Network Working Group
Request For Comments: 1848
Category: Standards Track

S. Crocker
CyberCash, Inc.
N. Freed
Innosoft International, Inc.
J. Galvin
S. Murphy
Trusted Information Systems

October 1995

MIME Object Security Services

Status of this Memo

Abstract

This document defines MIME Object Security Services (MOSS), a protocol that uses the multipart/signed and multipart/encrypted framework [7] to apply digital signature and encryption services to MIME objects. The services are offered through the use of end-to-end cryptography between an originator and a recipient at the application layer. Asymmetric (public key) cryptography is used in support of the digital signature service and encryption key management. Symmetric (secret key) cryptography is used in support of the encryption service. The procedures are intended to be compatible with a wide range of public key management approaches, including both ad hoc and certificate-based schemes. Mechanisms are provided to support many public key management approaches.

Table of Contents

1. Introduction 3
2. Applying MIME Object Security Services 4
2.1 Digital Signature Service 4
2.1.1 Canonicalization 5
2.1.2 Digital Signature Control Information 7
2.1.2.1 Version: 8
2.1.2.2 Originator-ID: 8
2.1.2.3 MIC-Info: 8
2.1.3 application/moss-signature Content Type Definition 9
2.1.4 Use of multipart/signed Content Type 10
2.2 Encryption Service 11

2.2.1 Encryption Control Information 12
2.2.1.1 DEK-Info: 13
2.2.1.2 Recipient-ID: 14
2.2.1.3 Key-Info: 14
2.2.2 application/moss-keys Content Type Definition 15
2.2.3 Use of multipart/encrypted Content Type 16
3. Removing MIME Object Security Services 17
3.1 Digital Signature Service 18
3.1.1 Preparation 18
3.1.2 Verification 19
3.1.3 Results 19
3.2 Encryption Service 20
3.2.1 Preparation 20
3.2.2 Decryption 20
3.2.3 Results 21
4. Identifying Originators, Recipients, and Their Keys 21
4.1 Name Forms 23
4.1.1 Email Addresses 23
4.1.2 Arbitrary Strings 23
4.1.3 Distinguished Names 23
4.2 Identifiers 24
4.2.1 Email Address 25
4.2.2 Arbitrary String 25
4.2.3 Distinguished Name 26
4.2.4 Public Key 26
4.2.5 Issuer Name and Serial Number 27
5. Key Management Content Types 27
5.1 application/mosskey-request Content Type Definition 28
5.2 application/mosskey-data Content Type Definition 29
6. Examples 31
6.1 Original Message Prepared for Protection 31
6.2 Sign Text of Original Message 32
6.3 Sign Headers and Text of Original Message 32
6.4 Encrypt Text of a Message 33
6.5 Encrypt the Signed Text of a Message 35
6.6 Protecting Audio Content 37
6.6.1 Sign Audio Content 37
6.6.2 Encrypt Audio Content 37
7. Observations 38
8. Comparison of MOSS and PEM Protocols 39
9. Security Considerations 41
10. Acknowledgements 41
11. References 41
12. Authors' Addresses 43
Appendix A: Collected Grammar 44
Appendix B: Imported Grammar 47

1. Introduction

MIME [2], an acronym for "Multipurpose Internet Mail Extensions", defines the format of the contents of Internet mail messages and provides for multi-part textual and non-textual message bodies. An Internet electronic mail message consists of two parts: the headers and the body. The headers form a collection of field/value pairs structured according to STD 11, RFC 822 [1], whilst the body, if structured, is defined according to MIME. MIME does not provide for the application of security services.

PEM [3-6], an acronym for "Privacy Enhanced Mail", defines message encryption and message authentication procedures for text-based electronic mail messages using a certificate-based key management mechanism. The specifications include several features that are easily and more naturally supported by MIME, for example, the transfer encoding operation, the Content-Domain header, and the support services specified by its Part IV [6]. The specification is limited by specifying the application of security services to text messages only.

MOSS is based in large part on the PEM protocol as defined by RFC 1421. Many of PEMs features and most of its protocol specification are included here. A comparison of MOSS and PEM may be found in Section 8.

In order to make use of the MOSS services, a user (where user is not limited to being a human, e.g., it could be a process or a role) is required to have at least one public/private key pair. The public key must be made available to other users with whom secure communication is desired. The private key must not be disclosed to any other user.

An originator’s private key is used to digitally sign MIME objects; a recipient would use the originator’s public key to verify the digital signature. A recipient’s public key is used to encrypt the data encrypting key that is used to encrypt the MIME object; a recipient would use the corresponding private key to decrypt the data encrypting key so that the MIME object can be decrypted.

As long as the private keys are protected from disclosure, i.e., the private keys are accessible only to the user to whom they have been assigned, the recipient of a digitally signed message will know from whom the message was sent and the originator of an encrypted message will know that only the intended recipient is able to read it. For assurance, the ownership of the public keys used in verifying digital signatures and encrypting messages should be verified. A stored public key should be protected from modification.

The framework defined in [7] provides an embodiment of a MIME object and its digital signature or encryption keys. When used by MOSS the framework provides digital signature and encryption services to single and multi-part textual and non-textual MIME objects.

2. Applying MIME Object Security Services

The application of the MOSS digital signature service requires the following components.

(1) The data to be signed.

(2) The private key of the originator.

The data to be signed is prepared according to the description below. The digital signature is created by generating a hash of the data and encrypting the hash value with the private key of the originator. The digital signature, some additional ancillary information described below, and the data are then embodied in a multipart/signed body part. Finally, the multipart/signed body part may be transferred to a recipient or processed further, for example, it may be encrypted.

The application of the MOSS encryption service requires the following components.

(1) The data to be encrypted.

(2) A data encrypting key to encrypt the data.

(3) The public key of the recipient.

The data to be encrypted is prepared according to the description below. The originator creates a data encrypting key and encrypts the data. The recipient’s public key is used to encrypt the data encrypting key. The encrypted data, the encrypted data encrypting key, and some additional ancillary information described below are then embodied in a multipart/encrypted body part, ready to be transferred to a recipient or processed further, for example, it may be signed.

The next two sections describe the digital signature and encryption services, respectively, in detail.

2.1. Digital Signature Service

The MOSS digital signature service is applied to MIME objects, specifically a MIME body part. The MIME body part is created

according to a local convention and then made available to the digital signature service.

The following sequence of steps comprises the application of the digital signature service.

(1) The body part to be signed must be canonicalized.

(2) The digital signature and other control information must be generated.

(3) The control information must be embodied in an appropriate MIME content type.

(4) The control information body part and the data body part must be embodied in a multipart/signed content type.

Each of these steps is described below.

2.1.1. Canonicalization

The body part must be converted to a canonical form that is uniquely and unambiguously representable in at least the environment where the digital signature is created and the environment where the digital signature will be verified, i.e., the originator and recipient's environment, respectively. This is required in order to ensure that both the originator and recipient have the same data with which to calculate the digital signature; the originator needs to be able to create the digital signature value while the recipient needs to be able to compare a re-computed value with the received value. If the canonical form is representable on many different host computers, the signed data may be forwarded by recipients to additional recipients, who will also be able to verify the original signature. This service is called forwardable authentication.

The canonicalization transformation is a two step process. First, the body part must be converted to a form that is unambiguously representable on as many different host computers as possible. Second, the body part must have its line delimiters converted to a unique and unambiguous representation.

The representation chosen to satisfy the first step is 7bit, as defined by MIME; the high order bit of each octet of the data to be

signed must be zero. A MIME body part is comprised of two parts: headers and content. Since the headers of body parts are already required to be represented in 7bit, this step does not require changes to the headers. This step requires that if the content is not already 7bit then it must be encoded with an appropriate MIME content transfer encoding and a Content-Transfer-Encoding: header must be added to the headers. For example, if the content to be signed contains 8bit or binary data, the content must be encoded with either the quoted-printable or base64 encoding as defined by MIME.

IMPLEMENTORS NOTE: Since the MIME standard explicitly disallows nested content transfer encodings, i.e., the content types multipart and message may not themselves be encoded, the 7bit transformation requires each nested body part to be individually encoded in a 7bit representation. Any valid MIME encoding, e.g., quoted-printable or base64, may be used and, in fact, a different encoding may be used on each of the non-7bit body parts.

Representing all content types in a 7bit format transforms them into text-based content types. However, text-based content types present a unique problem. In particular, the line delimiter used for a text-based content type is specific to a local environment; different environments use the single character carriage-return (<CR>), the single character line-feed (<LF>), or the two character sequence "carriage-return line-feed (<CR><LF>)".

The application of the digital signature service requires that the same line delimiter be used by both the originator and the recipient. This document specifies that the two character sequence "<CR><LF>" must be used as the line delimiter. Thus, the second step of the canonicalization transformation includes the conversion of the local line delimiter to the two character sequence "<CR><LF>".

