Network Working Group
Request for Comments: 2876
Category: Informational

J. Pawling
WGSI, A Getronics Company

July 2000

Use of the KEA and SKIPJACK Algorithms in CMS

Status of this Memo

Copyright Notice

Abstract

This document describes the conventions for using the Key Exchange Algorithm (KEA) and SKIPJACK encryption algorithm in conjunction with the Cryptographic Message Syntax [CMS] enveloped-data and encrypted-data content types.

1. Introduction

Throughout this document, the terms MUST, MUST NOT, SHOULD and MAY are used in capital letters. This conforms to the definitions in [MUSTSHOULD]. [MUSTSHOULD] defines the use of these key words to help make the intent of standards track documents as clear as possible. The same key words are used in this document to help implementers achieve interoperability. Software that claims compliance with this document MUST provide the capabilities as indicated by the MUST, MUST NOT, SHOULD and MAY terms. The KEA and SKIPJACK cryptographic algorithms are described in [SJ-KEA].

2. Content Encryption Process

This section applies to the construction of both the enveloped-data and encrypted-data content types. Compliant software MUST meet the requirements stated in [CMS] Section 6.3, "Content-encryption Process". The input to the encryption process MUST be padded to a multiple of eight octets using the padding rules described in [CMS] Section 6.3. The content MUST be encrypted as a single string using the SKIPJACK algorithm in 64-bit Cipher Block Chaining (CBC) mode using randomly-generated 8-byte Initialization Vector (IV) and 80-bit SKIPJACK content-encryption key (CEK) values.

3. Content Decryption Process

This section applies to the processing of both the enveloped-data and encrypted-data content types. The encryptedContent MUST be decrypted as a single string using the SKIPJACK algorithm in 64-bit CBC mode. The 80-bit SKIPJACK CEK and the 8-byte IV MUST be used as inputs to the SKIPJACK decryption process. Following decryption, the padding MUST be removed from the decrypted data. The padding rules are described in [CMS] Section 6.3, "Content-encryption Process".

4. Enveloped-data Conventions

The CMS enveloped-data content type consists of an encrypted content and wrapped CEKs for one or more recipients. Compliant software MUST meet the requirements for constructing an enveloped-data content type stated in [CMS] Section 6, "Enveloped-data Content Type". [CMS] Section 6 should be studied before reading this section, because this section does not repeat the [CMS] text.

An 8-byte IV and 80-bit CEK MUST be randomly generated for each instance of an enveloped-data content type as inputs to the SKIPJACK algorithm for use to encrypt the content. The SKIPJACK CEK MUST only be used for encrypting the content of a single instance of an enveloped-data content type.

KEA and SKIPJACK can be used with the enveloped-data content type using either of the following key management techniques defined in [CMS] Section 6:

1) Key Agreement: The SKIPJACK CEK is uniquely wrapped for each recipient using a pairwise symmetric key-encryption key (KEK) generated using KEA using the originator's private KEA key, recipient's public KEA key and other values. Section 4.2 provides additional details.

2) "Previously Distributed" Symmetric KEK: The SKIPJACK CEK is wrapped using a "previously distributed" symmetric KEK (such as a Mail List Key). The methods by which the symmetric KEK is generated and distributed are beyond the scope of this document. Section 4.3 provides more details.

[CMS] Section 6 also defines the concept of the key transport key management technique. The key transport technique MUST NOT be used with KEA.

4.1. EnvelopedData Fields

The enveloped-data content type is Abstract Syntax Notation.1 (ASN.1) encoded using the EnvelopedData syntax. The fields of the EnvelopedData syntax must be populated as follows:

The EnvelopedData version MUST be 2.

If key agreement is being used, then the EnvelopedData originatorInfo field SHOULD be present and SHOULD include the originator's KEA X.509 v3 certificate containing the KEA public key associated with the KEA private key used to form each pairwise symmetric KEK used to wrap each copy of the SKIPJACK CEK. The issuers' X.509 v3 certificates required to form the complete certification path for the originator's KEA X.509 v3 certificate MAY be included in the EnvelopedData originatorInfo field. Self-signed certificates SHOULD NOT be included in the EnvelopedData originatorInfo field.

