MIME Object Security Services:
Issues in a Multi-User Environment

James M. Galvin and Mark S. Feldman
Trusted Information Systems

MIME Object Security Services: Issues in a Multi-User Environment

James M. Galvin <galvin@tis.com>
Trusted Information Systems

Mark S. Feldman <feldman@tis.com>
Trusted Information Systems

1. Introduction

An Internet email message consists of two parts: the headers and the body. The format of the headers and how they should be interpreted is described in RFC822 [1]. The body is text under the user's control and is not changed during normal mail transport.

Privacy Enhanced Mail (PEM) [2,3,4,5] was the first Internet standard to address security in email messages. It adds structure to message bodies to provide digital signature and encryption to text-based email messages. It uses a certificate-based key management system, based on the X.509 [6] standard.

Multipurpose Internet Mail Extensions (MIME) [7] was developed to provide for multi-part textual and non-textual email message bodies. It defines a structure for the format of the body. However, until recently, MIME did not include support for security services.

A specification has been proposed that defines a framework for digitally signing and encrypting MIME objects: Security Multiparts for MIME [8]. The framework provides an embodiment of a MIME object and its digital signature or encryption key. It was designed to be useful by a variety of security protocols.

The MOSS protocol [9] is a derivative of PEM. Although a MIME message could carry a PEM object or a PEM message could carry a MIME object, a better solution is to combine the features of both and provide a single, uniform solution in which the protocols function in a complementary fashion. MOSS is just such a solution.

MOSS enhances the MIME protocol without changing its currently supported functions or features. It uses the security multiparts framework to provide digital signature and encryption protection to single- and multi-part textual and non-textual MIME objects. MOSS expects MIME objects as input and outputs MIME objects.

MOSS is PEM insofar as it supports all of the functionality and features included in the PEM protocol. However, MOSS does not enforce some of the functionality required to be enforced by PEM. It also provides more functionality than PEM, including some that PEM could never provide.

The popularity of the MIME protocol continues to increase. There are several publicly available implementations for various hardware/software platforms and a few off-the-shelf products. In addition, the need for secure electronic communications is paramount, and PEM is not meeting the needs of the user community. The security multiparts framework and MOSS are a natural evolution of the state-of-the-art.

However, it is not enough for vendors to implement MOSS. A MOSS implementation must address the protection of the public/private key pairs used by the protocol. The MOSS specification includes a description of the issues that must be addressed but does not provide specific solutions. Following a brief summary of the protocol, the next section discusses the protection of the public/private key pairs and proposes a software-based solution. This solution is applicable to other protocols.

2. MOSS Protocol

The basic operation of a single application of the MOSS protocol is to input a MIME object and to output a MIME object. The input object may be any valid MIME object, including an output object from a previous application of the MOSS protocol. The output object is an embodiment of the input object and the control information that specifies which security service was applied to the object. This embodiment may be conveyed to a recipient or archived for later use by its originator. An overview and some details about each of the security services is described below.

The MOSS protocol is independent of the cryptographic algorithm used in support of its security services. The current specification includes recommended choices to facilitate interoperability. If other choices are desired, the required features of the algorithms needed are specified.

2.1 Overview

The two possible embodiments output by the MOSS protocol are defined by the security multiparts framework. When a digital signature is applied to a MIME object the multipart/signed content type is used. When encryption is applied the multipart/encrypted content type is used. Each of these content types is comprised of two nested objects: one for the MOSS control information and one for the protected MIME object. The content of each of these is described in succeeding sections.

The MOSS protocol assumes that there exists a public/private key pair for the originator applying the digital signature. When encryption is applied, the MOSS protocol assumes that there exists a public key for each recipient. Specifically, although the MOSS specification acknowledges the importance of validating the public keys used and evaluating the degree to which private keys are protected from disclosure, the issues are considered independent topics and not addressed.

