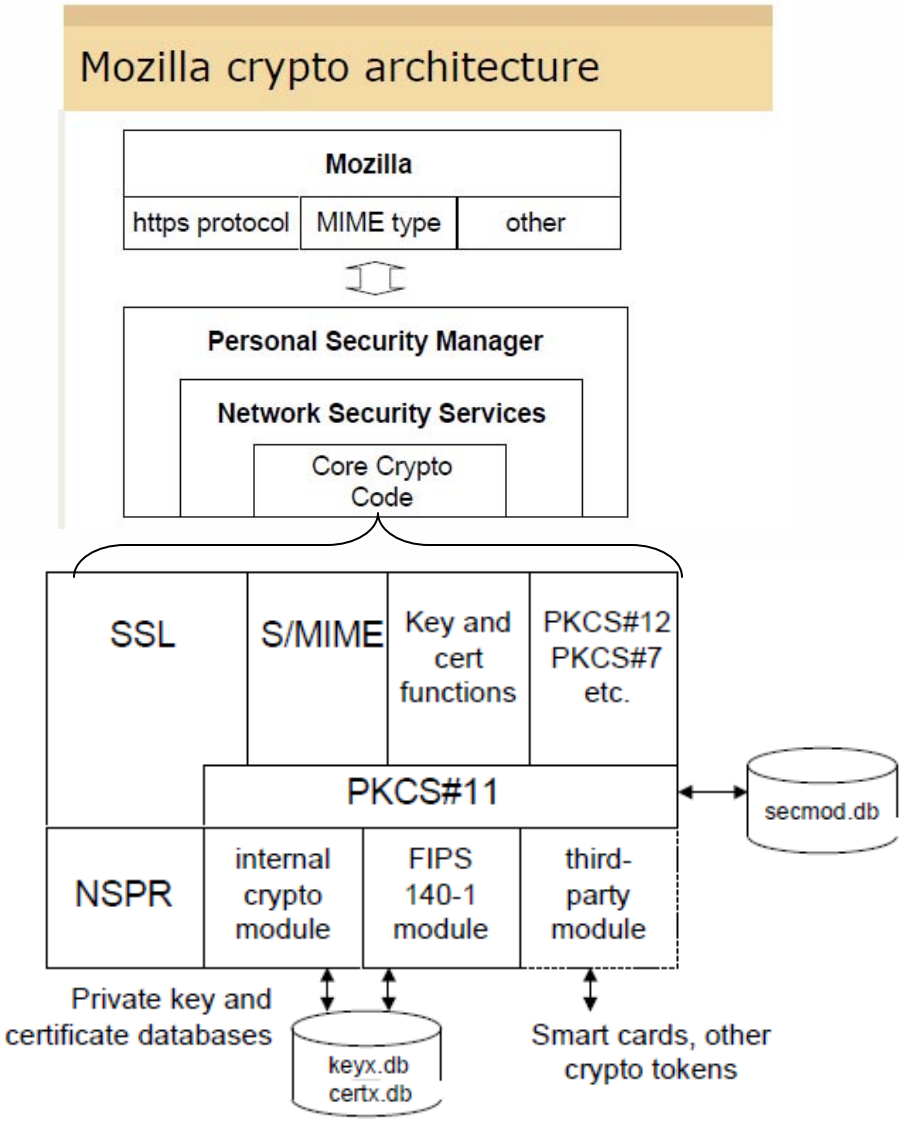


Smartcard Interface

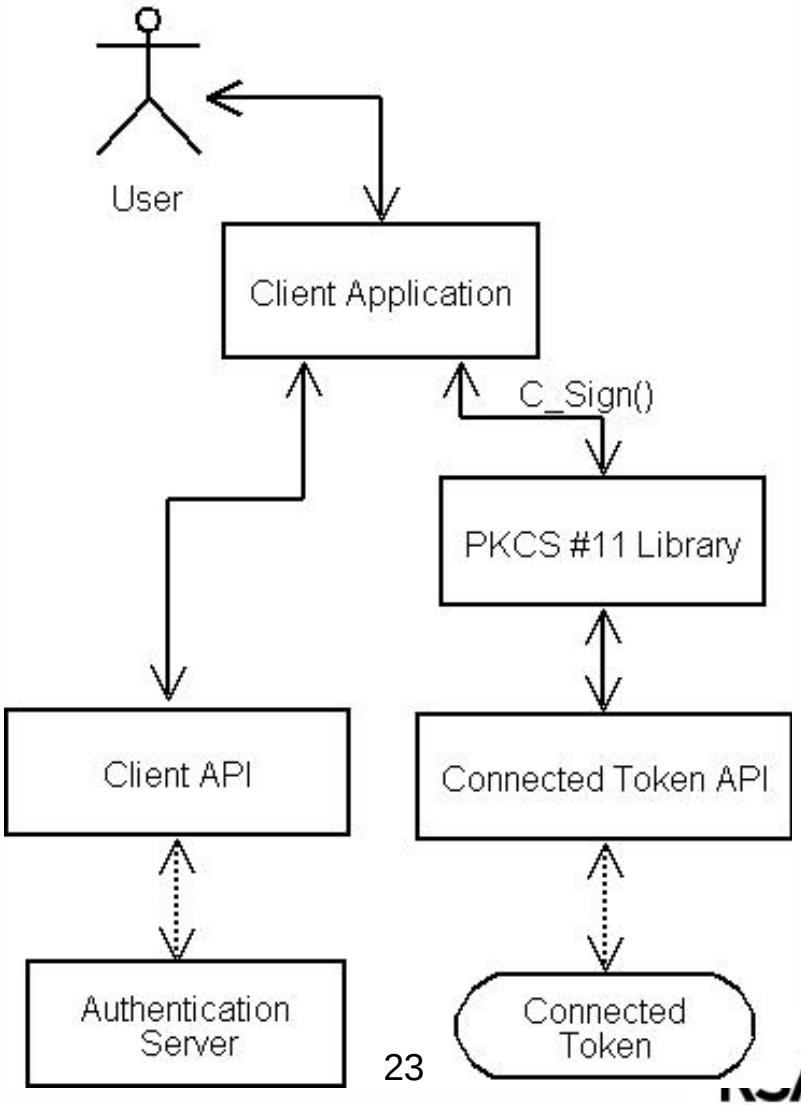
The two primary ways for applications to communicate with smart cards or other cryptographic tokens are via a PKCS#11 interface, or by a Cryptographic Service Provider (CSP).



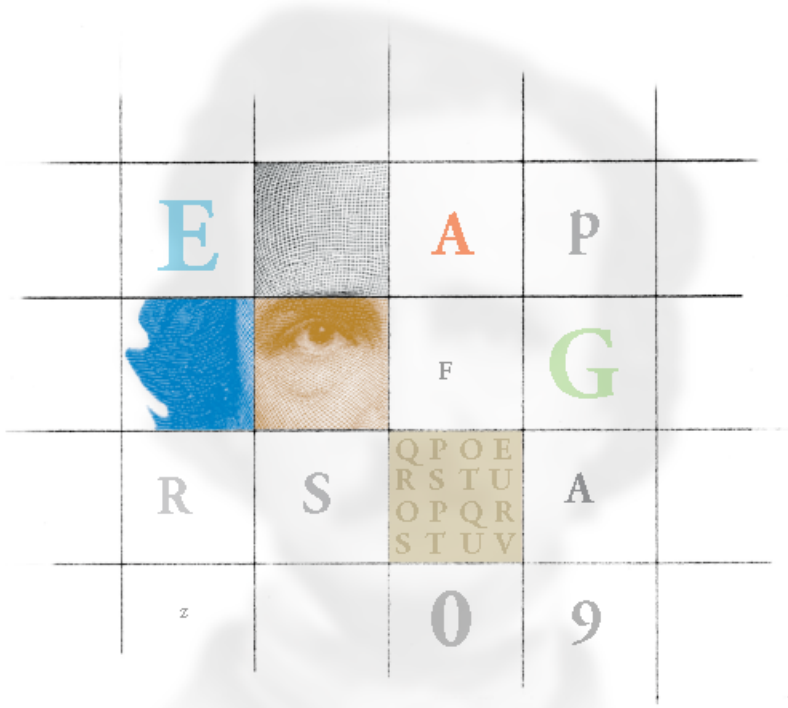
Certificate Distribution



One-Time Password



PKCS #11 v2.30 Enhancements



PKCS #11 V2.30 Enhancements

- New mechanisms
  - GOST
  - SEED
  - TPM
- Additional or enhanced mechanisms
  - *AES-CCM authenticated Encryption / Decryption*
  - *AES-GCM authenticated Encryption / Decryption*
  - *CKM_BLOWFISH_CBC_PAD*

GOST Mechanisms

- The GOST standards are maintained by the Euro-Asian Council for Standardization, Metrology and Certification (EASC).
  - Official web site at http://www.gost.ru/wps/portal/
  - English information at http://tools.ietf.org/html/rfc4357
- GOST 28147-89 is a block cipher with 64-bit block size and 256-bit keys.
  - GOST 28147-89 secret key objects (object class **CKO_SECRET_KEY,** key type **CKK_GOST28147**) hold GOST 28147-89 keys for key generation, encrypt, etc.
  - Additional objects support other GOST mechanisms.
- GOST R 34.11-94 is a mechanism for message digesting.
- GOST R 34.10-2001 is a mechanism for single- and multiple-part signatures and verification.