The EnvelopedData RecipientInfo CHOICE is dependent on the key management technique used. Sections 4.2 and 4.3 provide more information.

The EnvelopedData encryptedContentInfo contentEncryptionAlgorithm algorithm field MUST be the id-fortezzaConfidentialityAlgorithm object identifier (OID). The EnvelopedData encryptedContentInfo contentEncryptionAlgorithm parameters field MUST include the random 8-byte IV used as the input to the content encryption process.

The EnvelopedData unprotectedAttrs MAY be present.

4.2. Key Agreement

This section describes the conventions for using KEA and SKIPJACK with the CMS enveloped-data content type to support key agreement. When key agreement is used, then the RecipientInfo keyAgreeRecipientInfo CHOICE MUST be used.

If the EnvelopedData originatorInfo field does not include the originator's KEA X.509 v3 certificate, then each recipientInfos KeyAgreementRecipientInfo originator field MUST include the issuerAndSerialNumber CHOICE identifying the originator's KEA X.509 v3 certificate. If the EnvelopedData originatorInfo field includes the originator's KEA X.509 v3 certificate, then each recipientInfos KeyAgreementRecipientInfo originator field MUST include either the subjectKeyIdentifier CHOICE containing the value from the subjectKeyIdentifier extension of the originator's KEA X.509 v3 certificate or the issuerAndSerialNumber CHOICE identifying the

originator's KEA X.509 v3 certificate. To minimize the size of the EnvelopedData, it is recommended that the subjectKeyIdentifier CHOICE be used.

In some environments, the KeyAgreementRecipientInfo originator field MAY include the originatorKey CHOICE. The originatorKey CHOICE SHOULD NOT be used with KEA for e-mail transactions. Within a controlled security architecture, a module may produce KEA key pairs for use in conjunction with internal/local storage of encrypted data. In this case, there may not be an X.509 certificate associated with a (possibly) short term or one time use public KEA key. When originatorKey is used, then the KEA public key MUST be conveyed in the publicKey BIT STRING as specified in [KEA] Section 3.1.2. The originatorKey algorithm identifier MUST be the id-keyExchangeAlgorithm OID. The originatorKey algorithm parameters field MUST contain the KEA "domain identifier" (ASN.1 encoded as an OCTET STRING) identifying the KEA algorithm parameters (i.e., p/q/g values) associated with the KEA public key. [KEA] Section 3.1.1 describes the method for computing the KEA domain identifier value.

4.2.1. SKIPJACK CEK Wrap Process

The SKIPJACK CEK is uniquely wrapped for each recipient of the EnvelopedData using a pairwise KEK generated using the KEA material of the originator and the recipient along with the originator's User Keying Material (UKM) (i.e. Ra). The CMS EnvelopedData syntax provides two options for wrapping the SKIPJACK CEK for each recipient using a KEA-generated KEK. The "shared Originator UKM" option SHOULD be used when constructing EnvelopedData objects. The "unique originator UKM" option MAY be used when constructing EnvelopedData objects. Compliant software MUST be capable of processing EnvelopedData objects constructed using both options.

1) Shared Originator UKM Option: CMS provides the ability for a single, shared originator's UKM to be used to generate each pairwise KEK used to wrap the SKIPJACK CEK for each recipient. When using the shared originator UKM option, a single RecipientInfo KeyAgreeRecipientInfo structure MUST be constructed to contain the wrapped SKIPJACK CEKs for all of the KEA recipients sharing the same KEA parameters. The KeyAgreeRecipientInfo structure includes multiple RecipientEncryptedKey fields that each contain the SKIPJACK CEK wrapped for a specific recipient.

