S/MIME Working Group   Jongwook Park (KISA)
Internet Draft   Sungjae Lee (KISA)
Document: draft-park-cms-seed-00.txt   Jinsu Hyun (KISA)
Expires: April 2003   Jaeil Lee (KISA)
                                                            October 2003

Use of the SEED Encryption Algorithm in CMS

Status of this Memo

Abstract

This document specifies the conventions for using the SEED encryption algorithm for encryption with the Cryptographic Message Syntax (CMS).

1. Introduction

This document specifies the conventions for usting the SEED encryption algorithm [TTASSEED] for encryption with the Cryptographic Message Syntax (CMS) [CMS]. The relevant object identifiers (OIDs) and processing steps are provided so that SEED may be used in the CMS specification (RFC 3369, RFC 3370) for content and key encryption.

1.1 SEED

SEED is a symmetric encryption algorithm that had been developed by KISA (Korea Information Security Agency) and a group of experts since 1998. The input/output block size of SEED is 128-bit and the key length is also 128-bit. SEED has the 16-round Feistel structure. A 128-bit input is divided into two 64-bit blocks and the right 64-bit block is an input to the round function with a 64-bit subkey generated from the key scheduling.

SEED is easily implemented in various software and hardware because it is designed to increase the efficiency of memory storage and the simplicity in generating keys without degrading the security of the algorithm. In particular, it can be effectively adopted to a computing environment with a restricted resources such as a mobile devices, smart cards and so on.

SEED is robust against known attacks including DC (Differential cryptanalysis), LC (Linear cryptanalysis) and related key attacks, etc. SEED has gone through wide public scrutinizing procedures. Especially, it has been evaluated and also considered cryptographically secure by trustworhty organizations such as ISO/IEC JTC 1/SC 27 and Japan CRYTEC (Cryptography Reasearch and Evaluation Comittees) [ISOSEED][CRYPTEC].

SEED is a national industrial association standard [TTASSEED] and is widely used in South Korea for electronic commerce and financial services operated on wired & wireless PKI.

1.2 Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document (in uppercase, as shown) are to be interpreted as described in [RFC2119].

2. Object Identifiers for Content and Key Encryption

This section provides the OIDs and processing information necessary for SEED to be used for content and key encryption in CMS. SEED is added to the set of optional symmetric encryption algorithms in CMS by providing two classes of unique object identifiers (OIDs). One OID class defines the content encryption algorithms and the other defines the key encryption algorithms. Thus a CMS agent can apply

SEED either for content or key encryption by selecting the corresponding object identifier, supplying the required parameter, and starting the program code.

2.1 OIDs for Content Encryption

SEED is added to the set of symmetric content encryption algorithms defined in [CMSALG]. The SEED content-encryption algorithm in Cipher Block Chaining (CBC) mode has the following object identifier:

```
id-seedCBC OBJECT IDENTIFIER ::=
 { iso(1) member-body(2) korea(410) kisa(200004)
      algorithm(1) seedCBC(4) }
```

The AlgorithmIdentifier parameters field MUST be present, and the parameters field MUST contain the value of IV:

```
SeedCBCParameter ::= SeedIV  --  Initialization Vector

SeedIV ::= OCTET STRING (SIZE(16))
```

The plain text is padded according to Section 6.3 of [CMS].

2.2 OIDs for Key Encryption

The key-wrap/unwrap procedures used to encrypt/decrypt a SEED content-encryption key (CEK) with a SEED key-encryption key (KEK) are specified in Section 3. Generation and distribution of key-encryption keys are beyond the scope of this document.

The SEED key-encryption algorithm has the following object identifier:

```
id-npki-app-cmsSeed-wrap OBJECT IDENTIFIER ::=
   { iso(1) member-body(2) korea(410) kisa(200004) npki-app(7)
     smime(1) alg(1) cmsSEED-wrap(1) }
```

The parameter associated with this object identifier MUST be absent, because the key wrapping procedure itself defines how and when to use an IV.

3. Key Wrap Algorithm

SEED key wrapping and unwrapping is done in conformance with the AES key wrap algorithm [AES-WRAP][RFC3394].