The conversion to the canonical line delimiter is only required for the purposes of computing the digital signature. Thus, originators must apply the line delimiter conversion before computing the digital signature but must transfer the data without the line delimiter conversion. Similarly, recipients must apply the line delimiter conversion before computing the digital signature.

NOTE: An originator can not transfer the content with the line delimiter conversion intact because the conversion process is not idempotent. In particular, SMTP servers may themselves convert the line delimiter to a local line delimiter, prior to the message being delivered to the recipient. Thus, a recipient has no way of knowing if the conversion is present or not. If the recipient applies the conversion to a content in which it is already present, the resulting content may have two line delimiters

present, which would cause the verification of the signature to fail.

IMPLEMENTORS NOTE: Implementors should be aware that the conversion to a 7bit representation is a function that is required in a minimally compliant MIME user agent. Further, the line delimiter conversion required here is distinct from the same conversion included in that function. Specifically, the line delimiter conversion applied when a body part is converted to a 7bit representation (transfer encoded) is performed prior to the application of the transfer encoding. The line delimiter conversion applied when a body part is signed is performed after the body part is converted to 7bit (transfer encoded). Both line delimiter conversions are required.

2.1.2. Digital Signature Control Information

The application of the digital signature service generates control information which includes the digital signature itself. The syntax of the control information is that of a set of RFC 822 headers, except that the folding of header values onto continuation lines is explicitly forbidden. Each header and value pair generated by the digital signature service must be output on exactly one line.

The complete set of headers generated by the digital signature service is as follows.

Version:
: indicates which version of the MOSS protocol the remaining headers represent.

Originator-ID:
: indicates the private key used to create the digital signature and the corresponding public key to be used to verify it.

MIC-Info:
: contains the digital signature value.

Each invocation of the digital signature service must emit exactly one Version: header and at least one pair of Originator-ID: and MIC-Info: headers. The Version: header must always be emitted first. The Originator-ID: and MIC-Info: headers are always emitted in pairs in the order indicated. This specification allows an originator to generate multiple signatures of the data, presumably with different signature algorithms, and to include them all in the control information. The interpretation of the presence of multiple signatures is outside the scope of this specification except that a MIC-Info: header is always interpreted in the context of the

immediately preceding Originator-ID: header.

2.1.2.1. Version:

The version header is defined by the grammar token <version> as follows.

```
<version>  ::= "Version:" "5" CRLF
```

Its value is constant and MOSS implementations compliant with this specification must recognize only this value and generate an error if any other value is found.

2.1.2.2. Originator-ID:

The purpose of the originator header is two-fold: to directly identify the public key to be used to verify the digital signature and to indirectly identify the user who owns both it and its corresponding private key. Typically, a recipient is less interested in the actual public key value, although obviously the recipient needs the value to verify the signature, and more interested in identifying its owner. Thus, the originator header may convey either or both pieces of information:

the public key to be used to verify the signature

the name of the owner and which of the owner's public keys to use to verify the signature

The decision as to what information to place in the value rests entirely with the originator. The suggested value is to include both. Recipients with whom the originator has previously communicated will have to verify that the information presented is consistent with what is already known. New recipients will want all of the information, which they will need to verify prior to storing in their local database.

The originator header is defined by the grammar token <origid> as follows.

```
<origid>  ::= "Originator-ID:" <id> CRLF
```

The grammar token <id> is defined in Section 4.

2.1.2.3. MIC-Info:

The purpose of the Message Integrity Check (MIC) header is to convey the digital signature value. Its value is a comma separated list of

three arguments: the hash (or MIC) algorithm identifier, the signature algorithm identifier, and the digital signature.

The MIC header is defined by the grammar token <micinfo> as follows.

```
<micinfo>  ::= "MIC-Info:" <micalgid> "," <ikalgid> ","
                   <asymsignmic> CRLF
```

The grammar tokens for the MIC algorithms and identifiers (<micalgid>), signature algorithms and identifiers (<ikalgid>), and signed MIC formats (<asymsignmic>) are defined by RFC 1423. They are also reprinted in Appendix B.

IMPLEMENTORS NOTE: RFC 1423 is referenced by the PEM protocol, which includes support for symmetric signatures and key management. As a result, some of the grammar tokens defined there, for example, <ikalgid>, will include options that are not legal for this protocol. These options must be ignored and have not been included in the appendix.

2.1.3. application/moss-signature Content Type Definition

(1) MIME type name: application

(2) MIME subtype name: moss-signature

(3) Required parameters: none

(4) Optional parameters: none

(5) Encoding considerations: quoted-printable is always sufficient

(6) Security considerations: none

The "application/moss-signature" content type is used on the second body part of an enclosing multipart/signed. Its content is comprised of the digital signature of the data in the first body part of the enclosing multipart/signed and other control information required to verify that signature, as defined by Section 2.1.2. The label "application/moss-signature" must be used as the value of the protocol parameter of the enclosing multipart/signed; the protocol parameter must be present.

Part of the signature verification information will be the Message Integrity Check (MIC) algorithm(s) used during the signature creation process. The MIC algorithm(s) identified in this body part must match the MIC algorithm(s) identified in the micalg parameter of the enclosing multipart/signed. If it does (they do) not, a user agent

should identify the discrepancy to a user and it may choose to either halt or continue processing, giving precedence to the algorithm(s) identified in this body part.

An application/moss-signature body part is constructed as follows:

```
Content-Type: application/moss-signature

<mosssig>
```

where the grammar token <mosssig> is defined as follows.

```
<mosssig>       ::= <version> ( 1*<origasymflds> )

<version>       ::= "Version:" "5" CRLF

<origasymflds>  ::= <origid> <micinfo>

<origid>        ::= "Originator-ID:" <id> CRLF

<micinfo>       ::= "MIC-Info:" <micalgid> "," <ikalgid> ","
                    <asymsignmic> CRLF
```

The token <id> is defined in Section 4. All other tokens are defined in Section 2.1.2.3.

2.1.4. Use of multipart/signed Content Type

The definition of the multipart/signed content type in [7] specifies three steps for creating the body part.

(1) The body part to be digitally signed is created according to a local convention, for example, with a text editor or a mail user agent.

(2) The body part is prepared for the digital signature service according to the protocol parameter, in this case according to Section 2.1.1.

(3) The prepared body part is digitally signed according to the protocol parameter, in this case according to Section 2.1.2.

The multipart/signed content type is constructed as follows.

(1) The value of its required parameter "protocol" is set to "application/moss-signature".

(2) The signed body part becomes its first body part.

(3) Its second body part is labeled "application/moss-signature" and is filled with the control information generated by the digital signature service.

(4) The value of its required parameter "micalg" is set to the same value used in the MIC-Info: header in the control information. If there is more than one MIC-Info: header present the value is set to a comma separated list of values from the MIC-Info headers. The interpretation of the order of the list of values is outside the scope of this specification.

A multipart/signed content type with the MOSS protocol might look as follows:

```
Content-Type: multipart/signed;
  protocol="application/moss-signature";
  micalg="rsa-md5"; boundary="Signed Message"

--Signed Message
Content-Type: text/plain

This is some example text.

--Signed Message
Content-Type: application/moss-signature

Version: 5
Originator-ID: ID-INFORMATION
MIC-Info: RSA-MD5,RSA,SIGNATURE-INFORMATION
--Signed Message--
```

where ID-INFORMATION and SIGNATURE-INFORMATION are descriptive of the content that would appear in a real body part.

2.2. Encryption Service

The MOSS encryption service is applied to MIME objects, specifically a MIME body part. The MIME body part is created according to a local convention and then made available to the encryption service.

The following sequence of steps comprises the application of the encryption service.

(1) The body part to be encrypted must be in MIME canonical form.

(2) The data encrypting key and other control information must be generated.

(3) The control information must be embodied in an appropriate MIME content type.

(4) The control information body part and the encrypted data body part must be embodied in a multipart/encrypted content type.

The first step is defined by MIME. The latter three steps are described below.

2.2.1. Encryption Control Information

The application of the encryption service generates control information which includes the data encrypting key used to encrypt the data itself. The syntax of the control information is that of a set of RFC 822 headers, except that the folding of header values onto continuation lines is explicitly forbidden. Each header and value pair generated by the encryption service must be output on exactly one line.

First, the originator must retrieve the public key of the recipient. The retrieval may be from a local database or from a remote service. The acquisition of the recipient’s public key is outside the scope of the specification, although Section 5 defines one possible mechanism.

With the public key, the originator encrypts the data encrypting key according to the Key-Info: header defined below. The complete set of headers generated by the encryption service is as follows.

Version:
indicates which version of the MOSS protocol the remaining headers represent and is defined in Section 2.1.2.1.

DEK-Info:
indicates the algorithm and mode used to encrypt the data.

Recipient-ID:
indicates the public key used to encrypt the data encrypting key that was used to encrypt the data.