2) Unique Originator UKM Option: CMS also provides the ability for a unique originator UKM to be used to generate each pairwise KEK used to wrap the SKIPJACK CEK for each recipient. When using the unique originator UKM option, a separate RecipientInfo KeyAgreeRecipientInfo structure MUST be constructed for each recipient. Each KeyAgreeRecipientInfo structure includes a single RecipientEncryptedKey field containing the SKIPJACK CEK wrapped for the recipient. This option requires more overhead than the shared UKM option because the KeyAgreeRecipientInfo fields (i.e. version, originator, ukm, keyEncryptionAlgorithm) must be repeated for each recipient.

The next two paragraphs apply to both options.

The KeyAgreeRecipientInfo keyEncryptionAlgorithm algorithm field MUST include the id-kEAKeyEncryptionAlgorithm OID. The KeyAgreeRecipientInfo keyEncryptionAlgorithm parameters field MUST contain a KeyWrapAlgorithm as specified in [CMS] Appendix A, "ASN.1 Module". The algorithm field of KeyWrapAlgorithm MUST be the id-fortezzaWrap80 OID indicating that the FORTEZZA 80-bit wrap function is used to wrap the 80-bit SKIPJACK CEK. Since the FORTEZZA 80-bit wrap function includes an integrity check value, the wrapped SKIPJACK key is 96 bits long. The parameters field of KeyWrapAlgorithm MUST be absent.

If the originator is not already an explicit recipient, then a copy of the SKIPJACK CEK SHOULD be wrapped for the originator and included in the EnvelopedData. This allows the originator to decrypt the contents of the EnvelopedData.

4.2.1.1. SKIPJACK CEK Wrap Process Using A Shared Originator UKM Value

This section describes how a shared originator UKM value is used as an input to KEA to generate each pairwise KEK used to wrap the SKIPJACK CEK for each recipient.

When using the shared originator UKM option, a single RecipientInfo KeyAgreeRecipientInfo structure MUST be constructed to contain the wrapped SKIPJACK CEKs for all of the KEA recipients using the same set of KEA parameters. If all recipients' KEA public keys were generated using the same set of KEA parameters, then there MUST only be a single RecipientInfo KeyAgreeRecipientInfo structure for all of the KEA recipients. If the recipients' KEA public keys were generated using different sets of KEA parameters, then multiple RecipientInfo KeyAgreeRecipientInfo fields MUST be constructed because the originatorIdentifierOrKey will be different for each distinct set of recipients' KEA parameters.

A unique 128-byte originator's UKM MUST be generated for each distinct set of recipients' KEA parameters. The originator's UKM MUST be placed in each KeyAgreeRecipientInfo ukm OCTET STRING.

The originator's and recipient's KEA parameters MUST be identical to use KEA to successfully generate a pairwise KEK. [KEA] describes how a KEA public key is conveyed in an X.509 v3 certificate. [KEA] states that the KEA parameters are not included in KEA certificates; instead, a "domain identifier" is supplied in the subjectPublicKeyInfo algorithm parameters field of every KEA certificate. The values of the KEA domain identifiers in the originator's and recipient's KEA X.509 v3 certificates can be compared to determine if the originator's and recipient's KEA parameters are identical.

The following steps MUST be repeated for each recipient:

1) KEA MUST be used to generate the pairwise KEK based on the originator's UKM, originator's private KEA key, recipient's 128 byte public KEA key (obtained from the recipient's KEA X.509 v3 certificate) and the recipient's 128-byte public KEA key used as the Rb value.

2) The SKIPJACK CEK MUST be wrapped using the KEA-generated pairwise KEK as input to the FORTEZZA 80-bit wrap function. The FORTEZZA 80-bit wrap function takes the 80-bit SKIPJACK CEK along with a 16-bit integrity checkvalue and produces a 96-bit result using the KEA-generated pairwise KEK.

3) A new RecipientEncryptedKey SEQUENCE MUST be constructed for the recipient.

4) The value of the subjectKeyIdentifier extension from the recipient's KEA X.509 v3 certificate MUST be placed in the recipient's RecipientEncryptedKey rid rKeyId subjectKeyIdentifier field. The KeyAgreeRecipientIdentifier CHOICE MUST be rKeyId. The date and other fields MUST be absent from the recipientEncryptedKey rid rKeyId SEQUENCE.

