On the Security of PKCS#11

Jolyon Clulow

School of Mathematical and Statistical Sciences

University of Natal, Durban, South Africa.

Jolyon.Clulow@cl.cam.ac.uk

Recent Results on APIs and Standards

- Specifically focussed on cryptographic hardware (HSM):
  - Bond, 2001
  - Bond & Anderson, 2001
  - Clayton & Bond, 2002
  - Clulow, 2002
  - Bond & Zielinski, 2002
- Non HSM focussed:
  - Bleichenbacher, Manger, Klima & Rosa,…

Motivation

- Analyse the security of PKCS #11 as an interface for a security device.
- Provide a comprehensive reference of the known security issues (pitfalls) for standards and APIs.
  - Evaluation
  - Future developments
- Evaluate the response of standards bodies and API designers to published vulnerabilities.
  - Transfer of knowledge of published (academic) research to industry.

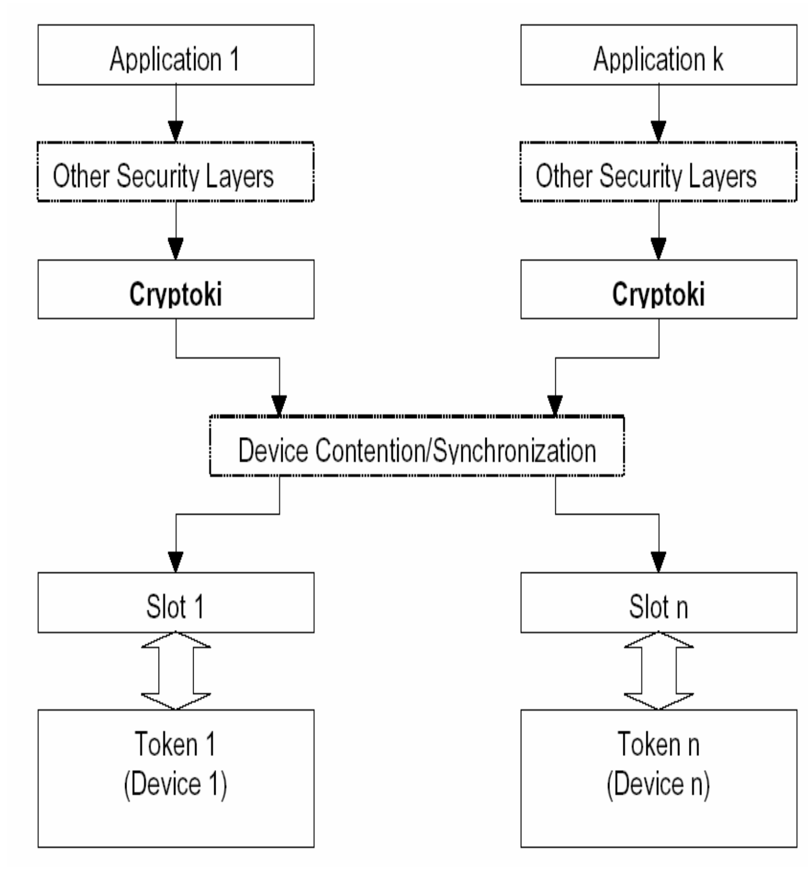
Public Key Cryptography Standard (PKCS)

- Developed by RSA Labs in cooperation with representatives of industry, academia and government.
- Many important, existing APIs and protocols have been built upon PKCS#11 (e.g. SSL). Notable products include Mozilla and SSL hardware accelerators from nCipher, IBM, Thales, Rainbow, AEP/Baltimore, etc.

Design Goals of PKCS#11

- Provide a standard interface between applications and (portable) cryptographic devices to allow interoperability and compatibility between vendor devices and implementations.
- Allow resource sharing (a many-to-many relationship between applications and devices).
- Not intended to be a general interface to cryptographic operations or security services but could be used to build such services, operations or suitable APIs.