Key-Info:
contains data encrypting key encrypted with the recipient's public key.

Each invocation of the encryption service must emit exactly one Version: header, exactly one DEK-Info: header, and at least one pair of Recipient-ID: and Key-Info: headers. Headers are always emitted in the order indicated. The Recipient-ID: and Key-Info: headers are always emitted in pairs in the order indicated, one pair for each recipient of the encrypted data. A Key-Info: header is always interpreted in the context of the immediately preceding Recipient-ID: header.

IMPLEMENTORS NOTE: Implementors should always generate a Recipient-ID: and Key-Info header pair representing the originator of the encrypted data. By doing so, if an originator sends a message to a recipient that is returned undelivered, the originator will be able to decrypt the message and determine an appropriate course of action based on its content. If not, an originator will not be able to review the message that was sent.

2.2.1.1. DEK-Info:

The purpose of the data encrypting key information header is to indicate the algorithm and mode used to encrypt the data, along with any cryptographic parameters that may be required, e.g., initialization vectors. Its value is either a single argument indicating the algorithm and mode or a comma separated pair of arguments where the second argument carries any cryptographic parameters required by the algorithm and mode indicated in the first argument.

The data encrypting key information header is defined by the grammar token <dekinfo> as follows.

```
<dekinfo>  ::= "DEK-Info" ":" <dekalgid>
               [ "," <dekparameters> ] CRLF
```

The grammar tokens for the encryption algorithm and mode identifier (<dekalgid>) and the optional cryptographic parameters (<dekparameters>) are defined by RFC 1423. They are also reprinted in Appendix B.

2.2.1.2. Recipient-ID:

The purpose of the recipient header is to identify the private key that must be used to decrypt the data encrypting key that will be used to decrypt the data. Presumably the recipient owns the private key and thus is less interested in identifying the owner of the key and more interested in the private key value itself. Nonetheless, the recipient header may convey either or both pieces of information:

the public key corresponding to the private key to be used to decrypt the data encrypting key

the name of the owner and which of the owner's private keys to use to decrypt the data encrypting key

The decision as to what information to place in the value rests entirely with the originator. The suggested choice is to include just the public key. However, some recipients may prefer that originators not include their public key. How this preference is conveyed to and managed by the originator is outside the scope of this specification.

The recipient header is defined by the grammar token <recipid> as follows.

<recipid> ::= "Recipient-ID:" <id> CRLF

The grammar token <id> is defined in Section 4.

2.2.1.3. Key-Info:

The purpose of the key information header is to convey the encrypted data encrypting key. Its value is a comma separated list of two arguments: the algorithm and mode identifier in which the data encrypting key is encrypted and the encrypted data encrypting key.

The key information header is defined by the grammar token <asymkeyinfo> as follows.

<asymkeyinfo> ::= "Key-Info" ":" <ikalgid> "," <asymencdek> CRLF

The grammar tokens for the encryption algorithm and mode identifier (<ikalgid>) and the encrypted data encrypting key format (<asymsignmic>) are defined by RFC 1423. They are also reprinted in Appendix B.

IMPLEMENTORS NOTE: RFC 1423 is referenced by the PEM protocol, which includes support for symmetric signatures and key

management. As a result, some of the grammar tokens defined there, for example, <ikalgid>, will include options that are not legal for this protocol. These options must be ignored and have not been included in the appendix.

2.2.2. application/moss-keys Content Type Definition

(1) MIME type name: application

(2) MIME subtype name: moss-keys

(3) Required parameters: none

(4) Optional parameters: none

(5) Encoding considerations: quoted-printable is always sufficient

(6) Security considerations: none

The "application/moss-keys" content type is used on the first body part of an enclosing multipart/encrypted. Its content is comprised of the data encryption key used to encrypt the data in the second body part and other control information required to decrypt the data, as defined by Section 2.2.1. The label "application/moss-keys" must be used as the value of the protocol parameter of the enclosing multipart/encrypted; the protocol parameter must be present.

An application/moss-keys body part is constructed as follows:

```
Content-Type: application/moss-keys

<mosskeys>
```

where the <mosskeys> token is defined as follows.

```
<mosskeys>      ::= <version> <dekinfo> 1*<recipasymflds>

<version>       ::= "Version:" "5" CRLF

<dekinfo>       ::= "DEK-Info" ":" <dekalgid>
                    [ "," <dekparameters> ] CRLF

<recipasymflds> ::= <recipid> <asymkeyinfo>

<recipid>       ::= "Recipient-ID:" <id> CRLF

<asymkeyinfo>   ::= "Key-Info" ":" <ikalgid> "," <asymencdek> CRLF
```

The token <id> is defined in Section 4. The token <version> is defined in Section 2.1.2.1. All other tokens are defined in Section 2.2.1.3.

2.2.3. Use of multipart/encrypted Content Type

The definition of the multipart/encrypted body part in [7] specifies three steps for creating the body part.

(1) The body part to be encrypted is created according to a local convention, for example, with a text editor or a mail user agent.

(2) The body part is prepared for encryption according to the protocol parameter, in this case the body part must be in MIME canonical form.

(3) The prepared body part is encrypted according to the protocol parameter, in this case according to Section 2.2.1.

The multipart/encrypted content type is constructed as follows.

(1) The value of its required parameter "protocol" is set to "application/moss-keys".

(2) The first body part is labeled "application/moss-keys" and is filled with the control information generated by the encryption service.

(3) The encrypted body part becomes the content of its second body part, which is labeled "application/octet-stream".

A multipart/encrypted content type with the MOSS protocol might look as follows:

```
Content-Type: multipart/encrypted;
  protocol="application/moss-keys";
  boundary="Encrypted Message"

--Encrypted Message
Content-Type: application/moss-keys

Version: 5
DEK-Info: DES-CBC,DEK-INFORMATION
Recipient-ID: ID-INFORMATION
Key-Info: RSA,KEY-INFORMATION

--Encrypted Message
Content-Type: application/octet-stream

ENCRYPTED-DATA
--Encrypted Message--
```

where DEK-INFORMATION, ID-INFORMATION, and KEY-INFORMATION are descriptive of the content that would appear in a real body part.

3. Removing MIME Object Security Services

The verification of the MOSS digital signature service requires the following components.

(1) A recipient to verify the digital signature.

(2) A multipart/signed body part with two body parts: the signed data and the control information.

(3) The public key of the originator.

The signed data and control information of the enclosing multipart/signed are prepared according to the description below. The digital signature is verified by re-computing the hash of the data, decrypting the hash value in the control information with the originator's public key, and comparing the two hash values. If the two hash values are equal, the signature is valid.

The decryption of the MOSS encryption service requires the following components.

(1) A recipient to decrypt the data.

(2) A multipart/encrypted body part with two body parts: the encrypted data and the control information.

(3) The private key of the recipient.

The encrypted data and control information of the enclosing multipart/encrypted are prepared according to the description below. The data encrypting key is decrypted with the recipient's private key and used to decrypt the data.

The next two sections describe the digital signature and encryption services in detail, respectively.

3.1. Digital Signature Service

This section describes the processing steps necessary to verify the MOSS digital signature service. The definition of the multipart/signed body part in [7] specifies three steps for receiving it.

(1) The digitally signed body part and the control information body part are prepared for processing.

(2) The prepared body parts are made available to the digital signature verification process.

(3) The results of the digital signature verification process are made available to the user and processing continues with the digitally signed body part, as returned by the digital signature verification process.

Each of these steps is described below.

3.1.1. Preparation

The digitally signed body part (the data) and the control information body part are separated from the enclosing multipart/signed body part.

The control information is prepared by removing any content transfer encodings that may be present.

The digitally signed body part is prepared by leaving the content transfer encodings intact and canonicalizing the line delimiters according to Step 2 of Section 2.1.1.

3.1.2. Verification

First, the recipient must obtain the public key of the originator. The public key may be contained in the control information or it may be necessary for the recipient to retrieve the public key based on information present in the control information. The retrieval may be from a local database or from a remote service. The acquisition of the originator's public key is outside the scope of the specification, although Section 5 defines one possible mechanism.

With the public key, the recipient decrypts the hash value contained in the control information. Then, a new hash value is computed over the body part purported to have been digitally signed.

Finally, the two hash values are compared to determine the accuracy of the digital signature.

3.1.3. Results

There are two required components of the results of the verification process. The first is an indication as to whether a public key could be found that allows the hash values in the previous step to compare equal. Such an indication verifies only that the data received is the same data that was digitally signed.

The second indication identifies the owner of the public key who is presumably the holder of the private key that created the digital signature. The indication must include a testament as to the accuracy of the owner identification.

