RIPEM User's Guide

for RIPEM version 1.2

Mark Riordan

mrr@scss3.cl.msu.edu

June 1993

revised January 1994
for RIPEM 1.2 by Jeff Thompson
jefft@netcom.com

Introduction

What is RIPEM?

RIPEM (Riordan's Internet Privacy-Enhanced Mail, pronounced RYE-pehm) is a public key encryption program oriented toward use with electronic mail. It allows you to generate your own public keypairs, and to encrypt and decrypt messages based on your key and the keys of your correspondents. RIPEM is free, but each user is required to agree to a license agreement which places some limitations on its use.

This document is meant to instruct new users on the basic use of RIPEM. It does not replace the Unix `man` page `ripem.man`, also distributed with RIPEM. The `man` page describes all RIPEM options in reference format; some obscure options are discussed only in the `man` page. See the file CHANGE12.TXT in the RIPEM distribution for a discussion of the changes between RIPEM 1.1 and RIPEM 1.2. See also the Usenet newsgroup `alt.security.ripem`.

Public Key Encryption

Public key encryption, a fairly recent concept, is an encryption scheme in which messages are encrypted and decrypted with pairs of keys. One component of a user's keypair is used for encryption; the other is used for decryption. Thus, public key cryptography is sometimes referred to as *asymmetric* cryptography. Though both halves of the keypair are computed at the same time, neither can be derived from the other. (However, for convenience, a copy of the public key information is usually kept with the private key.)

This arrangement allows each correspondent to publish one half of his keypair (the encryption key, public key, or *public component*), keeping secret only the decryption half, or *private component*. Users wishing to send a message to, say, Alice, simply consult a non-secret directory of public components to find Alice's public key component. They encrypt their messages to Alice using her public key. Because only Alice knows her private component, only she can decrypt any of these messages to her. And none of the users corresponding with Alice need ever have first exchanged any secret information with her.

Each user needs keep secret only his/her own private component. Contrast this with traditional secret-key, or symmetric, cryptography. In a group of N correspondents, each user must keep track of N-1 secret keys. Furthermore, the **total** number of secret keys required for traditional cryptography is (N)*(N-1)/2, much larger than the N keys required by public key cryptography.

Thus, public key cryptography's value lies in improved key management, especially for large numbers of correspondents. However, for the value of public key cryptography to be realized, there must be an effective way for individual users to widely advertise their public key components.

Privacy-Enhanced Mail

RIPEM provides capabilities very similar to Privacy-Enhanced Mail (PEM), as described by Internet RFC's 1421-1424. RIPEM uses the concept of a *certificate*, a document which guarantees that you have the correct public key of a correspondent. However, RIPEM does not implement the full certificate *hierarchy* as described by RFC 1422. Instead, RIPEM supports a “direct trust” model where users certify each other directly without subscribing to a hierarchy. Because RIPEM does not implement a certificate hierarchy, it is not compliant with these Internet PEM RFC's. However, RIPEM is as compliant as is possible without implementing a certificate hierarchy.

As specified in the PEM RFC's, RIPEM generates a pseudo-random message key, and uses this key to encipher the message using a traditional *symmetric-key* encryption algorithm. In the current implementation of RIPEM, the DES (Data Encryption Standard) algorithm in one of two different modes. RIPEM then enciphers the message key using the RSA (Rivest-Shamir-Adleman) public key algorithm, and includes the enciphered message key with the message. Although the actual message text is never enciphered with a public key algorithm, the effect is the same. The advantage of this hybrid approach is performance-related: DES and other typical symmetric cryptosystems are typically thousands of times faster than public key systems.

RIPEM also "signs" the message by computing a checksum or hash function of the message plaintext, and encrypting this hash value with the sender's private key component. (Private RSA key components are usually used for decryption of messages encrypted with the public component, but in fact the reverse process also works.) Rivest's MD5 message digest algorithm is used for the hash function. This signature is verified by the recipient, to ensure that the message really was from the purported sender. The recipient computes her own message digest of the message after decrypting the message. The recipient then decrypts the encrypted message digest using the sender's public key and checks it against the recomputed digest. If the two match, the message must have been encrypted by the sender, since only the sender knows his private component.

