Network Working Group
Request for Comments: 4010
Category: Standards Track

J. Park
S. Lee
J. Kim
J. Lee
KISA

February 2005

Use of the SEED Encryption Algorithm in Cryptographic Message Syntax (CMS)

Status of This Memo

Copyright Notice

Abstract

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

1. Introduction

This document specifies the conventions for using the SEED encryption algorithm [SEED][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 3852, RFC 3370) for content and key encryption.

1.1. SEED

SEED is a symmetric encryption algorithm developed by KISA (Korea Information Security Agency) and a group of experts since 1998. The input/output block size and key length of SEED is 128-bits. 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 takes less memory to implement than other algorithms and generates keys without degrading the security of the algorithm. In particular, it can be effectively adopted in a computing environment with a restricted resources, such as mobile devices and smart cards.

SEED is robust against known attacks including DC (Differential cryptanalysis), LC (Linear cryptanalysis), and related key attacks. SEED has gone through wide public scrutinizing procedures. It has been evaluated and is considered cryptographically secure by credible organizations such as ISO/IEC JTC 1/SC 27 and Japan CRYPTREC (Cryptography Research and Evaluation Committees) [ISOSEED][CRYPTREC].

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

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 Initialization Vector (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 [RFC3394].

3.1. Notation and Definitions

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 {P[1], P[2], ..., P[n]}, and
           Key, K (the KEK).
  Outputs: Ciphertext, (n+1) 64-bit values {C[0], C[1], ..., C[n]}.
```

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[s]
For i = 1 to n
  C[i] = R[s][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 {P[1], P[2], ..., P[n]}, and
         Key, K (the KEK).
Outputs: Ciphertext, (n+1) 64-bit values {C[0], C[1], ..., C[n]}.
```

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 {C[0], C[1], ..., C[n]},
           and Key, K (the KEK).
  Outputs: Plaintext, n 64-bit values {P[1], P[2], ..., P[n]}.
```

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 {C[0], C[1], ..., C[n]},
           and Key, K (the KEK).
  Outputs: Plaintext, n 64-bit values {P[0], P[1], ..., P[n]}.
```

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 a 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 the 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, and the desired duration may be more than just the period that it is wrapped. Also, if the key data is not just a 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 RFC 3394 [RFC3394], NIST will define alternative initial values in future key management publications as they are needed. To accommodate a set of alternatives that may evolve over time, non-application-specific key wrap implementations will require some flexibility in the way 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 3851 [RFC3851], 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 0C 06 08 2A 83 1A 8C 9A 44 01 04 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 and legal restrictions. If local policy requires the use of SEED for symmetric encryption, then both the sending and receiving S/MIME clients must support it, and SEED must be configured as the preferred symmetric algorithm.

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, with the other mechanisms being the same as the existing ones. Therefore, the security considerations described in the CMS specifications [CMS][CMSALG] and the AES key wrap algorithm [RFC3394] can be applied to this document. No security problem has been found on SEED [CRYPTREC].

6. References

6.1. Normative References

[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

[CMS] Housley, R., "Cryptographic Message Syntax (CMS)", RFC 3852, July 2004.

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

[RFC3851] Ramsdell, B., "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.1 Message Specification", RFC 3851, July 2004.

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

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

6.2. Informative References

[SEED] Park, J., Lee, S., Kim, J., and J. Lee, "The SEED Encryption Algorithm", RFC 4009, February 2005.

[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

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(24) }


DEFINITIONS IMPLICIT TAGS ::=

BEGIN

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

  --  Initialization Vector (IV)

  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 Capability parameter

  SeedSMimeCapability ::= NULL

END
```

Authors' Addresses

Jongwook Park
Korea Information Security Agency
78, Garak-Dong, Songpa-Gu, Seoul, 138-803
REPUBLIC OF KOREA

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

Jeeyeon Kim
Korea Information Security Agency

Phone: +82-2-405-5238
FAX : +82-2-405-5499
EMail: jykim@kisa.or.kr

Jaeil Lee
Korea Information Security Agency
Phone: +82-2-405-5300
FAX : +82-2-405-5499
EMail: jilee@kisa.or.kr

Full Copyright Statement

Intellectual Property

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.