Architecture



- A “*token*” is a device that stores objects (e.g. Keys, Data and Certificates) and performs cryptographic operations.
- A logical rather than physical characterization - one device may have several, distinct logical tokens (e.g. like domains)

Stated Security Target

- Access to private objects on the token, … , requires a PIN. Thus, possessing the cryptographic device that implements the token may not be sufficient to use it; the PIN may also be needed.
- Does not appear to be the intention to prevent one user from using another user’s private objects.
- Additional protection can be given to private keys and secret keys by marking them as “sensitive” or “unextractable”.
  - Sensitive keys cannot be revealed in plaintext off the token, and unextractable keys cannot be revealed off the token even when encrypted (though they can still be used as keys).

Stated Security Concerns

- Areas of concerns
  - Operating system security, rogue applications, linked libraries, device drivers
  - Sniffing communication lines to cryptographic device
- Possible compromises
  - PIN recovery
  - Access to session (insertion, modification or deletion of commands)
  - Impersonation of token/device
- Solutions
  - Code signing

The Security Claim

"We note that none of the attacks just described can compromise keys marked sensitive, since a key that is sensitive will always remain sensitive. Similarly, a key that is unextractable cannot be modified to be extractable."

C_WrapKey

C_WrapKey wraps (*i.e.*, encrypts) a private or secret key and can be used in the following situations:

- To wrap any secret key with an RSA public key.
- To wrap any secret key with any other secret key.
- To wrap an RSA, Diffie-Hellman, or DSA private key with any secret key.

C_WrapKey Secret Key Mechanisms

CKM_<NAME>_<MODE>

Some mechanisms:

- CKM_DES_ECB
- CKM_DES_CBC
- CKM_DES_CBC_PAD
- CKM_DES3_ECB
- CKM_DES3_CBC
- CKM_DES3_CBC_PAD

Other ciphers include:

- RC2, RC4, RC5, CAST, IDEA, etc

C_WrapKey Vulnerabilities

- Weaker Key
- Weaker Algorithm
- Key Conjuring
- Key Binding
- Key Separation

Key Conjuring

- Observations:
  - Optional use of either ECB or CBC mode.
  - Optional use of a MAC.
- Implications:
  - The unauthorized generation of keys.
  - e.g.

    Import a random byte string (R) as the encrypted key (T_{random}) yielding a new key $k_{random} = d_{MK}(T_{random})$.

Key Binding

- Observations:
  - Optional use of either ECB or CBC mode.
  - Optional use of a MAC.
  - No restrictions on Keys with repeated halves.
- Implications
  - Can attack each half of a key(component) independently.
  - e.g.
    - Export a double length key (any mode).
    - Re-import the first half as a single length key encrypted under ECB.
    - Re-import the second half as a single length key encrypted under ECB.
    - Perform key search against each single length individually.

Key Separation

Attribute	**Meaning**
CKA_ENCRYPT	TRUE if key supports encryption
CKA_DECRYPT	TRUE if key supports decryption
CKA_SIGN	TRUE if key supports signatures (*i.e.*, authentication codes)
CKA_VERIFY	TRUE if key supports verification (*i.e.*, of authentication codes)
CKA_WRAP	TRUE if key supports wrapping
CKA_UNWRAP	TRUE if key supports unwrapping

Key Separation

- Observation:
  - No enforced separation between Encryption, Authentication and Key Wrapping Keys.
- Implications:
  - e.g.
    - Export the target key (k1) under any key (k2 – the key wrapping key) using the function C_WrapKey.
    - Decrypt the resultant data using C_Decrypt using k2 (the key wrapping key) as a data decryption key.
    - The data returned is the clear value of the target key (i.e. k1).

What about the claim again?

“We note that none of the attacks just described can compromise keys marked “sensitive,” since a key that is sensitive will always remain sensitive. Similarly, a key that is unextractable cannot be modified to be extractable.”

