Proposal: Authenticated Attributes for Key Wrap in PKCS#11

Graham Steel

13 August 2014

1 About this Document

This document describes a shortcoming in PKCS#11 up to and including version 2.40 that has serious consequences for the applicability of the standard to a number of use cases, in particular in the world of Hardware Security Modules (HSMs), and a proposal for its solution.

2 The Problem

PKCS#11 includes the commands `C_WrapKey` and `C_UnwrapKey` for encrypted key export and import respectively. This allows a certain amount of security when managing keys, since keys are encrypted during transport. However, in an environment in which, in the worst case, an attacker might be able to make authorized PKCS#11 calls, very little security is offered. One reason is that `C_UnwrapKey` takes a template as input. This means that an attacker can, for example, take an encrypted key and import is as a non-sensitive key, i.e. one with `CKA_SENSITIVE=FALSE`, and then read it using `C_GetAttribute`.

Simple example: Target of the attack is key k, stored on the device with attributes `CKA_SENSITIVE` and `CKA_EXTRACTABLE` set, note that $\{|x|\}_y$ denotes encryption of plaintext x under key y, while $\&k$ indicates a handle pointing to key k. All keys are symmetric.

1. `C_GenerateKey`($\&k_1$, {`CKA_WRAP,CKA_UNWRAP`}) — generates k_1 with `CKA_WRAP,CKA_UNWRAP` set
2. `C_WrapKey`($\&k_1$, $\&k$) — gives $\{|k|\}_{k_1}$
3. `C_UnwrapKey`($\{|k|\}_{k_1}$, $\&k_1$, {`CKA_SENSITIVE=FALSE`}) — stores k at new location $\&k'$
4. `C_GetAttribute`($\&k'$) — gives value of k

There are many variations on this kind of attack, such as importing a key with both `CKA_WRAP` and `CKA_DECRYPT` set to `TRUE` [1, 3].

Since version 2.20, PKCS#11 contains a solution to this problem: `CKA_UNWRAP_TEMPLATE`. If a key k_1 has a certain template t as its value `CKA_UNWRAP_TEMPLATE` , then any time `C_UnwrapKey` is called using using k_1, attribute values in t will be assigned to the new key created. If the calling application supplies a template that is not consistent with the `CKA_UNWRAP_TEMPLATE`, the call fails.

This can be used to prevent the above attack. Assume k now has `CKA_UNWRAP_TEMPLATE` set to `CKA_SENSITIVE=TRUE`:

1. `C_GenerateKey`($\&k_1$, {`CKA_WRAP,CKA_UNWRAP`})
   generates k_1 with `CKA_WRAP,CKA_UNWRAP` set
2. `C_WrapKey`($\&k_1$, $\&k$) gives $\{|k|\}_{k_1}$
3. `C_UnwrapKey`($\{|k|\}_{k_1}$, $\&k_1$, {`CKA_SENSITIVE=FALSE`})
   fails with `CKR_TEMPLATE_INCONSISTENT`

The problem with this solution is that it is inflexible. To avoid attacks like wrap/decrypt, one is forced to specify all the attribute values in the `UNWRAP_TEMPLATE`, so only one kind of key profile can now be transported under each key encrypting key k. It makes sense to ensure that `CKA_UNWRAP_TEMPLATE` is not modifiable by `C_SetAttribute`, but this means it has to be fixed once and for all at key generation time. What's more, this solution is only secure when the attributes `CKA_TRUSTED` and `CKA_WRAP_WITH_TRUSTED` are also used to ensure that a key is not wrapped by an insecure wrapping key [5, 7]. This requires new wrapping keys to be approved by Security Officer login.

Given this complexity and the limited support for `UNWRAP_TEMPLATE` in real implementations, many users of HSMs just set their keys to have `CKA_EXTRACTABLE=FALSE`, to avoid any attacks. This makes backups and key sharing difficult, so they have to use proprietary solutions specific to their vendors at the expense of interoperability.

What is required is a flexible, interoperable way to share keys of any attribute profile securely using `C_WrapKey` and `C_UnwrapKey` that avoids all these problems.