At issue is a recipient knowing who created the digital signature. In order for the recipient to know with certainty who digitally signed the message, the binding between the owner's name and the public key must have been verified by the recipient prior to the verification of the digital signature. The verification of the binding may have been completed offline and stored in a trusted, local database or, if the owner's name and public key are embodied in a certificate, it may be possible to complete it in realtime. See Section 5 for more information.

3.2. Encryption Service

This section describes the processing steps necessary to decrypt the MOSS encryption service. The definition of the multipart/encrypted body part in [7] specifies three steps for receiving it.

(1) The encrypted body part and the control information body part are prepared for processing.

(2) The prepared body parts are made available to the decryption process.

(3) The results of the decryption process are made available to the user and processing continues with the decrypted body part, as returned by the decryption process.

Each of these steps is described below.

3.2.1. Preparation

The encrypted body part (the data) and the control information body part are separated from the enclosing multipart/encrypted body part. The body parts are prepared for the decryption process by removing any content transfer encodings that may be present.

3.2.2. Decryption

First, the recipient must locate the encrypted data encrypting key in the control information. Each Recipient-ID: header is checked in order to see if it identifies the recipient or a public key of the recipient.

If it does, the immediately following Key-Info: header will contain the data encrypting key encrypted with the public key of the recipient. The recipient must use the corresponding private key to decrypt the data encrypting key.

The data is decrypted with the data encrypting key. The decrypted data will be a MIME object, a body part, ready to be processed by a MIME agent.

3.2.3. Results

If the recipient is able to locate and decrypt a data encrypting key, from the point of view of MOSS the decryption should be considered successful. An indication of the owner of the private key used to decrypt the data encrypting key must be made available to the user.

Ultimately, the success of the decryption is dependent on the ability of a MIME agent to continue processing with the decrypted body part.

4. Identifying Originators, Recipients, and Their Keys

In the PEM specifications, public keys are required to be embodied in certificates, an object that binds each public key with a distinguished name. A distinguished name is a name form that identifies the owner of the public key. The embodiment is issued by a certification authority, a role that is expected to be trustworthy insofar as the certification authority would have procedures to verify the identity of the owner prior to issuing the certificate.

In MOSS, a user is not required to have a certificate. The MOSS services require that the user have at least one public/private key pair. The MOSS protocol requires the digital signature and encryption services to emit Originator-ID: and Recipient-ID: headers, as appropriate. In the discussion above the actual value of these headers was omitted, having been relegated to this section. Although the value of each of these headers serves a distinct purpose, for simplicity the single grammar token <id> represents the value that may be assigned to either header.

One possible value for the Originator-ID: and Recipient-ID: headers is the public key values themselves. However, while it is true that the public keys alone could be exchanged and used by users to communicate, the values are, in fact, large and cumbersome. In addition, public keys would appear as a random sequence of characters and, as a result, would not be immediately consumable by human users.

> NOTE: It should be pointed out that a feature of being able to specify the public key explicitly is that it allows users to exchange encrypted, anonymous mail. In particular, receiving users will always know a message comes from the same originating user even if the real identity of the originating user is unknown.

Recognizing that the use of public keys is, in general, unsuitable for use by humans, MOSS allows other identifiers in Originator-ID: and Recipient-ID: headers. These other identifiers are comprised of two parts: a name form and a key selector.

The name form is chosen and asserted by the user who owns the public/private key pair. Three name forms are specified by this document. The use of a distinguished name is retained for compatibility with PEM (and compatibility with the X.500 Directory should it become a ubiquitous service). However, the Internet community has a great deal of experience with the use of electronic mail addresses as a name form. Also, arbitrary strings are useful to identify the owners of public keys when private name forms are used. Hence, email addresses and arbitrary strings are included as name forms to increase flexibility.

Since a user may have more than one public key and may wish to use the same name form for each public key, a name form is insufficient for uniquely identifying a public key. A unique "key selector" must be assigned to each public key. The combination of a name form and the key selector uniquely identifies a public key. Throughout this document, this combination is called an identifier. There are 5 identifiers specified by this document.

> NOTE: In the simplest case, key selectors will be assigned by the owners of the public/private key pairs. This works best when users generate their own key pairs for personal use, from which they distribute their public key to others asserting by declaration that the public key belongs to them. When the assertion that the public key belongs to them is made by a third party, for example when a certification authority issues a certificate to a user according to [4], the key selector may be assigned by that third party.

The value of the key selector must be unique with respect to the name form with which it forms an identifier. Although the same key selector value may be used by more than one name form it must not be used for two different keys with the same name form. When considered separately, neither a name form nor a key selector is sufficient for identifying the public key to be used. Either could be used to determine a set of public keys that may be tried in turn until the desired public key is identified.

With a public/private key pair for one’s self and software that is MOSS aware, an originating user may digitally sign arbitrary data and send it to one or more recipients. With the public keys of the recipients, a user may encrypt the data so that only the intended recipients can decrypt and read it. With the name forms assigned to the public keys, originators and recipients can easily recognize their peers in a communication.

In the next section the 3 name forms are described in detail. Following that is the specification of the 5 identifiers.

4.1. Name Forms

There are 3 name forms specified by this document: email addresses, distinguished names, and arbitrary strings.

4.1.1. Email Addresses

The email address (grammar token <emailstr>) used must be a valid RFC822 address, which is defined in terms of one of the two grammar tokens <addr-spec> or <route-addr>. The grammar for these two tokens is included in the Appendix as a convenience; the definitive source for these tokens is necessarily RFC822 [1].

```
<emailstr>      ::= <addr-spec> / <route-addr>
                    ; an electronic mail address as defined by
                    ; one of these two tokens from RFC822
```

For example, the strings "crocker@tis.com", "galvin@tis.com", "murphy@tis.com", and "ned@innosoft.com" are all email addresses.

4.1.2. Arbitrary Strings

The arbitrary string (grammar token <string>) must have a length of at least 1. There are no other restrictions on the value chosen.

```
<string>        ::= ; a non-null sequence of characters
```

For example, the string

```
the SAAG mailing list maintainer
```

is an arbitrary string.

4.1.3. Distinguished Names

The distinguished name (grammar token <dnamestr>) must be constructed according to the guidelines of the X.500 Directory. The actual syntax of the distinguished name is outside the scope of this specification. However, RFC1422, for example, specifies syntactic restrictions based on its choice of a certification hierarchy for certificates.

For the purposes of conveying a distinguished name from an originator to a recipient, it must be ASN.1 encoded and then printably encoded according to the base64 encoding defined by MIME.

```
<dnamestr>      ::= <encbin>
                    ; a printably encoded, ASN.1 encoded
                    ; distinguished name (as defined by the 'Name'
                    ; production specified in X.501 [8])
```

For example,

```
/Country Name=US
/State or Province Name=MD
/Organization Name=Trusted Information Systems
/Organizational Unit Name=Glenwood
/Common Name=James M. Galvin/
```

is a distinguished name in a user friendly format (line breaks and leading spaces present only to improve readability). When encoded, it would appear as follows (line breaks present only to improve readability):

```
MG0xCzAJBgNVBAYTAlVTMQswCQYDVQQIEwJNRDEkMCIGA1UEChMbVHJ1c3RlZCBJ
bmZvcm1hdGlvbiBTeXN0ZW1zMREwDwYDVQQLEwhHbGVud29vZDEYMBYGA1UEAxMP
SmFtZXMgTS4gR2Fsdmlu
```

4.2. Identifiers

There are 5 types of identifiers specified by this document:

email address identifiers

arbitrary string identifiers

distinguished name identifiers

the public keys themselves

issuer name serial number pairs from a certificate

All of these have approximately the same structure (except issuer name and serial number which has 'TYPE, STRING, KEYSEL' for historical reasons):

TYPE, KEYSEL, STRING

The TYPE field is a literal string chosen from the set "EN", "STR", "DN", "PK", and "IS", one for each of the possible identifiers.

The KEYSEL field is used to distinguish between the multiple public keys that may be associated with the name form in the STRING field. Its value must be unique with respect to all other key selectors used

with the same name form. An example would be to use a portion (low-order 16 or 32 bits) or all of the actual public key used.

The STRING field is the name form and has a different syntax according to the value of the TYPE field.

The identifier used in each of the originator and recipient fields is described by the following grammar. The definition of the key selector token is included here since it used by several of the identifiers below.

```
<id>            ::=   <id-email> / <id-string>    / <id-dname>
                    / <id-publickey> / <id-issuer>

<keysel>        ::= 1*<hexchar>
                    ; hex dump of a non-null sequence of octets
```

Each of the identifier name forms is described below.

4.2.1. Email Address

The email address identifier has the following syntax.

```
<id-email>      ::= "EN"  "," <keysel> "," <emailstr> CRLF
```

The syntax of the token <emailstr> is defined in Section 4.1.1.

For example:

```
EN,1,galvin@tis.com
```

is an email address identifier.

4.2.2. Arbitrary String

The arbitrary string identifier has the following syntax.

```
<id-string>     ::= "STR" "," <keysel> "," <string> CRLF
```

The syntax of the token <string> is defined in Section 4.1.2.