MOSS differs from PEM in this respect since PEM requires that public keys be embodied in certificates that are managed by the Internet Certification Hierarchy defined in RFC1422. MOSS, in addition to supporting certificates, has the advantage of supporting bare public/private key pairs. This makes the protocol immediately usable by individuals and small communities of cooperating users.

Since the MOSS protocol uses the security multiparts framework, it can and does take advantage of the recursive characteristic of MIME. A MIME object can have either a digital signature or encryption applied to it or, if it has already had one applied to it, can have another applied by simply using the output object of the previous application as the input object to the next application of the MOSS protocol. In the case of encryption, this allows it to be used by itself, in partial support of encrypted anonymous email.[1] It also allows different protection to be applied to different parts of the email message, in arbitrarily complex ways.

Here again, MOSS differs from PEM, since PEM requires that encrypted objects be signed. Also, PEM does not currently specify how to recursively apply its services. Although the PEM specifications could be enhanced to include this, MOSS inherits the characteristic from MIME.

[1] The issue is the headers of the email message typically provide at least a hint of the origin of the message. So, while the source of the encrypted MIME object can be anonymous, the source of the message in which it appears may not be.

2.2 Digital Signature

The MOSS protocol supports the application of a digital signature by hashing the MIME object to be digitally signed and encrypting the hash value with a private key of an originator. A set of header/value pairs is created, formatted according to RFC822, and, where the header names match header names used by PEM, the values are set exactly as they are for PEM. In addition to other management information, the signature (encrypted hash value) is included in the set of headers.

Prior to hashing the MIME object to be signed, the object must be represented in 7bit MIME canonical format. This guarantees both that a forwardable authentication service is always available and that the object is uniquely and unambiguously represented on all hardware/software platforms.

The MOSS headers created are inserted in the control information body part of the enclosing multipart/signed content; the signed MIME object is inserted in the other body part and labeled according to its actual type.

The digital signature is verified by the recipient as follows.

1. The received signed object is canonicalized.
2. The hash value of the canonical object is re-computed.
3. The encrypted hash value found in the control information is decrypted with the originator's public key.
4. The re-computed value is compared to the decrypted value. If the values are equal and the correct public key is used, the signature is valid.

Determining whether the correct public key has been used is essential to the validation of the digital signature. A complete discussion of the issues is beyond the scope of this paper but may be found in [10].

2.3 Encryption

The MOSS protocol supports encryption by generating a new encryption key for each MIME object to be encrypted and encrypting the object with it. The encryption key is then encrypted with the public key of the recipient(s) for whom the object is intended, including the originator if that is desired. A set of header/value pairs is created, formatted according to

RFC822, and, where the header names match header names used by PEM, the values are set exactly as they are for PEM. In addition to other management information, the encrypted encryption key is included in the set of headers.

The MOSS headers created are inserted in the control information body part of the enclosing multipart/encrypted content; the encrypted MIME object is inserted in the other body part and labeled application/octet-stream.

A recipient decrypts an encrypted object by obtaining the encrypted encryption key from the control information, decrypting it with the recipient's private key, decrypting the MIME object with the decrypted encryption key, and processing the output of the decryption as a MIME object.

3. Non-Protocol Issues

There are three issues not specifically addressed by the MOSS protocol specification that are important to any implementation.[2] These issues relate to the security of local information. In a single-user environment, a user controls the physical access to the computer. As a result, the user may not be concerned about what others may do to or with the information on the computer. However, a user should still employ additional protection in case physical security is lost.

However, in a multi-user environment, users must be concerned about both the integrity and the privacy of their information beyond physical security. In order to apply the MOSS protocol, an implementation must be able to access two kinds of information: private keys and public keys. The private keys are needed to digitally sign MIME objects and to decrypt the encryption keys of encrypted MIME objects. The public keys are needed to verify digital signatures and to encrypt the encryption keys of encrypted MIME objects.