GOST Mechanisms - 2

Mechanism	Functions						
	Encrypt & Decrypt	Sign & Verify	SR & VR	Digest	Gen. Key/ Key Pair	Wrap & Unwrap	Derive
CKM_GOST28147_KEY_GEN					√		
CKM_ GOST28147_ECB	√					√	
CKM_GOST28147	√					√	
CKM_ GOST28147_MAC		√					
CKM_ GOST28147_KEY_WRAP						√	
CKM_GOST3411				√			
CKM_GOST3411_HMAC		√					
CKM_GOST3410_KEY_PAIR_GEN					√		
CKM_GOST3410		√[1]					
CKM_GOST3410_WITH_ GOST3411		√					
CKM_ GOST3410_KEY_WRAP						√	
CKM_ GOST3410_DERIVE							√

SEED Mechanisms

- SEED is a block cipher with 128-bit block size and 128-bit keys.
  - developed by the Korean Information Security Agency and first published in 1998.
  - Used broadly in South Korean industry.
- The new key type is CKK_SEED. It is a general block cipher key type with a fixed CKA_VALUE length of 16 bytes.

SEED Mechanisms - 2

Mechanism	Functions						
	Encrypt & Decrypt	Sign & Verify	SR & VR	Digest	Gen. Key/ Key Pair	Wrap & Unwrap	Derive
CKM_SEED_KEY_GEN					√		
CKM_SEED_ECB	√					√	
CKM_SEED_CBC	√					√	
CKM_SEED_CBC_PAD	√					√	
CKM_SEED_MAC		√					
CKM_SEED_MAC_GENERAL		√					
CKM_SEED_ECB_ENCRYPT_DATA							√
CKM_SEED_CBC_ENCRYPT_DATA							√

TPM Mechanisms

- New mechanisms proposed for V2.30 to support Trusted Platform Modules (TPM).
  - Support RSA encrypt/decrypt as specified in Trusted Computing Group (TCG)'s Trusted Platform Module specification.
  - Minor modification of PKCS#1 v 1.5 and PKCS#1-style OAEP to prepend TCG-specific header information to plaintext before padding.
- Proposal is to introduce two new mechanisms to automatically prepend / strip the header information
  - Could be supported using existing mechanisms by requiring the developer to process the header manually.
  - Mechanisms make interoperability with TPM more developer-friendly.

TPM Mechanisms - 2

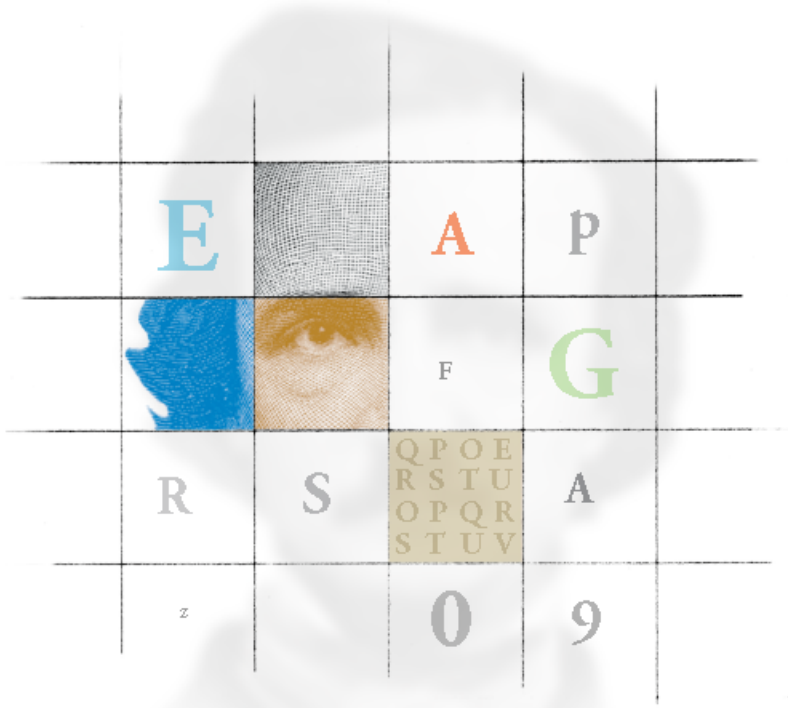
Mechanisms	Functions						
	Encrypt & Decrypt	**Sign & Verify**	**SR & VR[1]**	**Digest**	**Gen. Key/ Key Pair**	**Wrap & Unwrap**	**Derive**
CKM_RSA_PKCS_TPM_1_1	✓					✓	
CKM_RSA_OAEP_TPM_1_1	✓					✓	