3.1 Notation and Defintions

The following notation is used in the description of the key wrapping algorithms:


```
SEED(K, W)     Encrypt W using the SEED codebook with key K
SEED-1(K, W)   Decrypt W using the SEED codebook with key K
MSB(j, W)      Return the most significant j bits of W
LSB(j, W)      Return the least significant j bits of W
B1 ^ B2        The bitwise exclusive or (XOR) of B1 and B2
B1 | B2        Concatenate B1 and B2
K              The key-encryption key K
n              The number of 64-bit key data blocks
s              The number of steps in the wrapping process,
               s = 6n
P[i]           The ith plaintext key data block
C[i]           The ith ciphertext data block
A              The 64-bit integrity check register
R[i]           An array of 64-bit registers where
                  i = 0, 1, 2, ..., n
A[t], R[i][t]  The contents of registers A and R[i] after
               encryption step t.
IV             The 64-bit initial value used during the
               wrapping process.
```

In the key wrap algorithm, the concatenation function will be used to concatenate 64-bit quantities to form the 128-bit input to the SEED codebook. The extraction functions will be used to split the 128-bit output from the SEED codebook into two 64-bit quantities.

3.2 SEED Key Wrap

Key wrapping with SEED is identical to Section 2.2.1 of [RFC3394] with "AES" replaced by "SEED".

The inputs to the key wrapping process are the KEK and the plaintext to be wrapped. The plaintext consists of n 64-bit blocks, containing the key data being wrapped. The key wrapping process is described below.

```
Inputs:      Plaintext, n 64-bit values {P1, P2, ..., Pn}, and
             Key, K (the KEK).
Outputs:     Ciphertext, (n+1) 64-bit values {C0, C1, ..., Cn}.
```

1) Initialize variables.

```
       Set A[0] to an initial value (see Section 3.4)
       For i = 1 to n
            R[0][i] = P[i]
```

2) Calculate intermediate values.

```
       For t = 1 to s, where s = 6n
```



```
           A[t] = MSB(64, SEED(K, A[t-1] | R[t-1][1])) ^ t
           For i = 1 to n-1
               R[t][i] = R[t-1][i+1]
           R[t][n] = LSB(64, SEED(K, A[t-1] | R[t-1][1]))
```

3) Output the results.

```
       Set C[0] = A[t]
       For i = 1 to n
           C[i] = R[t][i]
```

An alternative description of the key wrap algorithm involves indexing rather than shifting. This approach allows one to calculate the wrapped key in place, avoiding the rotation in the previous description. This produces identical results and is more easily implemented in software.

```
Inputs:  Plaintext, n 64-bit values {P1, P2, ..., Pn}, and
         Key, K (the KEK).
Outputs: Ciphertext, (n+1) 64-bit values {C0, C1, ..., Cn}.
```

1) Initialize variables.

```
        Set A = IV, an initial value (see Section 3.4)
        For i = 1 to n
            R[i] = P[i]
```

2) Calculate intermediate values.

```
        For j = 0 to 5
            For i=1 to n
                B = SEED(K, A | R[i])
                A = MSB(64, B) ^ t where t = (n*j)+i
                R[i] = LSB(64, B)
```

3) Output the results.

```
        Set C[0] = A
        For i = 1 to n
            C[i] = R[i]
```

3.3 SEED Key Unwrap

Key unwrapping with SEED is identical to Section 2.2.2 of [RFC3394], with "AES" replaced by "SEED".

The inputs to the unwrap process are the KEK and (n+1) 64-bit blocks of ciphertext consisting of previously wrapped key. It returns n

blocks of plaintext consisting of the n 64-bit blocks of the decrypted key data.

```
Inputs:  Ciphertext, (n+1) 64-bit values {C0, C1, ..., Cn}, and
         Key, K (the KEK).
Outputs: Plaintext, n 64-bit values {P1, P2, ..., Pn}.
```

1) Initialize variables.

```
        Set A[s] = C[0] where s = 6n
        For i = 1 to n
            R[s][i] = C[i]
```

2) Calculate the intermediate values.

```
        For t = s to 1
            A[t-1] = MSB(64, SEED-1(K, ((A[t] ^ t) | R[t][n]))
            R[t-1][1] = LSB(64, SEED-1(K, ((A[t]^t) | R[t][n]))
            For i = 2 to n
                R[t-1][i] = R[t][i-1]
```