For example:

```
STR,1,The SAAG mailing list maintainer
```

is an arbitrary string identifier.

4.2.3. Distinguished Name

The distinguished name identifier has the following syntax.

```
<id-dname>      ::= "DN"  "," <keysel> "," <dnamestr> CRLF
```

The syntax of the token <dnamestr> is defined in Section 4.1.3.

For example (line breaks present only to improve readability):

```
DN,1,MG0xCzAJBgNVBAYTAlVTMQswCQYDVQQIEwJNRDEkMCIGA1UEChMbVHJ1c3R
lZCBJbmZvcm1hdGlvbiBTeXN0ZW1zMREwDwYDVQQLEwhHbGVud29vZDEYMBYGA1U
EAxMPSmFtZXMgTS4gR2Fsdmlu
```

is a distinguished name identifier.

4.2.4. Public Key

The public key identifier has the following syntax.

```
<id-publickey>  ::= "PK"  "," <publickey> [ "," <id-subset> ] CRLF

<publickey>     ::= <encbin>
                    ; a printably encoded, ASN.1 encoded public
                    ; key (as defined by the
                    ; 'SubjectPublicKeyInfo' production specified
                    ; in X.509 [9])

<id-subset>     ::= <id-email> / <id-string> / <id-dname>
```

The production SubjectPublicKeyInfo is imported from the X.500 Directory from the certificate object. It is currently the best choice for a general purpose public key encoding.

For example, (line breaks present only to improve readability):

```
PK,MHkwCgYEVQgBAQICAwADawAwaAJhAMAHQ45ywA357G4fqQ61aoC1fO6BekJmG
4475mJkwGIUxvDkwuxe/EFdPkXDGBxzdGrW1iuh5K8kl8KRGJ9wh1HU4TrghGdhn
0Lw8gG67Dmb5cBhY9DGwq0CDnrpKZV3cQIDAQAB
```

is a public key identifier without the optional <id-subset>.

In normal usage, the token <id-subset> is expected to be present. It represents a mechanism by which an identifier (name form and key selector) can be associated with a public key. Recipients of a public key identifier must take care to verify the accuracy of the purported association. If they do not, it may be possible for a malicious originator to assert an identifier that accords the

originator unauthorized privileges. See Section 5.2 for more details.

For example, (line breaks present only to improve readability):

```
PK,MHkwCgYEVQgBAQICAwADawAwaAJhAMAHQ45ywA357G4fqQ61aoC1fO6BekJmG
4475mJkwGIUxvDkwuxe/EFdPkXDGBxzdGrW1iuh5K8kl8KRGJ9wh1HU4TrghGdhn
0Lw8gG67Dmb5cBhY9DGwq0CDnrpKZV3cQIDAQAB,EN,2,galvin@tis.com
```

is a public key identifier with the optional <id-subset>.

4.2.5. Issuer Name and Serial Number

The issuer name and serial number identifier has the following syntax.

```
<id-issuer>     ::= "IS"  "," <dnamestr>  "," <serial> CRLF

<serial>        ::= 1*<hexchar>
                        ; hex dump of a certificate serial number
```

The <id-issuer> identifier is included for compatibility with the ID-ASymmetric fields defined in [3] (and compatibility with X.500 Directory certificates should they become ubiquitously available). Its syntax was chosen such that the older fields are easily converted to this new form by prefixing the old value with "IS" (and replacing the field name of [3] with an appropriate new ID field name). For example, (line breaks present only to improve readability):

```
IS,MFMxCzAJBgNVBAYTAlVTMQswCQYDVQQIEwJNRDEkMCIGA1UEChMbVHJ1c3
RlZCBJbmZvcm1hdGlvbiBTeXN0ZW1zMREwDwYDVQQLEwhHbGVud29vZA==,02
```

is an issuer name and serial number identifier according to MOSS, while

```
MFMxCzAJBgNVBAYTAlVTMQswCQYDVQQIEwJNRDEkMCIGA1UEChMbVHJ1c3
RlZCBJbmZvcm1hdGlvbiBTeXN0ZW1zMREwDwYDVQQLEwhHbGVud29vZA==,02
```

is an issuer name and serial number identifier according to PEM.

5. Key Management Content Types

This document defines two key management content types: one for requesting cryptographic key material and one for sending cryptographic key material. Since MOSS depends only on the existence of public/private key pairs, these content types provide a means for conveying public keys and an assertion as to the identity of the owner. In addition, in order to be compatible with the certificate-

base key management system proposed by RFC 1422, the content types may also be used to convey certificate and certificate revocation list material.

The functions defined here are based on the exchange of body parts. In particular, a user would send a message containing at least one application/mosskey-request content, as defined below. In response, a user would expect to receive a message containing at least one application/mosskey-data content, as defined below. MIME provides a convenient framework for a user to send several request body parts and to receive several data (response) body parts in one message.

5.1. application/mosskey-request Content Type Definition

(1) MIME type name: application

(2) MIME subtype name: mosskey-request

(3) Required parameters: none

(4) Optional parameters: none

(5) Encoding considerations: quoted-printable is always sufficient

(6) Security Considerations: none

The content of this body part corresponds to the following production.

```
<request>       ::= <version>
                    ( <subject> / <issuer> / <certification> )

<version>       ::= "Version:" "5" CRLF

<subject>       ::= "Subject:" <id> CRLF

<issuer>        ::= "Issuer:" <id> CRLF

<certification> ::= "Certification:" <encbin> CRLF
```

A user would use this content type to specify needed cryptographic key information. The message containing this content type might be directed towards an automatic or manual responder, which may be mail-based, depending on the local implementation and environment. The application/mosskey-request content type is an independent body part because it is entirely independent of any other body part.

If the application/mosskey-request content contains a Certification: field it requests certification of the self-signed certificate in the field value. If the content contains an Issuer: field it requests the Certificate Revocation List (CRL) chain beginning with the CRL of the issuer identified in the field value. If the content contains a Subject: field it requests either the public key of the subject or a certificate chain beginning with the subject identified in the field value, or both if both exist.

The Subject: and Issuer: fields each contain a value of type <id>, which is defined in Section 4.

One possible response to receiving an application/mosskey-request body part is to construct and return an application/mosskey-data body part. When returning public keys, certificate chains, and certificate revocation list chains, if there exists more than one, several application/mosskey-data body parts are to be returned in the reply message, one for each.

5.2. application/mosskey-data Content Type Definition

The principal objective of this content type is to convey cryptographic keying material from a source to a destination. This might be in response to the receipt of an application/mosskey-request content type or it might be in anticipation of receiving an application/mosskey-request if it is not sent, e.g., it may be combined with a multipart/signed object by an originator to ensure that a recipient has the cryptographic keying material necessary to verify the signature. When combined with other content types, the processing by a recipient is enhanced if the application/mosskey-data content type is positioned in its enclosing content type prior to the content types that will make use of its cryptographic keying material.

However, no explicit provision is made in this document for determining the authenticity or accuracy of the data being conveyed. In particular, when a public key and its identifier is conveyed, there is nothing to prevent the source or an interloper along the path from the source to the destination from substituting alternate values for either the public key or the identifier.

It is incumbent upon a recipient to verify the authenticity and accuracy of the data received in this way prior to its use. This problem can be addressed by the use of certificates, since a certification hierarchy is a well-defined mechanism that conveniently supports the automatic verification of the data. Alternatively, the source of the application/mosskey-data body part could digitally sign it. In this way, if the destination believes that a correct source’s

public key is available locally and if the destination believes the source would convey accurate data, then the contents of the application/mosskey-data from the source could be believed to be accurate.

NOTE: Insofar as a certificate represents a mechanism by which a third party vouches for the binding between a name and a public key, the signing of an application/mosskey-data body part is a similar mechanism.

(1) MIME type name: application

(2) MIME subtype name: mosskey-data

(3) Required parameters: none

(4) Optional parameters: none

(5) Encoding considerations: quoted-printable is always sufficient.

(6) Security Considerations: none

The content of this body part corresponds to the following production.

```
<mosskeydata>   ::= <version>
                    ( <publickeydata> / <certchain> / <crlchain> )

<version>       ::= "Version:" "5" CRLF

<publickeydata> ::= "Key:" "PK" "," <publickey> ","
                    <id-subset> CRLF

<certchain>     ::= <cert> *( [ <crl> ] <cert> )

<crlchain>      ::= 1*( <crl> [ <cert> ] )

<cert>          ::= "Certificate:" <encbin> CRLF

<crl>           ::= "CRL:" <encbin> CRLF
```

This content type is used to transfer public keys, certificate chains, or Certificate Revocation List (CRL) chains. The information in the body part is entirely independent of any other body part. (Note that the converse is not true: the validity of a protected body part cannot be determined without the proper public keys, certificates, or current CRL information.) As such, the application/mosskey-data content type is an independent body part.