With respect to private keys, the principal issue for a user is preventing the disclosure of the private key to others. If an adversary learns the user's private key, they would be able to sign MIME objects and make it appear as if the user had signed the object. In addition, the adversary would be able to read MIME objects encrypted for the user. When a user's private key is disclosed, no security is available to the user with that public/private key pair.

[2] One issue explicitly not addressed at this time is the vulnerability of the software to Trojan horses. This paper assumes root is not hostile.

With respect to public keys, a user labels the public keys owned by other users with their identity. The principal issue is preventing modification of the binding between the public key and the identity of its owner. If an adversary could modify the binding, a recipient would incorrectly believe the indicated origin of a signed object or an originator would unintentionally encrypt a message for the wrong recipient, probably the adversary. When a binding is compromised, no security is available between the local user and the owner indicated by the previously valid binding.

Finally, a MOSS implementation must have **access** to a source of randomness, more precisely a source of unpredictable bits. The unpredictable bits are required when generating public/private key pairs for users and when generating encryption keys for sending encrypted messages. As many as several thousand bits may be required each time they are needed. Without hardware support, the bits can be difficult to obtain [11].

There are many solutions for these issues, including hardware, software, and hybrid approaches. In this paper we consider only software solutions for several reasons. First, software solutions are applicable to a broader range of environments. Any changes required for a new environment can usually be completed quickly and easily. Second, software solutions are quick and easy to install, since they do not require any specialized hardware or other computing resources. Third, software solutions are typically less expensive than alternate per-host or per-user hardware solutions.

3.1 Protection of Private Keys

If private keys are kept on-line, they are vulnerable to access by unauthorized users. File permissions alone are not adequate for protecting private keys on most systems, though they are part of an overall solution. Private keys protected only by file permissions are vulnerable to account intruders and the accidental mis-setting of the file permissions.

One solution is store the private key in a file on a removable media, e.g., a diskette. Since diskette drives are typically included with most hardware configurations, the user is not burdened with a significant additional cost. However, if the diskette is lost or otherwise not physically protected, the private key is still vulnerable.

Encryption is an accepted solution for protecting information from disclosure. However, encryption also requires access to a key that must be protected from disclosure.

A solution to this problem is to derive the encryption key needed to encrypt the private key from easily available information. This process is straightforward for secret key algorithms, while no similar solution has been proposed for public key algorithms.

The recommended advice is to make the easily available information a passphrase selected by the user. A passphrase is different from a password in that no restrictions are placed on its length or value. This accomplishes two important features. First, the domain from which the passphrase is chosen is limited only by the input device used by the user. Second, the user can select an easily remembered value, e.g., a favorite quotation or other concatenation of many easily remembered words. Whenever the private key is needed, the user enters the passphrase, the encryption key is derived, the private key is decrypted, and then the private key is available for use.

The combination of file permissions and encryption provides effective non-disclosure protection to a user's private key, if the user chooses an appropriate passphrase. Better protection is provided if the file containing the encrypted private key is stored on a removable media, e.g., a diskette.

3.2 Protection of Public Keys

There are two issues relevant to the protection of public keys. The first is establishing the identity of the owner of the public key and the second is protecting the binding of the identity and the public key.

With respect to the first (establishing the identity), an issue that is beyond the scope of this paper is exactly what constitutes an identity. A discussion of this issue can be found in [10]. In addition, an originator needs a mechanism with which an identity and a public key may be conveyed to recipients for storage in the recipients' local databases. The MOSS protocol includes a specification for just such a mechanism, that is not described here.

With respect to the second (protecting the binding), the recommended solution is to create a cryptographic binding between the public key and the identity. Since the MOSS protocol works with bare public/private key pairs, the most basic solution is for each user to digitally sign entries in their own local databases. This reduces the problem of protecting public keys to the problem of protecting private keys.