- Previous attacks require a key to be extractable.
- This property cannot be disabled (and once set cannot be cleared).
- What about other attacks?

C_DeriveKey Mechanisms

1. **CKM_CONCATENATE_BASE_AND_KEY**
   - derives a secret key from the concatenation of two existing secret keys
2. **CKM_CONCATENATE_BASE_AND_DATA**
   - derives a secret key by concatenating data onto the end of a specified secret key.
3. **CKM_CONCATENATE_DATA_AND_BASE**
   - derives a secret key by prepending data to the start of a specified secret key.

C_DeriveKey Mechanisms

4. **CKM_XOR_BASE_AND_DATA**
   - is a mechanism which provides the capability of deriving a secret key by performing a bit XORing of a key pointed to by a base key handle and some data.
5. **CKM_EXTRACT_KEY_FROM_KEY**
   - provides the capability of creating one secret key from the bits of another secret key.

C_DeriveKey Vulnerabilities

- Reduced Key Space
- Parallel Key Search using Related Keys
- Related Keys Attack
- Combined (Parallel Related Key Attack)

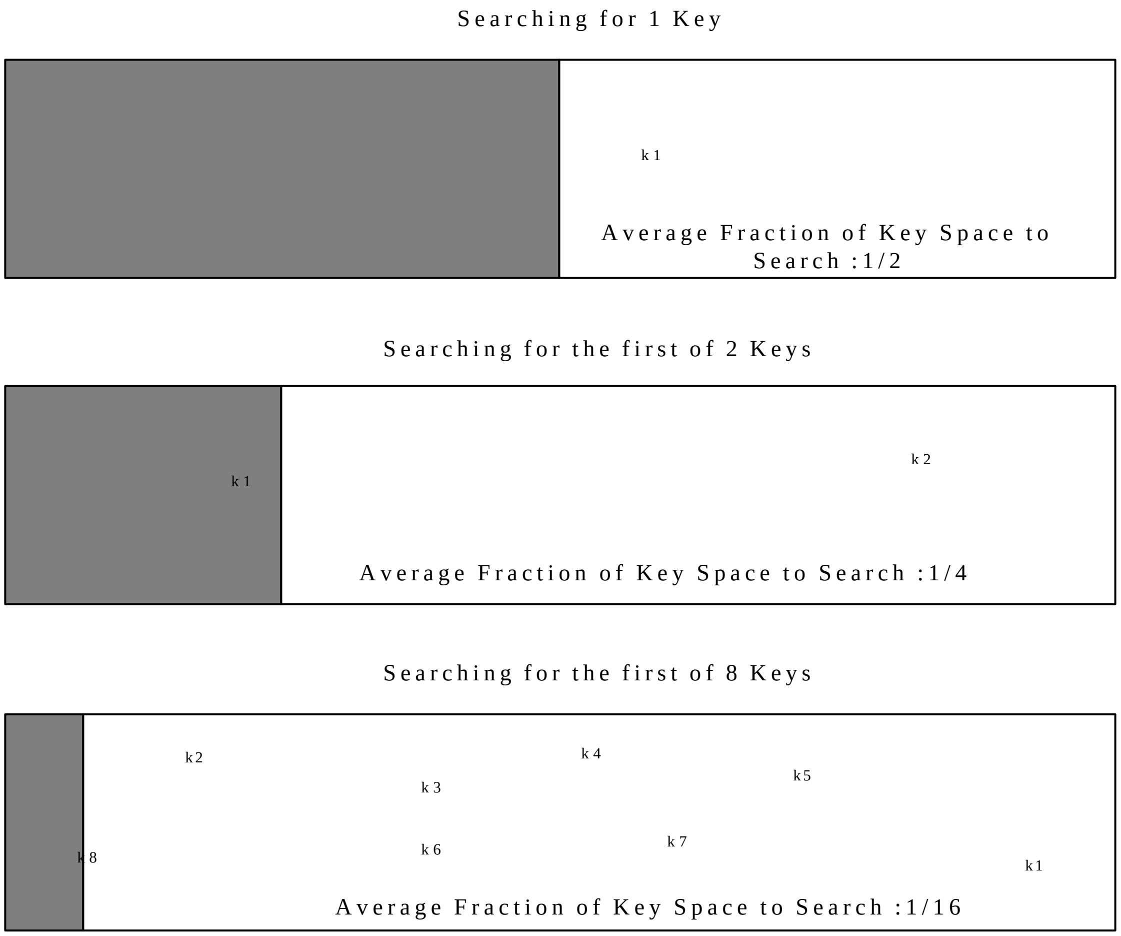
Reduced Key Space

- Observation:
  - Using the CKM_EXTRACT_KEY_FROM_KEY mechanism, one can extract a subset of the bits from a given key to create a shorter key.
- Implications:
  - Reduces the key space needed to be searched.
  - e.g.
    - Extract 40 bits from a DES key to create a 40 bit RC2 key.
    - Exhaustively search the 40 bit RC2 key.
      - May actually less due to parity bits in the DES key
    - Exhaustively search for the remaining 24 bits (less 3 parity bits).

Parallel Key Search

- Observation:
  - CKM_XOR_BASE_AND_DATA gives us an easy way to xor known patterns onto a key.
  - Reduce key space we need to search by generating a large number of related keys (with known differences).
- Implications:
  - Single length DES keys can be easily found.

Parallel Key Search



Searching for 1 Key
k 1
Average Fraction of Key Space to Search :1/2
Searching for the first of 2 Keys
k 2
k 1
Average Fraction of Key Space to Search :1/4