3) Output the results.

```
        If A[0] is an appropriate initial value (see Section 3.4),
        Then
            For i = 1 to n
                P[i] = R[0][i]
        Else
            Return an error
```

The unwrap algorithm can also be specified as an index based operation, allowing the calculations to be carried out in place. Again, this produces the same results as the register shifting approach.

```
Inputs:  Ciphertext, (n+1) 64-bit values {C0, C1, ..., Cn}, and
         Key, K (the KEK).
Outputs: Plaintext, n 64-bit values {P0, P1, K, Pn}.
```

1) Initialize variables.

```
        Set A = C[0]
        For i = 1 to n
            R[i] = C[i]
```

2) Compute intermediate values.

```
        For j = 5 to 0
```



```
            For i = n to 1
                B = SEED-1(K, (A ^ t) | R[i]) where t = n*j+i
                A = MSB(64, B)
                R[i] = LSB(64, B)
```

3) Output results.

```
        If A is an appropriate initial value (see Section 3.4),
```

```
        Then
            For i = 1 to n
                P[i] = R[i]
        Else
            Return an error
```

3.4 Key Data Integrity -- the Initial Value

The initial value (IV) refers to the value assigned to A[0] in the first step of the wrapping process. This value is used to obtain an integrity check on the key data. In the final step of the unwrapping process, the recovered value of A[0] is compared to the expected value of A[0]. If there is a match, the key is accepted as valid, and the unwrapping algorithm returns it. If there is not a match, then the key is rejected, and the unwrapping algorithm returns an error.

The exact properties achieved by this integrity check depend on the definition of the initial value. Different applications may call for somewhat different properties; for example, whether there is need to determine the integrity of key data throughout its lifecycle or just when it is unwrapped. This specification defines a default initial value that supports integrity of the key data during the period it is wrapped (in Section 3.4.1). Provision is also made to support alternative initial values (in Section 3.4.2).

3.4.1 Default Initial Value

The default initial value (IV) is defined to be the hexadecimal constant:

```
A[0] = IV = A6A6A6A6A6A6A6A6
```

The use of a constant as the IV supports a strong integrity check on the key data during the period that it is wrapped. If unwrapping produces A[0] = A6A6A6A6A6A6A6A6, then the chance that the key data is corrupt is 2^-64. If unwrapping produces A[0] any other value, then the unwrap must return an error and not return any key data.

3.4.2 Alternative Initial Values

When the key wrap is used as part of a larger key management protocol or system, the desired scope for data integrity may be more than just the key data or the desired duration for more than just the period that it is wrapped. Also, if the key data is not just an SEED key, it may not always be a multiple of 64 bits. Alternative definitions of the initial value can be used to address such problems. According to [RFC3394], NIST will define alternative initial values in future key management publications as needed. In order to accommodate a set of alternatives that may evolve over time, key wrap implementations that are not application-specific will require some flexibility in the way that the initial value is set and tested.

4. SMIMECapabilities Attribute

An S/MIME client SHOULD announce the set of cryptographic functions it supports by using the S/MIME capabilities attribute. This attribute provides a partial list of OIDs of cryptographic functions and MUST be signed by the client. The functions' OIDs SHOULD be logically separated in functional categories and MUST be ordered with respect to their preference.

RFC 2633 [RFC2633], Section 2.5.2 defines the SMIMECapabilities signed attribute (defined as a SEQUENCE of SMIMECapability SEQUENCEs) to be used to specify a partial list of algorithms that the software announcing the SMIMECapabilities can support.

If an S/MIME client is required to support symmetric encryption with SEED, the capabilities attribute MUST contain the SEED OID specified above in the category of symmetric algorithms. The parameter associated with this OID MUST be SeedSMimeCapability.

```
SeedSMimeCapabilty ::= NULL
```

The SMIMECapability SEQUENCE representing SEED MUST be DER-encoded as the following hexadecimal strings:

```
30 0d 06 0a 2a 83 1a 8c 9a 44 07 01 01 01 05 00
```

When a sending agent creates an encrypted message, it has to decide which type of encryption algorithm to use. In general the decision process involves information obtained from the capabilities lists included in messages received from the recipient, as well as other information such as private agreements, user preferences, legal restrictions, and so on. If users require SEED for symmetric

encryption, it MUST be supported by the S/MIME clients on both the sending and receiving side, and it MUST be set in the user preferences.