The <publickeydata> production contains exactly one public key. It is used to bind a public key with its corresponding name form and key selector. It is recommended that when responders are returning this information that the enclosing body part be digitally signed by the responder in order to protect the information. The <id-subset> token is defined in Section 4.2.4.

The <certchain> production contains one certificate chain. A certificate chain starts with the requested certificate and continues with the certificates of subsequent issuers. Each issuer certificate included must have issued the preceding certificate. For each issuer, a CRL may be supplied. A CRL in the chain belongs to the immediately following issuer. Therefore, it potentially contains the immediately preceding certificate.

The <crlchain> production contains one certificate revocation list chain. The CRLs in the chain begin with the requested CRL and continue with the CRLs of subsequent issuers. The issuer of each CRL is presumed to have issued a certificate for the issuer of the preceding CRL. For each CRL, the issuer's certificate may be supplied. A certificate in the chain must belong to the issuer of the immediately preceding CRL.

The relationship between a certificate and an immediately preceding CRL is the same in both <certchain> and <crlchain>. In a <certchain> the CRLs are optional. In a <crlchain> the certificates are optional.

6. Examples

Each example is included as a separate section for ease of reference.

6.1. Original Message Prepared for Protection

Except as explicitly indicated, the following message is used as the message to be protected.

```
To: Ned Freed <ned@innosoft.com>
Subject: Hi Ned!

How do you like the new MOSS?

Jim
```

6.2. Sign Text of Original Message

When the text of the original message is signed, it will look like this, where lines with an ampersand '&' are digitally signed (note the use of the public key identifier with the included email name identifier, on the lines marked with an asterisk '*'):

```
     To: Ned Freed <ned@innosoft.com>
     Subject: Hi Ned!
     MIME-Version: 1.0
     Content-Type: multipart/signed;
       protocol="application/moss-signature";
       micalg="rsa-md5"; boundary="Signed Boundary"

     --Signed Boundary
   & Content-Type: text/plain; charset="us-ascii"
   & Content-ID: <21436.785186814.2@tis.com>
   &
   & How do you like the new MOSS?
   &
   & Jim

     --Signed Boundary
     Content-Type: application/moss-signature
     Content-ID: <21436.785186814.1@tis.com>
     Content-Transfer-Encoding: quoted-printable

     Version: 5
   * Originator-ID: PK,MHkwCgYEVQgBAQICAwADawAwaAJhAMAHQ45ywA357G4f=
   * qQ61aoC1fO6BekJmG4475mJkwGIUxvDkwuxe/EFdPkXDGBxzdGrW1iuh5K8kl8=
   * KRGJ9wh1HU4TrghGdhn0Lw8gG67Dmb5cBhY9DGwq0CDnrpKZV3cQIDAQAB,EN,=
   * 2,galvin@tis.com
     MIC-Info: RSA-MD5,RSA,PnEvyFV3sSyTSiGh/HFgWUIFa22jbHoTrFIMVERf=
     MZXUKzFsHbmKtIowJlJR56OoImo+t7WjRfzpMH7MOKgPgzRnTwk0T5dOcP/lfb=
     sOVJjleV7vTe9yoNp2P8mi/hs7

     --Signed Boundary--
```

6.3. Sign Headers and Text of Original Message

If, instead, we choose to protect the headers with the text of the original message, it will look like this, where lines with an ampersand '&' are encrypted:

```
        To: Ned Freed <ned@innosoft.com>
        Subject: Hi Ned!
        MIME-Version: 1.0
        Content-Type: multipart/signed;
          protocol="application/moss-signature";
          micalg="rsa-md5"; boundary="Signed Boundary"

        --Signed Boundary
      & Content-Type: message/rfc822
      & Content-ID: <21468.785187044.2@tis.com>
      &
      & To:         Ned Freed <ned@innosoft.com>
      & Subject:    Hi Ned!
      &
      &
      & How do you like the new MOSS?
      &
      & Jim

        --Signed Boundary
        Content-Type: application/moss-signature
        Content-ID: <21468.785187044.1@tis.com>
        Content-Transfer-Encoding: quoted-printable

        Version: 5
        Originator-ID: PK,MHkwCgYEVQgBAQICAwADawAwaAJhAMAHQ45ywA357G4f=
        qQ61aoC1fO6BekJmG4475mJkwGIUxvDkwuxe/EFdPkXDGBxzdGrW1iuh5K8kl8=
        KRGJ9wh1HU4TrghGdhn0Lw8gG67Dmb5cBhY9DGwq0CDnrpKZV3cQIDAQAB,EN,=
        2,galvin@tis.com
        MIC-Info: RSA-MD5,RSA,ctbDBgkYtFW1sisb5w4/Y/p94LftgQ0IrEn3d6WT=
        wjfxFBvAceVWfawsZPLijVKZUYtbIqJmjKtzTJlagBawfA/KhUsvTZdR6Dj+4G=
        d8dBBwMKvqMKTHAUxGXYxwNdbK

        --Signed Boundary--
```

6.4. Encrypt Text of a Message

If we choose to encrypt the text of the following message, that is, encrypt the lines marked with asterisk '*':

```
        To: Jim Galvin <galvin@tis.com>
        Subject: an encrypted message

      * How do you like the new MOSS?
      *
      * Jim
```

the message would look as follows (note the use of the email name identifier, on the line marked with an asterisk '*'):

```
     To: Jim Galvin <galvin@tis.com>
     Subject: an encrypted message
     MIME-Version: 1.0
     Content-Type: multipart/encrypted;
       protocol="application/moss-keys";
       boundary="Encrypted Boundary"

     --Encrypted Boundary
     Content-Type: application/moss-keys
     Content-ID: <21535.785187667.1@tis.com>
     Content-Transfer-Encoding: quoted-printable

     Version: 5
     DEK-Info: DES-CBC,D488AAAE271C8159
   * Recipient-ID: EN,2,galvin@tis.com
     Key-Info: RSA,ISbC3IR01BrYq2rp493X+Dt7WrVq3V3/U/YXbxOTY5cmiy1/=
     7NvSqqXSK/WZq05lN99RDUQhdNxXI64ePAbFWQ6RGoiCrRs+Dc95oQh7EFEPoT=
     9P6jyzcV1NzZVwfp+u

     --Encrypted Boundary
     Content-Type: application/octet-stream
     Content-Transfer-Encoding: base64

     AfR1WSeyLhy5AtcX0ktUVlbFC1vvcoCjYWy/yYjVj48eqzUVvGTGMsV6MdlynU
     d4jcJgRnQIQvIxm2VRgH8W8MkAlul+RWGu7jnxjp0sNsU562+RZr0f4F3K3n4w
     onUUP265UvvMj23RSTguZ/nl/OxnFM6SzDgV39V/i/RofqI=

     --Encrypted Boundary--
```

6.5. Encrypt the Signed Text of a Message

If, instead, we choose to sign the text before we encrypt it, the structure would be as follows, where lines with an asterisk '*' are digitally signed and lines with an ampersand '&' are encrypted:

```
       Content-Type: multipart/encrypted;
         protocol="application/moss-keys";
         boundary="Encrypted Boundary"

       --Encrypted Boundary
       Content-Type: application/moss-keys

       KEY INFORMATION

       --Encrypted Boundary
       Content-Type: application/octet-stream

   &   Content-Type: multipart/signed;
   &     protocol="application/moss-signature";
   &     micalg="rsa-md5"; boundary="Signed Boundary"
   &
   &   --Signed Boundary
   & * Content-Type: text/plain
   & *
   & * How do you like the new MOSS?
   & *
   & * Jim
   &
   &   --Signed Boundary
   &   Content-Type: application/moss-signature
   &
   &   SIGNATURE INFORMATION
   &
   &   --Signed Boundary--

       --Encrypted Boundary--
```