Searching for the first of 8 Keys
k 2
k 4
k 5
k 3
k 7
k 6
k 8
k 1
Average Fraction of Key Space to Search :1/16

Parallel Key Search (cont.)

- e.g.
  - Generate 2^16 related keys of original target key.
  - Using each key, encrypt a known pattern and store result in searchable database.
  - Search for a key by iteratively performing trial encryptions of the known pattern and compare result to entries in database.
  - After 2^39 we expect to find a match (i.e. we find a key which produces an encrypted output in the database).

  Since we know how that key is related to all the others, we known all the 2^16 keys including the original one.
- What went wrong?
  - Being able to modify a key is dangerous.
  - Lack of appreciation of implications.

Related Keys Attack

- Observation:
  - Using CKM_XOR_BASE_AND_DATA can create related keys.
- Implications:
  - Reduces 3-key 3DES to only slight stronger than single DES (2^168 to 2^56).
  - Works by isolating the key components and searching for them independently.
  - 2-key 3DES can similarly be attacked by first ‘converting them into 3-key 3DES keys (using CKM_CONCATENATE_BASE_AND_DATA).

Related Keys Attack Explained

- The 3DES Attack
  - Start with related keys
    - K1 = <k1, k2, k3>
    - K2 = <k1 $\oplus$ Δ, k2, k3>

    So K1 and K2 are the same except for some small difference Δ.
  - Encrypt plaintext (P) under K1 to get cipher text (C)
    - $C = e_{K1}(P) = e_{K1}(d_{k2}(e_{K3}(P)))$
  - Decrypt ciphertext (C) under K2 to get new 'plaintext' (P')
    - $P' = e_{K2}(C) = d_{K1 \oplus\Delta}(e_{k2}(d_{K3}(C)))$

Related Keys Attack (cont.)

- Putting the equations,
  - $C = e_{K1}(P) = e_{K3}(d_{k2}(e_{K1}(P)))$
  - $P' = e_{K2}(C) = d_{K1 \oplus \Delta}(e_{k2}(d_{K3}(C)))$
- Together we get
  - $P' = d_{K1 \oplus \Delta}(e_{k2}(d_{K3}(e_{K3}(d_{k2}(e_{K1}(P))))))$
- and cancelling
  - $P' = d_{K1 \oplus \Delta}(e_{K1}(P))$
- Note the only key component present in the equation is k1 (k2, k3 are not present).
- Hence you can search for k1 in isolation.