Encrypt and decrypt support for single-part operations only.

Other Additional or Enhanced Mechanisms

- *CKM_AES_CCM*
  - Supports Counter with CBC Mac Mode (RFC 2610) for authenticated encrypt and decrypt processes.
  - Specified in RFC 3610.
  - For use in IPsec (RFC 4309), CMS (RFC 5084), etc.
- *CKM_AES_GCM*
  - Supports Galois Counter Mode for authenticated encrypt and decrypt processes.
  - For use in IPsec (RFC 4106), CMS (RFC 5084), TLS (RFC 5288) etc.
- CKM_BLOWFISH_CBC_PAD
  - Supports single- and multiple-part encryption and decryption, key wrapping and key unwrapping.
  - The PKCS padding in this mechanism allows the length of the plaintext value to be recovered from the ciphertext value.

PKCS #11 v2.30 Opportunities

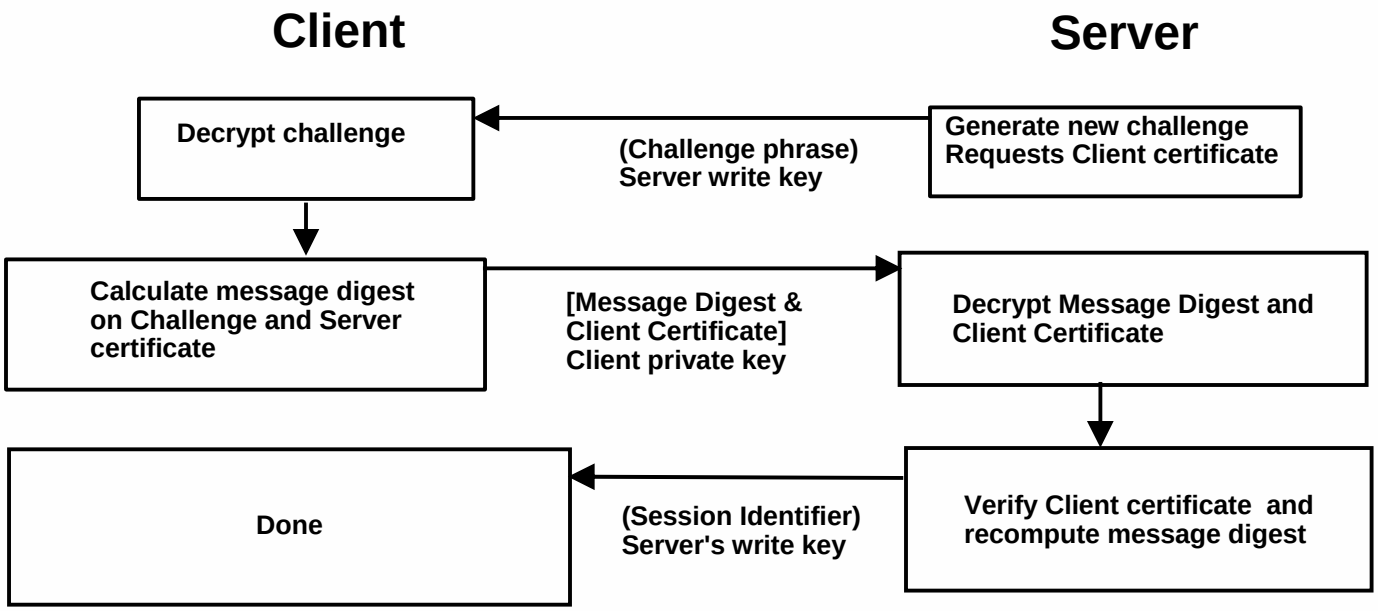


GOST

- Support PKCS #11 interfaces for disk encryption and other applications in the Commonwealth of Independent States (CIS).