where KEY INFORMATION and SIGNATURE INFORMATION are descriptive of the actual content that would appear in a real body part. The actual message would be like this:

```
To: Jim Galvin <galvin@tis.com>
Subject: an encrypted message
MIME-Version: 1.0
Content-Type: multipart/encrypted;
  protocol="application/moss-keys";
  boundary="Encrypted Boundary"

--Encrypted Boundary
Content-Type: application/moss-keys
Content-ID: <21546.785188458.1@tis.com>
Content-Transfer-Encoding: quoted-printable

Version: 5
DEK-Info: DES-CBC,11CC89F8D90F1DFE
Recipient-ID: EN,2,galvin@tis.com
Key-Info: RSA,AZTtlEc6xm0vjkvtVUITUh7sz+nOuOwP0tsym6CQozD9IwVIJz=
Y8+vIfbh5BpR0kS6prq3EGFBFR8gRMUvbgHtEKPD/4ICQ7b6ssZ7FmKhl/cJC5rV=
jpb4EOUlwOXwRZ

--Encrypted Boundary
Content-Type: application/octet-stream
Content-Transfer-Encoding: base64

ZvWvtosDzRBXJzkDFFRb9Qjrgm2nDWg3zotJ3ZpExpWUG/aRJ7Vwd+PWkSfrDPJ5
2V/wkxwMrum6xJHZonrtyd0AvaztvriMm2zXTefzwpGG1i5zK47PBqreLA3HDTK2
U6B13vzpE8wMSVefzaCTSpXRSCh08ceVEZrIYS53/CKZV2/Sga71pGNlux8MsJpY
Lwdj5Q3NKocg1LMngMo8yrMAe+avMjfOnhui49Xon1Gft+N5XDH/+wI9qxI9fkQv
NZVDlWIhCYEkxd5ke549tLkJjEqHQbgJW5C+K/uxdiD2dBt+nRCXcuO0Px3yKRyY
g/9BgTf36padSHuv48xBg5YaqaEWpEzLI0Qd31vAyP23rqiPhfBn6sjhQ2KrWhiF
2l3TV8kQsIGHHZUkaUbqkXJe6PEdWWhwsqCFPDdkpjzQRrTuJH6xleNUFg+CG1V+
tL4IgMjQqm3KVojRXx8bG2auVN89NfwFswmoq4fXTrh3xyVS1VgxjKkcYI8SVVmk
YjCxVviJP3zO2UzBvCoMfADtBVBz1njYETtVGDO97uT39MqL85uEgiF4E5TkOj/m
04+88G0/vvN/RISKJiFQJ3FyVIB/ShX9Dixl8WCx3rxwN5g2QFLiyQVulzuNhimS
D4ZxEo7smcTsAXUjwSLRtdjmTTutw2GmFESUaIrY81NcpQJRPNAvF0IkN6ddwL4q
vzUS99vjQp15g9FUv82lHtHwhM18a9GokVG8xYOjBBsn9anp9abh4Tp/c/vpbunQ
UqnpV29rF4wj+8OwUOMi9ymGabBXAjw7DhNH2RdRVr1upQO896OX81VWB0LsA0cp
+ymxhTrEI+wCHcrsNMoRK/7zAeuAi0f1t9bN594EFlLoIrBnKEa1/OUAhMT7kG1f
NkSRnc8BZswIoPyRetsTurQfD40nsVHvNwE9Jz7wbBo00gd6blPADOUYFxfW5zu6
ubygBqJiKPM4II2fCdNj7CptfQcoRTeguKMVPLVmFg/EINuWBFm10GqlYT7p4zhf
zysV/3r5LVZ1E8armTCRJ2GoYG5h+SKcytaQ0IT8S2nLPCZl1hzdajsrqHFe8omQ

--Encrypted Boundary--
```

6.6. Protecting Audio Content

In addition to text, the MOSS services as defined here will protect arbitrary body parts, for example, the following audio body part:

```
Content-Type: audio/basic

AUDIO DATA HERE
```

6.6.1. Sign Audio Content

When signed an audio content would appear as follows, where lines with an ampersand '&' are digitally signed:

```
     Content-Type: multipart/signed;
       protocol="application/moss-signature";
       micalg="rsa-md5"; boundary="Signed Boundary"

     --Signed Boundary
   & Content-Type: audio/basic
   & Content-Transfer-Encoding: base64
   &
   & base64(AUDIO-DATA-HERE)

     --Signed Boundary
     Content-Type: application/moss-signature

     SIGNATURE-INFORMATION-HERE

     --Signed Boundary--
```

where AUDIO-DATA-HERE and SIGNATURE-INFORMATION-HERE are descriptive of the content that would appear in a real body part.

6.6.2. Encrypt Audio Content

When encrypted an audio content would appear as follows, where lines with an ampersand '&' are encrypted:

```
      Content-Type: multipart/encrypted;
        protocol="application/moss-keys";
        boundary="Encrypted Boundary"

      --Encrypted Boundary
      Content-Type: application/moss-keys

      KEY-INFORMATION-HERE

      --Encrypted Boundary
      Content-Type: application/octet-stream
      Content-Transfer-Encoding: base64

    & Content-Type: audio/basic
    &
    & base64(encrypted(AUDIO-DATA-HERE))

      --Encrypted Boundary--
```

where KEY-INFORMATION-HERE and AUDIO-DATA-HERE are descriptive of the content that would appear in a real body part.

7. Observations

The use of MIME and the framework defined by [7] exhibits several properties:

(1) It allows arbitrary content types to be protected, not just the body of an RFC822 message.

(2) It allows a message to contain several body parts which may or may not be protected.

(3) It allows the components of a multipart or message content to be protected with different services.

The use of a MIME-capable user agent makes complex nesting of protected message body parts much easier. For example, the user can separately sign and encrypt a message. This allows complete separation of the confidentiality security service from the digital signature security service. That is, different key pairs could be used for the different services and could be protected separately.

This is useful for at least two reasons. First, some public key algorithms do not support both digital signatures and encryption; two key pairs would be required in this case. Second, an employee's company could be given access to the (private) decryption key but not the (private) signature key, thereby granting the company the ability to decrypt messages addressed to the employee in emergencies without also granting the company the ability to sign messages as the employee.

8. Comparison of MOSS and PEM Protocols

MOSS differs from PEM in the following ways.

(1) When using PEM, users are required to have certificates. When using MOSS, users need only have a public/private key pair.

(2) MOSS broadens the allowable name forms that users may use to identify their public keys, including arbitrary strings, email addresses, or distinguished names.

(3) PEM currently only supports text-based electronic mail messages and the message text is required to be represented by the ASCII character set with "<CR><LF>" line delimiters. These restrictions no longer apply.

(4) The PEM specification currently requires that encryption services be applied only to message bodies that have been signed. By providing for each of the services separately, they may be applied in any order according to the needs of the requesting application.

(5) MIME includes transfer encoding operations to ensure the unmodified transfer of body parts. Therefore, unlike PEM, MOSS does not need to include these functions.

(6) PEM specifies a Proc-Type: header field to identify the type of processing that was performed on the message. This functionality is subsumed by the MIME Content-Type: headers. The Proc-Type: header also includes a decimal number that is used to distinguish among incompatible encapsulated header field interpretations which may arise as changes are made to the PEM standard. This functionality is replaced by the Version: header

specified in this document.

(7) PEM specifies a Content-Domain: header, the purpose of which is to describe the type of the content which is represented within a PEM message's encapsulated text. This functionality is subsumed by the MIME Content-Type: headers.

(8) The PEM specifications include a document that defines new types of PEM messages, specified by unique values used in the Proc-Type: header, to be used to request certificate and certificate revocation list information. This functionality is subsumed by two new content types specified in this document: application/mosskey- request and application/mosskey-data.

(9) The header fields having to do with certificates (Originator-Certificate: and Issuer-Certificate:) and CRLs (CRL:) are relegated for use only in the application/mosskey-data and application/mosskey-request content types and are no longer allowed in the header portion of a PEM signed or encrypted message. This separates key management services from the digital signature and encryption services.

(10) The grammar specified here explicitly separates the header fields that may appear for the encryption and signature security services. It is the intent of this document to specify a precise expression of the allowed header fields; there is no intent to disallow the functionality of combinations of encryption and signature security found in [3].

(11) With the separation of the encryption and signature security services, there is no need for a MIC-Info: field in the headers associated with an encrypted message.

(12) In [3], when asymmetric key management is used, an Originator-ID field is required in order to identify the private key used to sign the MIC argument in the MIC-Info: field. Because no MIC-Info: field is associated with the encryption security service under asymmetric key management, there is no requirement in that case to include an Originator-ID field.

(13) The protocol specified here explicitly excludes symmetric key

management.

(14) This document requires all data that is to be digitally signed to be represented in 7bit form.

9. Security Considerations

This entire document is about security.

10. Acknowledgements

David H. Crocker suggested the use of a multipart structure for the MIME and PEM interaction, which has evolved into the MOSS protocol.

The MOSS protocol is a direct descendant of the PEM protocol. The authors gratefully acknowledge the editors of those specification, especially John Linn and Steve Kent. This work would not have been possible had it not been for all of the PEM developers, users, and interested persons who are always present on the PEM developers mailing list and at PEM working group meetings at IETF meetings, especially, Amanda Walker, Bob Juenemann, Steve Dusse, Jeff Thomson, and Rhys Weatherly.

11. References

[1] Crocker, D., "Standard for the Format of ARPA Internet Text Messages", STD 11, RFC 822, University of Delaware, August 1982.

[2] Borenstein, N., and N. Freed, "MIME (Multipurpose Internet Mail Extension) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies", RFC 1521, Bellcore and Innosoft, September 1993.