5) The wrapped SKIPJACK CEK MUST be placed in the recipient's RecipientEncryptedKey encryptedKey OCTET STRING.

6) The recipient's RecipientEncryptedKey MUST be included in the KeyAgreeRecipientInfo recipientEncryptedKeys SEQUENCE OF RecipientEncryptedKey.

4.2.1.2. SKIPJACK CEK Wrap Process Using Unique Originator UKM Values

This section describes how a unique originator UKM value is generated for each recipient to be used as an input to KEA to generate that recipient's pairwise KEK.

The following steps MUST be repeated for each recipient:

1) A new RecipientInfo KeyAgreeRecipientInfo structure MUST be constructed.

2) A unique 128-byte originator's UKM MUST be generated. The originator's UKM MUST be placed in the KeyAgreeRecipientInfo ukm OCTET STRING.

3) KEA MUST be used to generate the pairwise KEK based on the originator's UKM, originator's private KEA key, recipient's 128-byte public KEA key and recipient's 128-byte public KEA key used as the Rb value.

4) The SKIPJACK CEK MUST be wrapped using the KEA-generated pairwise KEK as input to the FORTEZZA 80-bit wrap function. The FORTEZZA 80-bit wrap function takes the 80-bit SKIPJACK CEK along with a 16-bit integrity check value and produces a 96-bit result using the KEA-generated pairwise KEK.

5) A new RecipientEncryptedKey SEQUENCE MUST be constructed.

6) The value of the subjectKeyIdentifier extension from the recipient's KEA X.509 v3 certificate MUST be placed in the RecipientEncryptedKey rid rKeyId subjectKeyIdentifier field. The KeyAgreeRecipientIdentifier CHOICE MUST be rKeyId. The date and other fields MUST be absent from the RecipientEncryptedKey rid rKeyId SEQUENCE.

7) The wrapped SKIPJACK CEK MUST be placed in the RecipientEncryptedKey encryptedKey OCTET STRING.

8) The recipient's RecipientEncryptedKey MUST be the only RecipientEncryptedKey present in the KeyAgreeRecipientInfo recipientEncryptedKeys SEQUENCE OF RecipientEncryptedKey.

9) The RecipientInfo containing the recipient's KeyAgreeRecipientInfo MUST be included in the EnvelopedData RecipientInfos SET OF RecipientInfo.

4.2.2. SKIPJACK CEK Unwrap Process

This section describes the recipient processing using KEA to generate the SKIPJACK KEK and the subsequent decryption of the SKIPJACK CEK.

1) Compliant software MUST be capable of processing EnvelopedData objects constructed using both the shared and the unique originator UKM options. To support the shared UKM option, the receiving software MUST be capable of searching for the recipient's RecipientEncryptedKey in a KeyAgreeRecipientInfo recipientEncryptedKeys SEQUENCE OF RecipientEncryptedKey. To support the unique UKM option, the receiving software MUST be capable of searching for the recipient's RecipientEncryptedKey in the EnvelopedData recipientInfos SET OF RecipientInfo, with each RecipientInfo containing exactly one RecipientEncryptedKey. For each RecipientEncryptedKey, if the rid rkeyId CHOICE is present, then the receiving software MUST attempt to match the value of the subjectKeyIdentifier extension from the recipient's KEA X.509 v3 certificate with the RecipientEncryptedKey rid rKeyId subjectKeyIdentifier field. If the rid issuerAndSerialNumber CHOICE is present, then the receiving software MUST attempt to match the values of the issuer name and serial number from the recipient's KEA X.509 v3 certificate with the RecipientEncryptedKey rid issuerAndSerialNumber field.

2) The receiving software MUST extract the originator's UKM from the ukm OCTET STRING contained in the same KeyAgreeRecipientInfo that includes the recipient's RecipientEncryptedKey.