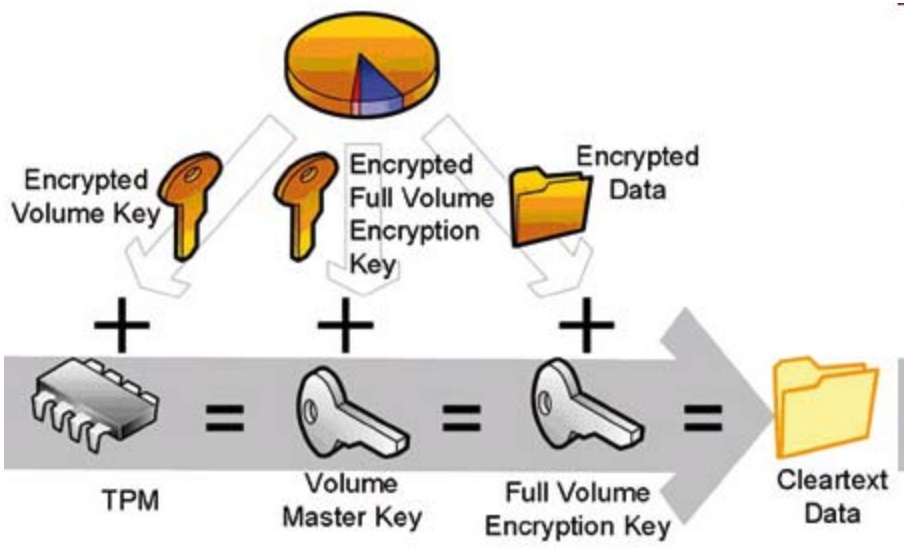
SEED Mechanism

Support PKCS #11 interfaces for browser security and other applications in the South Korean market.



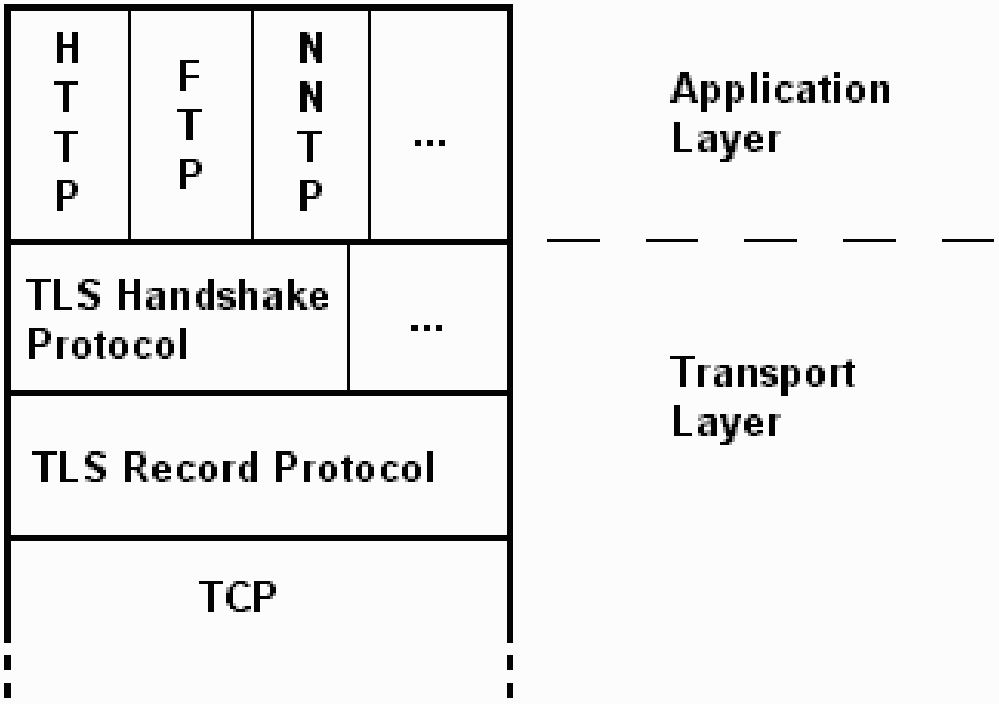
TPM Mechanism

- Allow encryption for TPM public key on non-TPM-enabled machine
- Allow greater ease of use of TPM functionality through PKCS#11 on TPM-enabled machine.



Authenticated Encryption

- TLS support through AES GCM and AES CCM mechanisms.



Questions?