3 Proposed Solution

Several academic papers have proposed cryptographic key management APIs with formal proofs of security properties in recent years [2, 4]. These proposals have one feature in common: wrapped key blobs contain the attributes the key had at the moment the wrap command was called. This enables a whole range of flexible configurations that are not vulnerable to the attacks described above.

In version 2.40 of the standard, there is already a new mechanism for RSA encryption that serves this purpose, `CKM_RSA_AES_KEY_WRAP`. We propose to add an equivalent mechanism for AES encryption, specifically in GCM mode. Additionally, we propose to clarify exactly how attributes are encoded and interpreted in `CKM_RSA_AES_KEY_WRAP` (which is not currently specified).

Below we show how the new wrapping mode prevents attack shown above.

1. `C_GenerateKey`($\&k_1$, {`CKA_WRAP,CKA_UNWRAP`})
   generates k_1 with `CKA_WRAP,CKA_UNWRAP` set
2. `C_WrapKey`($\&k_1$, $\&k$)
   gives $\{|k|\}_{k_1} t$ where t is an encoding of the attributes of k present as the *associated data* of the GCM encryption.
3. `C_UnwrapKey`($\{|k|\}_{k_1}$, $\&k_1$, {`CKA_SENSITIVE=FALSE`}
   fails with `CKR_WRAPPED_KEY_INVALID` since the template does not match t, so GCM decryption fails

4 Security

Various secure configurations become possible once authenticated encryption for wrapping is possible [6]. However, this security is dependent on a number of assumptions:

- The implementation of all commands that create and manage keys objects, e.g. `C_GenerateKey`, `C_CreateObject`, `C_SetAttribute`, `C_CopyObject` must avoid creating keys with conflicting attributes, e.g. wrap/decrypt, encrypt/unwrap etc. Note that this must be true for any implementation that is designed to achieve this level of security, whatever the wrapping mechanism.

- The new wrapping mechanisms with authenticated attributes must be the only ones made available, and no other copies of the keys with other mechanisms available can exist. Again, whatever the solution to the unwrapping issue, it seems clear that it must apply everywhere to achieve the level of security required.

References

[1] Matteo Bortolozzo, Matteo Centenaro, Riccardo Focardi, and Graham Steel. Attacking and fixing PKCS#11 security tokens. In *Proceedings of the 17th ACM Conference on Computer and Communications Security (CCS'10)*, pages 260 – 269, Chicago, Illinois, USA, October 2010. ACM Press.

[2] C. Cachin and N. Chandran. A secure cryptographic token interface. In *Computer Security Foundations (CSF-22)*, pages 141–153, Long Island, New York, 2009. IEEE Computer Society Press.

[3] J. Clulow. On the security of PKCS#11. In *Proceedings of CHES 2003*, pages 411–425, 2003.

[4] Vronique Cortier and Graham Steel. A generic security api for symmetric key management on cryptographic devices. *Information and Computation*, 2014. In press. Available online at http://dx.doi.org/10.1016/j.ic.2014.07.010.

[5] S. Delaune, S. Kremer, and G. Steel. Formal analysis of PKCS#11. In *Proceedings of the 21st IEEE Computer Security Foundations Symposium (CSF'08)*, pages 331–344, Pittsburgh, PA, USA, June 2008. IEEE Computer Society Press.

[6] S. Fröschle and G. Steel. Analysing PKCS#11 key management APIs with unbounded fresh data. In P. Degano and L. Viganò, editors, *Preliminary Proceedings of the Joint Workshop on Automated Reasoning for Security Protocol Analysis and Issues in the Theory of Security (ARSPA-WITS'09)*, volume 5511 of *Lecture Notes in Computer Science*, pages 92–106, York, UK, 2009. Springer. To appear.

[7] Sibylle B. Fröschle and Nils Sommer. Concepts and proofs for configuring pkcs#11. In Gilles Barthe, Anupam Datta, and Sandro Etalle, editors, *Formal Aspects in Security and Trust*, volume 7140 of *Lecture Notes in Computer Science*, pages 131–147. Springer, 2011.