The results of these computations--the encrypted message key, the encrypted message, the signature (encrypted hash value), and various pieces of control information--are formatted into lines of ASCII text suitable for inclusion into an electronic mail message.

About RIPEM

Platforms Supported

RIPEM runs on MS-DOS, Macintosh, OS/2, Windows NT, and a variety of Unix systems, including NeXTStep, SunOS, Sun Solaris 2.1, DEC ULTRIX, IBM AIX, HP/UX, SGI Irix, MIPS RISC/os, Motorola System V/88, Apollo, SCO Unix, Jolitz's 386BSD, Linux, ESIX, and others. Ports to other platforms are anticipated. Some ports of RIPEM do not have all the functionality of the Unix version; in particular, some versions do not implement direct network access to the RIPEM key server.

Licensing

The source code to RIPEM itself is in the public domain. However, because RIPEM was developed using RSA Data Security's RSAREF toolkit, use of RIPEM requires an RSAREF license. A copy of this license is included in RIPEM distributions, and users of RIPEM should read this license before running the program. The author of RIPEM believes that the current RSAREF license allows free personal use of RIPEM by citizens of the United States and Canada. Commercial use is forbidden. However, this personal interpretation has no legal standing, and RIPEM users are urged to read the RSAREF license agreement themselves. Note: persons wishing to redistribute RIPEM should consider relevant US government export restrictions.

How to Obtain RIPEM

RIPEM is distributed via anonymous FTP from `rsa.com`. RIPEM's home base, on which the most recent version can always be found, is the site `ripem.msu.edu`. RIPEM is distributed via non-anonymous FTP from this site. To comply with export restrictions, cryptology-related files on this server cannot be obtained via anonymous FTP. To apply for FTP access to `ripem.msu.edu`, send an email message to `ripem@ripem.msu.edu`. State your citizenship (must be USA or Canadian) and your willingness to comply with relevant export laws and software licenses. Also state the "canonical" Internet domain name of your host, and the country in which your host resides.

If you are not absolutely certain of the primary name of your host, FTP to `ripem.msu.edu` under user `anonymous`. The FTP server will inform you of your hostname. *This is extremely important*--experience distributing RIPEM to date has shown that many users do not know the canonical Internet hostname of their computer.

Here's a sample email message you might send:

```
To: ripem@ripem.msu.edu
Subject: FTP Access to ripem.msu.edu

Please give me access to ripem.msu.edu.  I am an American citizen,
and I agree to comply with crypto export laws and RSAREF license
terms.  My hostname is hobbit.egr.bigu.edu.  This host is located
in the United States.
```

After you have sent your request, you'll receive a special FTP username and password by return email. (There may be some delay, because I need to actually read your message before creating a username and password.) This username will work only from the hostname you specified in your message.

Once you have retrieved RIPEM, you are free to redistribute it, subject to export restrictions and RSAREF license terms. The complex distribution mechanism described above applies only to the site `ripem.msu.edu`, due to local site restrictions.

Caveats

Text files only. RIPEM encrypts only text-based messages; "binary" messages must be printably encoded (for instance, with *uuencode*) before being encrypted.

1023-character lines. The lines of text in plaintext messages processed by RIPEM must be less than 1024 characters long. (This restriction is borrowed from Internet RFC's on electronic mail and privacy-enhanced mail.)

Message size limits. Due to the nature of the RSAREF toolkit, RIPEM can encipher only messages which can fit entirely into the central memory of your computer. This is unlikely to be a problem on most workstations and larger computers, but may be a problem for some PC users. The vanilla MS-DOS version of RIPEM restricts messages to less than 48,000 characters.

Simple "filter" only. RIPEM acts only as a "filter": it simply reads an input source and produces output. RIPEM is not capable of formatting or delivering electronic mail messages. In fact, although RIPEM has some features to facilitate its use with electronic mail, it need not be used in conjunction with electronic mail at all. For use with electronic mail, RIPEM requires an external mail program; for instance, the Unix `mail` program.