3) The receiving software MUST locate the originator's KEA X.509 v3 certificate identified by the originator field contained in the same KeyAgreeRecipientInfo that includes the recipient's RecipientEncryptedKey.

4) KEA MUST be used to generate the pairwise KEK based on the originator's UKM, originator's 128-byte public KEA key (extracted from originator's KEA X.509 v3 certificate), recipient's private KEA key (associated with recipient's KEA X.509 v3 certificate identified by the RecipientEncryptedKey rid field) and the originator's 128-byte public KEA key used as the Rb value.

5) The SKIPJACK CEK MUST be unwrapped using the KEA-generated pairwise KEK as input to the FORTEZZA 80-bit unwrap function.

6) The unwrapped 80-bit SKIPJACK CEK resulting from the SKIPJACK CEK unwrap process and the 8-byte IV obtained from the EnvelopedData encryptedContentInfo contentEncryptionAlgorithm parameters field are used as inputs to the SKIPJACK content decryption process to decrypt the EnvelopedData encryptedContent.

4.3. "Previously Distributed" Symmetric KEK

This section describes the conventions for using SKIPJACK with the CMS enveloped-data content type to support "previously distributed" symmetric KEKs. When a "previously distributed" symmetric KEK is used to wrap the SKIPJACK CEK, then the RecipientInfo KEKRecipientInfo CHOICE MUST be used. The methods used to generate and distribute the symmetric KEK are beyond the scope of this document.

The KEKRecipientInfo fields MUST be populated as specified in [CMS] Section 6.2.3, "KEKRecipientInfo Type". The KEKRecipientInfo keyEncryptionAlgorithm algorithm field MUST be the id-fortezzaWrap80 OID indicating that the FORTEZZA 80-bit wrap function is used to wrap the 80-bit SKIPJACK CEK. The KEKRecipientInfo keyEncryptionAlgorithm parameters field MUST be absent. The KEKRecipientInfo encryptedKey field MUST include the SKIPJACK CEK wrapped using the "previously distributed" symmetric KEK as input to the FORTEZZA 80-bit wrap function.

5. Encrypted-data Conventions

The CMS encrypted-data content type consists of an encrypted content, but no recipient information. The method for conveying the SKIPJACK CEK required to decrypt the encrypted-data encrypted content is beyond the scope of this document. Compliant software MUST meet the requirements for constructing an encrypted-data content type stated [CMS] Section 8, "Encrypted-data Content Type". [CMS] Section 8 should be studied before reading this section, because this section does not repeat the [CMS] text.

The encrypted-data content type is ASN.1 encoded using the EncryptedData syntax. The fields of the EncryptedData syntax must be populated as follows:

The EncryptedData version MUST be set according to [CMS] Section 8.

The EncryptedData encryptedContentInfo contentEncryptionAlgorithm algorithm field MUST be the id-fortezzaConfidentialityAlgorithm OID. The EncryptedData encryptedContentInfo contentEncryptionAlgorithm parameters field MUST include the random 8-byte IV used as the input to the content encryption process.

The EncryptedData unprotectedAttrs MAY be present.

6. FORTEZZA 80-bit Wrap Function

The United States Government has not published the description of the FORTEZZA 80-bit wrap function.

7. SMIMECapabilities Attribute Conventions

RFC 2633 [MSG], Section 2.5.2 defines the SMIMECapabilities signed attribute (defined as a SEQUENCE of SMIMECapability SEQUNCEs) to be used to specify a partial list of algorithms that the software announcing the SMIMECapabilities can support. When constructing a signedData object, compliant software MAY include the SMIMECapabilities signed attribute announcing that it supports the KEA and SKIPJACK algorithms.

