PKCS #11 v2.20 Amendment 4, Draft 1

PKCS#11 HMAC Key Type Corrigendum

RSA Laboratories

16 June 2007

Table of Contents

Introduction

This document is an amendment to PKCS #11 v2.20 [1] and describes an update to clarify a problem with key management for HMAC objects.

1 Background

PKCS #11 section 12.8 requires that HMAC keys be stored as CKK_GENERIC_SECRET keys in CKO_SECRET_KEY objects:

> The HMAC secret key shall correspond to the PKCS11 generic secret key type.

However, section 12.7.2 of the specification, “Generic secret key objects”, states:

> Generic secret key objects (object class CKO_SECRET_KEY, key type CKK_GENERIC_SECRET) hold generic secret keys. *These keys do not support encryption, decryption, signatures or verification* (emphasis added); however, other keys can be derived from them.

Since the required CKK_GENERIC_SECRET key type doesn’t support the HMAC operations C_Sign and C_Verify, it’s not possible to key the HMAC algorithms using a conforming PKCS #11 implementation.

The lack of algorithm-specific keys also creates other problems. Once an HMAC key is stored in a device as a generic CKO_SECRET_KEY object, all type information is lost when the object is recreated by the PKCS #11 library for the consuming application. If for example an HMAC-SHA1 object is created in a device and then later retrieved by an application via C_FindObjects, there’s no indication of what the retrieved object actually is (or was) since the type information has been lost. Furthermore, the loss of type information may lead to security problems as it allows keys to be moved from their original algorithm to other, less secure algorithms, or in general to be reused for purposes for which they were never intended.

To date vendors have worked around the keying problem either by ignoring the requirements for CKK_GENERIC_SECRET keys in the PKCS #11 specification or by defining their own nonstandard types for HMAC keys. A third option used by some vendors is to keep the implementations strictly compliant with the specification, in which case HMAC objects can’t be used because they can’t be keyed.

This amendment defines algorithm-specific HMAC key types with MAC-capable semantics to replace the use of generic non-MAC-capable keys for HMAC objects.

1.1 Additional PKCS #11 Key Types

The following algorithm-specific key types replace the generic secret key type currently specified in PKCS #11 v2.20:

```
CKK_MD5_HMAC
CKK_SHA_1_HMAC
CKK_RIPEMD128_HMAC
CKK_RIPEMD160_HMAC
CKK_SHA256_HMAC
CKK_SHA384_HMAC
CKK_SHA512_HMAC
```

These key types are used in place of the current CKK_GENERIC_SECRET key type when creating HMAC CKO_SECRET_KEY objects.

1.2 Implementation Notes

As current implementations can only support HMAC keying by violating the PKCS #11 v2.20 specification, any behaviour in regard to allowing HMAC use with CKK_GENERIC_SECRET keys is, by definition, outside the scope of the specification. However, it is recommended that implementations that support the new HMAC-specific key types specified in this amendment return CKR_KEY_TYPE_INCONSISTENT when CKK_GENERIC_SECRET keys are used with an HMAC mechanism (which strictly conforming implementations should be doing anyway). It is also recommended that vendors who have already implemented their own HMAC key types in the CKK_VENDOR_DEFINED space move quickly to supporting the new standardised key types. Application developers using existing vendor-specific HMAC key types can make their applications forwards-compatible by first attempting to create an object using the new HMAC key type and if that fails to fall back to the existing vendor-specific type. In this way applications will automatically support the new types while remaining backwards-compatible with existing vendor-specific workarounds.

A. Manifest constants

The following definitions can be found in the appropriate header file.

```
#define CKK_MD5_HMAC                       0x00000027
#define CKK_SHA_1_HMAC                     0x00000028
#define CKK_RIPEMD128_HMAC                 0x00000029
#define CKK_RIPEMD160_HMAC                 0x0000002A
#define CKK_SHA256_HMAC                    0x0000002B
#define CKK_SHA384_HMAC                    0x0000002C
#define CKK_SHA512_HMAC                    0x0000002D
```

B. Intellectual property considerations

RSA Security Inc. makes no patent claims on the general constructions described in this document, although specific underlying techniques may be covered.

RSA is a registered trademark of RSA Security Inc. in the United States and/or other countries. The names of other products or services mentioned may be the trademarks of their respective owners.

This document and the information contained herein are provided on an "AS IS" basis and RSA SECURITY DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. RSA Security Inc. makes no representations regarding intellectual property claims by other parties. Such determination is the responsibility of the user.

C. References

[1] RSA Laboratories. *PKCS #11: Cryptographic Token Interface Standard*. Version 2.20, June 2004. URL: ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-11/v2-20/pkcs-11v2-20.pdf.

D. About PKCS

The *Public Key Cryptography Standards* are documents produced by RSA, The Security Division of EMC, in cooperation with secure systems developers for the purpose of simplifying integration and management of accelerating the deployment of public-key cryptography and strong authentication technology into secure applications, and to enhance the user experience of these technologies.

RSA plans further development of the PKCS series through mailing list discussions and occasional workshops, and suggestions for improvement are welcome. Results may also be submitted to standards forums. For more information, contact:

PKCS Editor
RSA, The Security Division of EMC
174 Middlesex Turnpike
Bedford, MA 01730 USA
pkcs-editor@rsasecurity.com
http://www.rsasecurity.com/rsalabs/