No guarantees. As RIPEM is free software, it should not be surprising that it comes with no guarantees of any type.

Credits

RIPEM was written primarily by Mark Riordan, but nearly all of the cryptographic technology comes from the RSAREF toolkit by RSA Data Security, Inc. Much-appreciated contributions were made by Mark Henderson, Richard Outerbridge, Greg Onufer, Marc VanHeyningen, Mark Windsor, and others. The Macintosh version of RIPEM was written by Ray Lau.

Using RIPEM

Usage Overview

Using RIPEM generally requires the following steps: generating a keypair, communicating the public component of your key to correspondents, encrypting messages, and decrypting messages.

RIPEM has a bewildering array of command line options. However, most of them are not needed for ordinary use. Also, RIPEM looks at certain *environment variables* to determine what to do in the absence of certain command line options. Environment variables are named entities attached to your session which have values which you can set, either interactively or, more commonly, automatically at login time. For instance, a Unix user running the C Shell might include a line like

```
setenv RIPEM_SERVER_NAME ripem.msu.edu
```

in his/her `.cshrc` file, while an MS-DOS user would accomplish the same thing by including

```
set RIPEM_SERVER_NAME=ripem.msu.edu
```

in the `AUTOEXEC.BAT` file.

For discussion of individual environment variables, see the sections below and the RIPEM `man` pages. However, there is one environment variable of general interest: the variable `RIPEM_ARGS` can be given the value of options using exactly the same syntax as used in command line parameters. Conflicts between parameters specified both in `RIPEM_ARGS` and on the command line are resolved in favor of the command line.

Here is a quick, simplified run-through of sample RIPEM usage:

To generate a keypair, placing the public component and the private component in the directory named `/usr/smith/ripemhom`:

```
ripem -g -H /usr/smith/ripemhom -R eks
```

The directory `ripemhom` must already be created.

Assume at this point that you have collected a number of correspondents' public components by validating messages from them using -v.

To encrypt a message to `recipient@bighost.edu,` whose public key has already been validated and can be found in the directory `/usr/smith/ripemh`, the input message is in `mymessage`, and the encrypted output is to be placed in `cipher.out`:

```
ripem -e -r recipient@bighost.edu
 -i mymessage -o cipher.out -H /usr/smith/ripemh
```

To decrypt a message to you, reading from the file `cipher.out` and placing the decrypted message in the file `plain.out`, given the home directory `ripemhom`:

```
ripem -d -i cipher.out -o plain.out -H /usr/smith/ripemh
```

The RIPEM Home Directory

Each of the RIPEM operations uses a home directory. The files in the RIPEM home directory contain your private key, your self-signed certificate which identifies you, the certificates you create to validate other users, and other information. You specify the RIPEM home directory with the `-H` option. You may also set the RIPEM_HOME_DIR environment variable.

Generating a Keypair

Before you can use RIPEM, you must generate your own keypair. To do this, you must run RIPEM with the `-g` (for *generate*) option, and specify sources of pseudo-random information that RIPEM can use to create a unique keypair for you. RIPEM can obtain pseudo-random information from the running system, from characters you type at the keyboard, from a file, and from the command line. The first two options are generally the most useful.

You must also specify your RIPEM home directory.

Because keypairs are typically left unchanged for long periods of time--a year or more--it is very important that the private component of your keypair be kept secret. For this reason, RIPEM stores private key components only in encrypted form. (The key is encrypted using DES in CBC mode, with a pseudo-random "salt" added to the key.) When generating a keypair, RIPEM asks you for a key to be used to encrypt the private key component. This secondary key will be needed whenever you use RIPEM subsequently. It is critical that this key-to-a-key be chosen carefully, and that you remember it. If you forget the key to your private key component, your public key is worthless and unusable. The key to your private key can be up to 255 characters long. (This length limitation is an arbitrary implementation detail; RIPEM takes a hash function of the password you type before actually using it to encrypt the private component.)