The SMIMECapability SEQUENCE representing KEA MUST include the id-kEAKeyEncryptionAlgorithm OID in the capabilityID field and MUST include a KeyWrapAlgorithm SEQUENCE in the parameters field. The algorithm field of KeyWrapAlgorithm MUST be the id-fortezzaWrap80 OID. The parameters field of KeyWrapAlgorithm MUST be absent. The SMIMECapability SEQUENCE for KEA SHOULD be included in the key management algorithms portion of the SMIMECapabilities list. The SMIMECapability SEQUENCE representing KEA MUST be DER-encoded as the following hexadecimal string:

```
3018 0609 6086 4801 6502 0101 1830 0b06 0960 8648 0165 0201 0117
```

The SMIMECapability SEQUENCE representing SKIPJACK MUST include the id-fortezzaConfidentialityAlgorithm OID in the capabilityID field and the parameters field MUST be absent. The SMIMECapability SEQUENCE for SKIPJACK SHOULD be included in the symmetric encryption algorithms portion of the SMIMECapabilities list. The SMIMECapability SEQUENCE representing SKIPJACK MUST be DER-encoded as the following hexadecimal string:

```
300b 0609 6086 4801 6502 0101 0400
```

8. Object Identifier Definitions

The following OIDs are specified in [INFO], but are repeated here for the reader's convenience:

```
id-keyExchangeAlgorithm OBJECT IDENTIFIER ::= {joint-iso-ccitt(2)
country(16) us(840) organization(1) gov(101) dod(2) infosec(1)
algorithms(1) keyExchangeAlgorithm (22)}
```

```
id-fortezzaWrap80 OBJECT IDENTIFIER ::= {joint-iso-ccitt(2)
country(16) us(840) organization(1) gov(101) dod(2) infosec(1)
algorithms(1) fortezzaWrap80Algorithm (23)}

id-kEAKeyEncryptionAlgorithm OBJECT IDENTIFIER ::= {joint-iso-
ccitt(2) country(16) us(840) organization(1) gov(101) dod(2)
infosec(1) algorithms(1) kEAKeyEncryptionAlgorithm (24)}

id-fortezzaConfidentialityAlgorithm OBJECT IDENTIFIER ::= {joint-
iso-ccitt(2) country(16) us(840) organization(1) gov(101) dod(2)
infosec(1) algorithms(1) fortezzaConfidentialityAlgorithm (4)}
```

As specified in [USSUP1], when the id-fortezzaConfidentialityAlgorithm OID is present in the AlgorithmIdentifier algorithm field, then the AlgorithmIdentifier parameters field MUST be present and MUST include the SKIPJACK IV ASN.1 encoded using the following syntax:

```
Skipjack-Parm ::= SEQUENCE { initialization-vector   OCTET STRING }
```

Note: [CMS] Section 2, "General Overview" describes the ASN.1 encoding conventions for the CMS content types including the enveloped-data and encrypted-data content types in which the id-fortezzaConfidentialityAlgorithm OID and parameters will be present.

References

[CMS] Housley, R., "Cryptographic Message Syntax", RFC 2630, June 1999.

[KEA] Housley, R. and W. Polk, "Representation of Key Exchange Algorithm (KEA) Keys in Internet X.509 Public Key Infrastructure Certificates", RFC 2528, March 1999.

[INFO] Registry of INFOSEC Technical Objects, 22 July 1999.

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

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

[SJ-KEA] SKIPJACK and KEA Algorithm Specifications, Version 2.0, http://csrc.nist.gov/encryption/skipjack-kea.htm.

[USSUP1] Allied Communication Publication 120 (ACP120) Common Security Protocol (CSP) United States (US) Supplement No. 1, June 1998; http://www.armadillo.huntsville.al.us/Fortezza_docs/missi2.html#specs.

Acknowledgments

The following people have made significant contributions to this memo: David Dalkowski, Phillip Griffin, Russ Housley, Pierce Leonberger, Rich Nicholas, Bob Relyea and Jim Schaad.

Author's Address

John Pawling
Wang Government Services, Inc. (WGSI),
A Getronics Company
141 National Business Pkwy, Suite 210
Annapolis Junction, MD 20701

Phone: (301) 939-2739
(410) 880-6095
EMail: john.pawling@wang.com

Full Copyright Statement

Acknowledgement

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