Parallel Related Keys Attack

- Observation:
  - Can combine the ideas of a parallel key search and the 3DES related key attack.
- Implications:
  - 3DES keys become practically vulnerable.
  - E.g. Clayton- Bond key search machine.

Back to the claim…

Both the Parallel Key Search attack and the Related Key attack contradict the claims of the API designers!

- Implication:
  - Any user with read/write access to token objects can recover an key.
  - Hence must trust all users with such ability.
  - Must prevent unauthorized access to all sessions with read/write access to token objects.
- What about a session with only read only access to token objects?

Wrapping/Unwrapping of Private Keys using Secret Keys

- Once a private key has been BER-encoded as a PrivateKeyInfo type, the resulting string of bytes is encrypted with the secret key. This encryption must be done in CBC mode with PKCS padding. Unwrapping a wrapped private key undoes the above procedure.

Private Key Modification

- Observation:
  - Possible to modify two blocks in the clear key token by modifying.
  - Create error -> lead to fault analysis attacks.
- Implications:
  - Possible to perform various attacks that lead to recovery of the original private key.
  - E.g. Create error in CRT exponent thereby enabling Lenstra’s attack.

Private Key Modification (cont.)

- Solutions
  - Encrypted private keys must have strong cryptographic method to ensure integrity of key (e.g. MAC, hash or signature).
  - Confirm integrity of key using mathematical properties.

Public Key Based Key Exchange

- Two mechaisms for wrapping keys
  - **CKM_RSA_PKCS** (PKCS #1 RSA)
  - **CKM_RSA_X_509** (X.509 (Raw) RSA)
    - "…encrypts a byte string by converting it to an integer, most-significant byte first, applying "raw" RSA exponentiation, and converting the result to a byte string, mostsignificant byte first."
    - Encrypted Token T = k^e mod n where e is public exponent, k the key being exported and n the modulus).

Small Public Exponent

- The clear key is right justified in the field provided, and the field padded to the left with zeroes up to the size of the RSA encryption block (e.g. for key $k=k_1k_2\ldots k_{128}$ the padded message $m=0_1 0_2\ldots 0_{l-128}\, k_1k_2\ldots k_{128}$, where l is the length of the modulus). The resultant field is encrypted yielding $y = m^e \bmod n$.
- If then $m^e < n$ (i.e.,$e < \log_2 n/128$) then $y = m^e$. Thus $m = \log_2(y)/e$ and so for we can easily recover m and hence k.

More comments

- Due to the speed implications of an exponent with low Hamming weight, it is common for public keys to have exponents of 3 and $2^{16}+1$. There is often an option when generating a public key.
- This is also achievable if it is possible to modify the public exponent or if a public key with a suitable exponent exists on a system.
- For PKCS #11 this is a moot point since one can ‘conjure’ a public key with public exponent 3.

Trojan Public Key

- Aim: To export an unknown secret key (the target) under a known key
  - Attacker calculates a key pair (e,n), (d,n) for which d is known.
  - Supplies (e,n) to key wrap call requesting a given unkown secret key k be wrapped.
  - $y = k^e \bmod n$ is returned
  - Attacker uses knowledge of d to calculate $y^d \bmod n = k$.
- Solutions
  - A public key needs to be authenticated before use.

Trojan Wrapped Key

- Aim: To establish a known key in the system.
  - Attacker selects a key k.
  - Attacker calculates $y = k^e \bmod n$ for the known public key (e, n).
  - Supplies y to key unwrap call under the unknown private key (d,n).
  - The known key k is extracted and is available to the attacker in the system.
- Solution:
  - The authentication of wrapped keys before import.

Conclusions

- Multiple vulnerabilities in the standard do exist.
- Most are widely known.
- Changes are required to the standard (work in progress).
- Documentation.
- A low rate of knowledge transfer?