A typical invocation of RIPEM to generate a keypair is:

```
ripem -g -u fred@snark.edu -R eks -H /usr/smith/ripemh
```

This call requests RIPEM to generate a keypair (`-g`). It identifies you (`-u`) as `fred@snark.edu`; this information is placed in the output files. RIPEM creates a self-signed certificate and adds it to the file `pubkeys` in your RIPEM home directory. The file `pubkeys` contains all the certificates which RIPEM creates using your private key. The self-signed certificate contains your generated public key and your name, and is signed with your generated private key. Note that generate also displays the digest of your self-signed certificate which you should record for later use. (See below for more on self-signed certificates.) The private (or secret) component is placed in the file `privkey` in the RIPEM home directory. RIPEM will prompt you (twice) for an encryption password before writing to this file. The `-R eks` option means that to obtain a pseudo-random data for key generation, RIPEM will use the **e**ntire command line, will prompt you at the **k**eyboard for a pseudo-random string, and will also query the **s**ystem for pseudo-random information before generating the keypair.

RIPEM identifies your key by your electronic mail address, which is specified by the `-u` option. If you omit the `-u` option, RIPEM will attempt to determine your email address by taking the value of the environment variable `RIPEM_USER_NAME` or, if that is not present, by querying the running system. It is best to identify your key in a form that others will be able to use as an email address. For instance, in the above example, `fred@snark.edu` is a better key identifier than just `fred`, because it is more readily used by correspondents on other hosts. If your host is known on the network by several different names, or if you ordinarily use several different computers interchangeably, it may be safer to explicitly specify your email address to RIPEM, rather than have it to figure out the address from the running system.

Note that not just an email address, but a full *distinguished name* is needed for the self-signed certificate. RIPEM uses the identifier as the common name in a distinguished name for the Persona certification authority. If your RIPEM identifier is fred@snark.edu, then your full distinguished name is “common name = fred@snark.edu, org unit = Persona Certificate, organization = RSA Data Security, Inc., country = US”.

By default, RIPEM generates keypairs roughly 516 bits in size. The author of RIPEM believes that this size is more than adequate for most purposes. However, the `-b` parameter is available for users who wish to generate larger keys. Specify `-b` *bitsize* to generate a key of size *bitsize* bits; *bitsize* must be between 512 and 1024, inclusive. Large keys are slower to generate as well as to subsequently use for encryption.

Generating a keypair is much slower than encryption or decryption. On a 386 PC-class computer, be prepared to wait several minutes for the key generation to complete.

Note that the first several bytes of all RIPEM keys are the same. This is due to RIPEM's use of OSI Distinguished Encoding Rules and associated key identifying strings to encode keys. It does not mean that the public keys generated are numerically similar.

Self-Signed Certificates

A self-signed certificate is like a normal certificate, except that it is issued by you for your own public key and name and signed with your own private key. This has two purposes: first, it is a convenient format for transmitting your name and public key; and second, when someone uses the public key in the certificate to verify the certificate's signature, it proves that you actually had possession of you private key when you made the signature. This prevents some crypto attacks. A self-signed certificate is included in every RIPEM message so that the recipient has the name and public key of the sender.

To establish trust for another person, you need a signed RIPEM message from that person. The first time you try to receive their message, RIPEM will tell you that you have not yet validated that user, and RIPEM will show you the *digest* on the sender's self-signed certificate. You may now call the sender on the phone and ask them to read the digest of their self-signed certificate. (Or you may get their digest through some other trustworthy channel.) If the digests match, then you have good reason to trust that the self-signed certificate is actually theirs. Now you can receive their signed message again in *validation mode* (see Receiving in Validation Mode below). RIPEM creates a certificate signed by you containing the other person's name and public key and adds it to `pubkeys` in your RIPEM home directory. Now you are ready to send that person encrypted mail and to trust further signed messages from them.

The reverse is also true: For another person to establish trust in your public key, you can send them a signed RIPEM message. Expect the other person to contact you and ask you to state the digest of your self-signed certificate. (If you forget your self-signed certificate digest, use RIPEM to change your key to private key. This will display your digest.) The other user then receives your signed message in validation mode, and now the two of you have a trusted channel.