5. Security Considerations

This document specifies the use of SEED for encrypting the content of a CMS message and for encrypting the symmetric key used to encrypt the content of a CMS message, and the other mechanisms are the same as the existing ones. Therefore, the security considerations described in the CMS specifications [CMS][CMSALG] and the AES key wrap algorithm [AES-WRAP][RFC3394] can be applied to this document. No security problem has been found on SEED [CRYPTREC].

6. Intellectual Property Statement

The IETF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on the IETF's procedures with respect to rights in standards-track and standards-related documentation can be found in BCP-11. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementors or users of this specification can be obtained from the IETF Secretariat.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to practice this standard. Please address the information to the IETF Executive Director.

7. Full Copyright Statement

8. References

8.1 Normative Reference

[CMS] R. Housley, "Cryptographic Message Syntax", RFC 3369, August 2002.

[CMSALG] R. Housley, "Cryptographic Message Syntax (CMS) Algorithms", RFC 3370, August 2002.

[RFC2119] S. Bradner, "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2633] Ramsdell, B., Editor. S/MIME Version 3 Message Specification. RFC 2633. June 1999.

[RFC3394] J. Schaad and R. Housley, "Advanced Encryption Standard (AES) Key Wrap Algorithm", RFC 3394, September 2002.

[AES-WRAP] National Institute of Standards and Technology. AES Key Wrap Specification. 17 November 2001. http://csrc.nist.gov/encryption/kms/key-wrap.pdf

8.2 Informative Reference

[TTASSEED] Telecommunications Technology Association (TTA), South Korea, "128-bit Symmetric Block Cipher (SEED)", TTAS.KO-12.0004, September, 1998 (In Korean) http://www.tta.or.kr/English/new/main/index.htm

[ISOSEED] ISO/IEC, ISO/IEC JTC1/SC 27 N 256r1, "National Body contributions on NP 18033 Encryption algorithms in response to document SC 27 N 2563", October, 2000

[CRYPTREC] Information-technology Promotion Agency (IPA), Japan, CRYPTREC. "SEED Evaluation Report", February, 2002 http://www.kisa.or.kr

9. Authors' Address

Jongwook Park
Korea Information Security Agency
Phone: +82-2-405-5432
FAX: +82-2-405-5499
Email: khopri@kisa.or.kr

Sungjae Lee
Korea Information Security Agency
Phone: +82-2-405-5243
FAX: +82-2-405-5499
Email: sjlee@kisa.or.kr

Jinsu Hyun

Korea Information Security Agency
Phone: +82-2-405-5252
FAX: +82-2-405-5499
Email: jshyun@kisa.or.kr

Jaeil Lee
Korea Information Security Agency
Phone: +82-2-405-5421
FAX: +82-2-405-5499
Email: jilee@kisa.or.kr

Appendix A ASN.1 Module

```
SeedEncryptionAlgorithmInCMS
```



```
    { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1)
      pkcs9(9) smime(16) modules(0) id-mod-cms-seed(?) }

DEFINITIONS IMPLICIT TAGS ::=
BEGIN


id-seedCBC OBJECT IDENTIFIER ::=
   { iso(1) member-body(2) korea(410) kisa(200004)
     algorithm(1) seedCBC(4) }

--  Initialization Vector

SeedCBCParameter ::= SeedIV

SeedIV ::= OCTET STRING (SIZE(16))

-- SEED Key Wrap Algorithm identifiers - Parameter is absent.

id-npki-app-cmsSeed-wrap OBJECT IDENTIFIER ::=
   { iso(1) member-body(2) korea(410) kisa(200004) npki-app(7)
     smime(1) alg(1) cmsSEED-wrap(1) }

-- SEED S/MIME Capabilty parameter
```

```
SeedSMimeCapability ::= NULL

END
```