Preventing the modification of the binding between public keys and identities is essential to the correct operation of the MOSS protocol. As indicated in the MOSS specification, a user must determine that all public keys received actually belong to the user to whom they purport to belong. The first time this validation process is completed is expensive, since it may involve an out-of-band communication with the owner of the public key. Thus, upon completion of this process, the user may choose to store the results with the public key in the database. However, if the database is kept on-line, it is vulnerable to modification by other users.

Digitally signing entries in a local database allows a user to keep the public keys on-line and to be able to detect if a binding has been modified. Whenever the public key is needed the signature would be verified first, thus guaranteeing that the binding of the public key and its name has not been modified since it was stored.

It should be noted that the above protection is useful even if the public keys are distributed in certificates. For those correspondents using certificates that are part of a hierarchy and not just bare public keys, the user could validate the certificate the first time it is received and store the result with the certificate, digitally signing all the information in the entry in the local database. This would enhance the performance of an implementation since it may not be necessary to complete the entire certificate validation each time it is accessed.

3.3 Unpredictable Bits

The ready availability of a sequence of unpredictable bits, which are distinct from random bits, is absolutely essential to the generation of public/private key pairs and encryption keys. Randomness is a statistical property of a sequence of values. The requirement is for an adversary to be unable to predict the next bit in a sequence even when all previous bits are known. Pseudo-random number generators rarely include this absolutely essential property.

The problem is if it is possible to predict some of the sequence of bits used, it may be possible to reduce the size of the domain from which the key being generated is selected. If the domain is significantly reduced, an exhaustive search of the domain for the key may be possible.

Locating a source of unpredictable bits presents a unique problem on multi-user systems (and most single-user systems for that matter). Typically, a hardware source of unpredictable bits is not included with most system configurations. Although most computer systems include applications that will display the always changing status of the local system, the unpredictability of these bits is limited, especially on a lightly loaded system on which an adversary has access to the local system around the same time that the keys are being generated.

The most effective software-based solution currently available is to hash with a cryptographically strong one-way hash function the largest quantity of information with limited unpredictability available. Since a hash typically generates a fixed size quantity, the process is iterated as many times as are necessary to get the required number of unpredictable bits.

4. Implementation

Trusted Information Systems (TIS) has been developing an openly available implementation of the MOSS protocol (TIS/MOSS) [12]. It is software-based and includes all of the solutions proposed here. In particular, the private keys are each stored in their own binary file while all other information, including the public keys, is stored in a flat ASCII file. A sequence of unpredictable bits is obtained by hashing system status information.

4.1 Private Keys

Each of a user's private keys is stored in its own file with permissions that are initially created to allow only the user to read and write the file. The location of the private key files is up to the user, with the default being the user's home directory. Private key files can be placed on removable media by simply including the device specification in the filename.

A user may optionally encrypt the private key in the file. If it is encrypted, the user must input the passphrase used to derive the encryption key every time the private key is accessed; the software retains the decrypted private key for as short a time period as possible.

From a security perspective, this is the optimal behavior. However, users quickly become irritated if they send or receive more than a few messages in a short time frame. As a result, TIS/MOSS includes a feature that allows users to choose usability over security. A pair of programs, mosslogin and mosslogout, is included with which users may enter their passphrase once for a timeframe they choose.

Each file is formatted as a sequence of octets, with the first octet excluded from any encryption that may be used to protect the private key. The first octet is used for bit flags that indicate if the private key is encrypted and the algorithm with which it is encrypted.

The next 16 octets are a hash of the private key with the remaining octets comprising the ASN.1 encoding of the private key. Encrypting the private key entails encrypting the octets of the hash and the encoded private key.

The hash in the file is used as an integrity check of the private key's value. This is most useful when the private key has been encrypted. Checking the hash value after decrypting the file contents provides a fast mechanism for determining if the correct passphrase was provided from which to derive the encryption key. Without the hash value, the only mechanism by which the private key's value can be checked would be to use it and see if it works. This typically causes an incorrect failure condition to be reported to the user. TIS/MOSS uses the MD5 [13] hash algorithm for this check.