Managing Keys According to RIPEM 1.1 and Earlier

Once you have generated a keypair, you must publicize the public component so that others can use it to send messages to you. Also, you must obtain access to the keys of other users. The best way for users to publish their public keys is by sending a RIPEM message which contains their self-signed certificate, as described in the previous section. However, some users may still be using RIPEM 1.1 or earlier which did not have self-signed certificates. This section describes the older model where public keys were distributed without being signed. (These distribution techniques will be useful in the future when RIPEM will support intermediate and hierarchical certificates.) If you don't need to interoperate with users of RIPEM 1.1 or earlier, skip to Managing Private Key Components.

In RIPEM 1.1 and earlier, key distribution can be by:

- Internet key server (requires Internet access for key lookup, but not for publication)
- The finger protocol (requires Internet access for key lookup and publication)
- Flat files (can be used with little or no network access)

You can choose the techniques that RIPEM uses to find a key by setting the `-Y` command line option. The `-Y` option takes an argument which is a string of one or more of the characters `s`, `g`, and `f`, which stand for **S**erver, fin**G**er, and **F**ile. For each correspondent, when necessary RIPEM will attempt to learn the correspondent's public key by consulting these sources in the order specified until the key is obtained.

The default value of the -Y option is "`sf`", which means that RIPEM first attempts to look up a public key via an Internet key **s**erver. If it is unsuccessful, it attempts to look up the key in a flat **f**ile. Read the discussion below for details on other related command line options.

Note: RIPEM 1.2 does not use key servers or finger to manage certificates. RIPEM 1.2 only transmits a self-signed certificate, and the only other certificates that are made are direct peer-to-peer. As a RIPEM 1.2 user, you make a certificate from yourself to, say, fred@snark.edu. No one other than you and fred would be interested in this certificate. Hence, RIPEM 1.2 makes no provision for these certificates to be on key servers. A future version of RIPEM is planned which will allow certificate chaining. This will allow you to indirectly trust users directly certified by users of your choice. You will be able to say "I trust all users certified by fred". When this future version of RIPEM is available, it will become meaningful to place certificates on key servers.

Key Distribution via the RIPEM Internet Key Server

Key Server Description and Limitations

If you have Internet access, you can communicate your key to others by registering the key on an Internet RIPEM key server. Currently, there is an "experimental" RIPEM key server running on the host `ripem.msu.edu`. This host is experimental in that it is an unofficial service which may have to be terminated with little or no advance notice.

This RIPEM key server acts as a central repository for public keys, saving users the effort of distributing their keys individually to all potential correspondents. This key server is not an especially secure mechanism. The level of security present in the key protocols is much less than that provided, for instance, by the Privacy Enhanced Mail certificate mechanism specified in the Internet PEM RFC's. The authenticity of keys maintained on the server is not guaranteed. The RIPEM key server is simply a means for RIPEM users to conveniently exchange keys.

Registering a Key via the Key Server

To allow the maximum number of users to publicize their keys via this mechanism, the RIPEM key server accepts key registration requests by electronic mail. Although the RIPEM key server itself is connected only to the Internet, users of non-Internet networks such as CompuServe, BITNET, and so on can register their keys by sending their key registration requests via an appropriate network gateway.

To register your key, send the public component (the output file from the -P option) to the email address

```
ripem-register-keys@ripem.msu.edu
```

On a Unix system, for instance, you can register your key by a command like:

```
mail ripem-register-keys@ripem.msu.edu <mypublickey
```

The key server will register your public key in its database and will send you a confirming message. The key is identified by the email address specified during the generation of the key, but the confirming message is sent to the address from which the key file was sent.

If you read electronic mail on several different hosts but wish to use the same public key on all of them, you can register the key under multiple names. You can do this by editing the key file before sending it to the server, and adding additional `User:` lines. (See the separate document on file formats.) Or, you can register the key under different names via separate email messages.

To subsequently delete your key from the server, encrypt a message starting with the string

```
RemoveKey
```

with the `-m mic-only` command line option and send the encrypted message to the address:

```
ripem-remove-keys@ripem.msu.edu
```

The message must have been encrypted by the owner of the key being removed.