[3] Linn, J., "Privacy Enhancement for Internet Electronic Mail: Part I: Message Encryption and Authentication Procedures", RFC 1421, IAB IRTF PSRG, IETF PEM WG, February 1993.

[4] Kent, S., "Privacy Enhancement for Internet Electronic Mail: Part II: Certificate-Based Key Management", RFC 1422, BBN Communications, February 1993.

[5] Balenson, D., "Privacy Enhancement for Internet Electronic Mail: Part III: Algorithms, Modes, and Identifiers", RFC 1423, Trusted Information Systems, February 1993.

[6] Kaliski, B., "Privacy Enhancement for Internet Electronic Mail: Part IV: Key Certification and Related Services", RFC 1424, RSA Laboratories, February 1993.

[7] Galvin, J., Murphy, S., Crocker, S., and N. Freed, "Security Multiparts for MIME: Multipart/Signed and Multipart/Encrypted", RFC 1847, Trusted Information Systems and Innosoft, September 1995.

[8] The Directory -- Models. X.501, 1988. Developed in collaboration, and technically aligned, with ISO 9594-2.

[9] The Directory -- Authentication Framework. X.509, 1988. Developed in collaboration, and technically aligned, with ISO 9594-8.

12. Authors' Addresses

Steve Crocker
CyberCash, Inc.
2086 Hunters Crest Way
Vienna, VA 22181

Phone: +1 703 620 1222
Fax: +1 703 391 2651
EMail: crocker@cybercash.com

James M. Galvin
Trusted Information Systems
3060 Washington Road
Glenwood, MD 21738

Phone: +1 301 854 6889
Fax: +1 301 854 5363
EMail: galvin@tis.com

Sandra Murphy
Trusted Information Systems
3060 Washington Road
Glenwood, MD 21738

Phone: +1 301 854 6889
Fax: +1 301 854 5363
EMail: murphy@tis.com

Ned Freed
Innosoft International, Inc.
1050 East Garvey Avenue South
West Covina, CA 91790

Phone: +1 818 919 3600
Fax: +1 818 919 3614
EMail: ned@innosoft.com

Appendix A: Collected Grammar

The version of the grammar in this document is as follows:

```
<version>       ::= "Version:" "5" CRLF
```

The following grammar tokens are used throughout this specification:

```
<encbin>        ::= 1*<encbingrp>

<encbingrp>     ::= 4*4<encbinchar>

<encbinchar>    ::= <ALPHA> / <DIGIT> / "+" / "/" / "="

<hexchar>       ::= <DIGIT> / "A" / "B" / "C" / "D" / "E" / "F"
                    ; no lower case
```

The content of an application/moss-signature body part is as follows:

```
<mosssig>       ::= <version> ( 1*<origasymflds> )

<version>       ::= "Version:" "5" CRLF

<origasymflds>  ::= <origid> <micinfo>

<origid>        ::= "Originator-ID:" <id> CRLF

<micinfo>       ::= "MIC-Info:" <micalgid> "," <ikalgid> ","
                    <asymsignmic> CRLF
```

The content of an application/moss-keys body part is as follows:

```
<mosskeys>      ::= <version> <dekinfo> 1*<recipasymflds>

<version>       ::= "Version:" "5" CRLF

<dekinfo>       ::= "DEK-Info" ":" <dekalgid>
                    [ "," <dekparameters> ] CRLF

<recipasymflds> ::= <recipid> <asymkeyinfo>

<recipid>       ::= "Recipient-ID:" <id> CRLF

<asymkeyinfo>   ::= "Key-Info" ":" <ikalgid> "," <asymencdek> CRLF
```

Identifiers are defined as follows:

```
<id>            ::= <id-subset> / <id-publickey> / <id-issuer>

<id-subset>     ::= <id-email> / <id-string> / <id-dname>

<id-email>      ::= "EN"  "," <keysel> "," <emailstr> CRLF

<id-string>     ::= "STR" "," <keysel> "," <string> CRLF

<id-dname>      ::= "DN"  "," <keysel> "," <dnamestr> CRLF

<id-publickey>  ::= "PK"  "," <publickey> [ "," <id-subset> ] CRLF

<id-issuer>     ::= "IS"  "," <dnamestr>  "," <serial> CRLF

<keysel>        ::= 1*<hexchar>
                    ; hex dump of a non-null sequence of octets

<emailstr>      ::= <addr-spec> / <route-addr>
                    ; an electronic mail address as defined by
                    ; these two tokens from RFC822

<string>        ::= ; a non-null sequence of characters

<dnamestr>      ::= <encbin>
                    ; a printably encoded, ASN.1 encoded
                    ; distinguished name (as defined by the 'Name'
                    ; production specified in X.501 [8])

<publickey>     ::= <encbin>
                    ; a printably encoded, ASN.1 encoded public
                    ; key (as defined by the
                    ; 'SubjectPublicKeyInfo' production specified
                    ; in X.509 [9])

<serial>        ::= 1*<hexchar>
                    ; hex dump of a certificate serial number
```

The content of an application/mosskey-request body part is as follows:

```
<request>       ::= <version>
                    ( <subject> / <issuer> / <certification> )

<version>       ::= "Version:" "5" CRLF
```

```
<subject>       ::= "Subject:" <id> CRLF

<issuer>        ::= "Issuer:" <id> CRLF

<certification> ::= "Certification:" <encbin> CRLF
```

The content of an application/mosskey-data body part is as follows:

```
<mosskeydata>   ::= <version>
                    ( <publickeydata> / <certchain> / <crlchain> )

<version>       ::= "Version:" "5" CRLF

<publickeydata> ::= "Key:" "PK" "," <publickey> ","
                    <id-subset> CRLF

<certchain>     ::= <cert> *( [ <crl> ] <cert> )

<crlchain>      ::= 1*( <crl> [ <cert> ] )

<cert>          ::= "Certificate:" <encbin> CRLF

<crl>           ::= "CRL:" <encbin> CRLF
```

Appendix B: Imported Grammar

Options normally present in the grammar reprinted here which are illegal in MOSS are excluded in this reprinting, for the convenience of the reader.

The following productions are taken from [5]. The grammar presented in [5] remains the authoritative source for these productions; they are repeated here for the convenience of the reader.

```
<dekalgid>          ::= "DES-CBC"
<ikalgid>           ::= "RSA"
<micalgid>          ::= "RSA-MD2" / "RSA-MD5"

<dekparameters>     ::= <DESCBCparameters>
<DESCBCparameters>  ::= <IV>
<IV>                ::= <hexchar16>
<hexchar16>         ::= 16*16<hexchar>

<asymsignmic>       ::= <RSAsignmic>
<RSAsignmic>        ::= <encbin>

<asymencdek>        ::= <RSAencdek>
<RSAencdek>         ::= <encbin>
```

The following productions are taken from [1]. The grammar presented in [1] remains the authoritative source for these productions; they are repeated here for the convenience of the reader.

```
<route-addr>    ::= "<" [ <route> ] <addr-spec> ">"

<route>         ::=  1# ( "@" <domain> ) ":" ; path-relative


<addr-spec>     ::= <local-part> "@" <domain>; global address

<local-part>    ::= <word> *( "." <word> )   ; uninterpreted
                                             ; case-preserved

<domain>        ::= <sub-domain> *( "." <sub-domain> )

<sub-domain>    ::= <domain-ref> / <domain-literal>

<domain-ref>    ::= <atom>                   ; symbolic
                                             ; reference

<domain-literal>::= "[" *( <dtext> / <quoted-pair> ) "]"
```

```
<dtext>          ::= <any CHAR excluding "[", "]",
                     "\" & <CR>, & including
                     linear-white-space>
                                          ; => may be folded

<word>           ::= <atom> / <quoted-string>

<quoted-string>  ::= """ *( <qtext> / <quoted-pair> ) """

<qtext>          ::= (any <CHAR> excepting """, "\", and CR,
                      and including <linear-white-space>)

<quoted-pair>    ::= "\" <CHAR>               ; may quote any
                                              ; char

<linear-white-space> ::= 1*( [ CRLF ] <LWSP-char> )
                                              ; semantics = SPACE
                                              ; CRLF => folding

<LWSP-char>      ::= SPACE / HTAB             ; semantics = SPACE


<atom>           ::= 1*(any <CHAR>
                     except <specials>, SPACE and <CTL>s)

<CHAR>           ::= <any ASCII character>

<CTL>            ::= <any ASCII control character and DEL>

<specials>       ::= "(" / ")" / "<" / ">" / "@"
                     /  "," / ";" / ":" / "\" / <">
                     /  "." / "[" / "]"
                                              ; Must be in quoted-
                                              ; string, to use
                                              ;  within a word.

<ALPHA>          ::= <any ASCII alphabetic character>
                                              ; (101-132, 65.-90.)
                                              ; (141-172, 97.-122.)

<DIGIT>          ::= <any ASCII decimal digit>; (60-71, 48.-57.)
```