4.2 Public Keys

Except for the private keys themselves, all information needed by MOSS is stored in a flat ASCII file, called the MOSS database. The permissions on the database file should be such that only the user owning the file can make changes. The database may be left readable by other users if it is desirable to share it. The user may have multiple databases and may choose the location of each database. The default is a single database in the file ".mossdb" in the user's home directory.

The database is organized as a set of user records separated by a blank line. Each user record is organized as a set of tag/value pairs, which conform to the RFC822 header syntax. This format is extensible and easily processed using many tools.

The tags described in this paper are those necessary for implementing local security. TIS/MOSS makes use of many other tags and silently ignores all unrecognized tags and their values.

Within a user record, all tags except the "public-key:" tag are optional but, if present and needed, their values will be used. All tags except the "alias:" tag must be unique. If multiple tags are present, all except the first are silently ignored.

The "public-key:" tag is required to be present in each user record and its value must be unique with respect to all other public key values in the database. User records without this tag are silently ignored. Its value is the base64 encoded, ASN.1 encoded public key.

The “alias:” tag is a grouping mechanism. Its value is a string. It provides a convenient way of *addressing* multiple recipients, for example when sending an encrypted message. Aliases can be unique to a single user record in the database or the same alias may appear in several user records. A single user record may contain multiple “alias:” tags, with some aliases that are unique to it and others that are not.

A local name may be bound to a public key by selecting a string and placing it in an “alias:” tag in the user record. Its value is arbitrary from the point of view of TIS/MOSS; the value is always used exactly as it appears in the database. TIS/MOSS also uses the first “alias:” tag in a user record for display purposes, providing a convenient way for a user to specify a local, short handle for a public key.

A “trusted:” tag is the mechanism with which a user creates a cryptographic embodiment for each user record in the user’s database, including the public key and the local identity specified in the “alias:” tag. The user record is canonicalized and a digital signature is created. The digital signature and the alias of the user that created it are stored as the value of a “trusted:” tag in the user record. A user may create their embodiment by adding their own digital signature to user records or the user may use the digital signature created by another user.

Whenever a “trusted:” tag is found in a user record, the digital signature is validated before the information in the record is used. This protects all information in the user record from modification. TIS/MOSS always indicates to a user whether the public keys used came from trusted or untrusted user records.

4.3 Unpredictable Bits

Unpredictable bits are required for key generation and for padding certain cryptographic values. The largest amount of unpredictable bits is required when public/private key pairs are generated. Since public/private key pair generation is relatively infrequent and all MOSS security is based on the non-disclosure of the private key, the cost associated with generating the unpredictable bits may be allowed to be quite large. When generating unpredictable bits on a per-message basis for encryption keys and padding, the same high-cost method of generating unpredictable bits used when generating public/private key pairs proves to be too expensive.

The required number of unpredictable bits in TIS/MOSS is produced by a cryptographically-strong pseudo-random number generator (PRNG) based on the MD5 hash of a seed. The seed is incremented and hashed iteratively to produce the necessary amount of unpredictable bits. The seed is initialized in one of two ways depending on how the unpredictable bits will be used.

If the unpredictable bits are being used to generate a public/private key pair, the seed is based on the hashing of the output of a large number of system commands. The system commands included were chosen because their output has the greatest probability of being different across multiple executions and hard to guess. This can be time consuming, but it is an infrequent operation and public/private key pair generation is already slow, so the additional time is inconsequential.

It is essential that only commands that are likely to produce unpredictable output be included. On multi-user systems, no special privileges are needed for another user to execute the same set of commands at approximately the same time and possibly aid an exhaustive search of the key space. This is especially important on a lightly loaded system, since otherwise the output is almost certain to be unpredictable for most if not all of the commands.

When unpredictable bits are required for purposes other than generating public/private key pairs, a faster method of seeding the PRNG is used that makes use of the unpredictability inherent in the private key. When unpredictable bits are required for per-message keys or padding, a value is signed using an existing private key to produce the seed. The value being signed need not be unpredictable, but it must be unique. Signing a unique value produces a value that is unpredictable to anyone without the private key used to generate the signature.

The uniqueness of the value to be signed is guaranteed by concatenating values that are specific to a certain time, a certain machine, a certain user, and a certain process. The concatenated values are obtained from system calls. The combination of the system calls and the signature are guaranteed to produce a seed with at least as much unpredictability as that in the private key and in much less time than the method used to generate unpredictable bits for public/private key pair generation.

5. Conclusions

Software-based solutions have been proposed for creating and protecting the signature and encryption keys used in MOSS. The proposals have been implemented, tested, and found to provide effective protection against various disclosure and modification threats.

The software-based solution proposed for obtaining a sequence of unpredictable bits has served the TIS/MOSS implementation (and prior to this the TIS/PEM implementation) for more than two years. Prior to that several alternatives methods were tried that failed. Ideally, the best solution is for hardware manufacturers to include a source of unpredictable bits in all configurations. The importance of unpredictable values must not be underestimated.

The MOSS protocol, in conjunction with MIME, provides a flexible, extensible means by which the addition of security services can be studied. While software-based solutions may be sufficient for many applications today, future work must include the study and implementation of hardware- and hybrid-based solutions. TIS has begun experimenting with hardware-based cryptography and private key storage. TIS/MOSS has been modified to integrate an experimental PCMCIA-based cryptographic engine.

In addition, future work must include the study of the use of trusted systems for secure email applications. Trusted systems can protect private keys from disclosure without the use of encryption. They also provide protection from Trojan horses and hostile roots.

6. References

[1] David H. Crocker. Standard for the Format of ARPA Internet Text Messages. RFC822, University of Delaware, August 1982.

[2] John Linn. Privacy Enhancement for Internet Electronic Mail: Part I: Message Encryption and Authentication Procedures. RFC1421, February 1993. Obsoletes RFC1113.

[3] Steve Kent. Privacy Enhancement for Internet Electronic Mail: Part II: Certificate-Based Key Management. RFC1422, BBN Communications, February 1993. Obsoletes RFC1114.

[4] David M. Balenson. Privacy Enhancement for Internet Electronic Mail: Part III: Algorithms, Modes, and Identifiers. RFC1423, Trusted Information Systems, February 1993. Obsoletes RFC1115.

[5] Burton S. Kaliski. Privacy Enhancement for Internet Electronic Mail: Part IV: Key Certification and Related Services. RFC1424, RSA Laboratories, February 1993.

[6] The Directory—Authentication Framework: X.509, 1993. Developed in collaboration and technically aligned with ISO 9594-8.

[7] Nathaniel Borenstein and Ned Freed. MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies. RFC1521, Bellcore and Innosoft, September 1993. Obsoletes RFC1341.

[8] James Galvin, Sandy Murphy, Steve Crocker, and Ned Freed. Security Multiparts for MIME: Multipart/Signed and Multipart/Encrypted. Trusted Information Systems and Innosoft. Work in progress.

[9] Jim Galvin, Sandy Murphy, Steve Crocker, and Ned Freed. MIME Object Security Services. Trusted Information Systems and Innosoft. Work in progress.

[10] James Galvin and Sandra Murphy. Using Public Key Cryptography: Issues of Binding and Protection. To be published, INET'95.

[11] Donald Eastlake, Steve Crocker, Jeff Schiller. Randomness Recommendations for Security. RFC1750, DEC, Trusted Information Systems, and MIT, December 1994.

[12] Available via anonymous FTP from the host ftp.tis.com. Users should retrieve the file /pub/MOSS/README for details.

[13] Ron Rivest. The MD5 Message Digest Algorithm. RFC1321, April 1992.