To change your key on the server, generate a new keypair and encrypt the public component (the file from RIPEM's `-P` option) with the `-m mic-only` command line option. Send the result to the address:

```
ripem-change-keys@ripem.msu.edu
```

The message must have been encrypted by the owner of the key being changed.

Obtaining Keys from the Key Server: Live Access

Real-time "live" queries to the RIPEM key server are made directly by RIPEM using the UDP IP network protocol. "Live" queries are possible if your computer is connected to the Internet, your copy of RIPEM has been compiled for network access, and your computer is running the right network software. This is often true of Unix computers but is generally not true of other computers. At this writing, for instance, the MS-DOS version of RIPEM supports only the PC/TCP network software from FTP Software, Inc.

In order to access the key server, RIPEM needs to know its Internet address. You can tell RIPEM the address of the server in two ways: you can set the environment variable `RIPEM_SERVER_NAME` to the name of the server, or you can specify the server name with the `-y` command line option. In either case, you can specify more than one server name, separating the server names with commas (and no blank spaces). If you specify a list of server names in this way, when querying servers RIPEM will query the servers in the order listed until it obtains the desired public key, or exhausts the list.

Obtaining Keys from the Key Server: Email-Only Access

For users for whom live UDP network access to the RIPEM key server is not possible or not feasible, electronic mail access to the key server has been implemented. To obtain a copy of the complete database of registered RIPEM keys via email, send a message to the address:

```
ripem-get-keys@ripem.msu.edu
```

The subject and content of the message are ignored by the server, and hence can be left blank.

The return email message will contain a flat file of public keys, readable directly by RIPEM. This same file can be gotten by anonymous FTP to the host `ripem.msu.edu`. This file can be used as described below in the section discussing flat files.

Key Distribution via the Internet *Finger* Program

Another means of distributing keys over the Internet is via the *finger* mechanism. Finger is a simple protocol which allows a user to publish personal information that can be accessed across a TCP/IP network. For the most part, only Unix users can publish their keys using this mechanism. The advantage of finger over a RIPEM key server is that it relies only upon the correct operation of the sender's and receiver's computers, and upon the link between them. If the RIPEM key server is unavailable due to hardware, networking, or human errors, finger will be a more reliable choice. In general, though, key lookup using *finger* is slower than the RIPEM key server.

To set up your Unix account to give out your public key, include your public component in a file named `.plan`, located in your home directory. Your computer must be set up to run the *fingerd* finger daemon, which is normally the case on most Unix computers. Your computer must be up and running on the network for a correspondent to be able to access your key via the finger mechanism.

In no case do you need the finger program itself; RIPEM contains its own implementation of the finger protocol sufficient for the purpose of looking up keys. Hence, in some cases you can use the finger mechanism to look up someone else's key even if you are unable to publish your own key via finger due to the lack of a *fingerd* server program on your computer. For instance, an MS-DOS version of RIPEM is available that can look up keys via finger on PC's running the PC/TCP network implementation from FTP Software, Inc.

Aside from the `-Y g` command line option, there are no RIPEM command line options specific to the use of the finger mechanism.

Key Distribution via Flat Files

The key files generated by the `-g` option are ordinary text files formatted in such a way that a database of public keys can be created simply concatenating a collection of separate public key files. RIPEM is capable of scanning such flat files. It looks up a user's key based upon email address, so there is no ambiguity even if a flat file of keys contains keys for a large number of individuals from different sites.

The best way of obtaining a flat file containing keys is from the RIPEM key server, via email or FTP as described above.

When RIPEM uses a flat key for public key lookup, it determines the file's name in one of three ways:

- By using the value of the `-p` (for *public* key component) command line argument. If you specify `-p mypubfile` on the RIPEM command line, RIPEM will use `mypubfile` as the flat file of public keys.
- In the absence of `-p`, by using the value of the RIPEM_PUBLIC_KEY_FILE environment variable.
- If neither of the above is specified, by using a default file name. On Unix systems, this file name is the fixed name `/usr/local/etc/rpubkeys`. On MS-DOS systems, it is the name `\RIPEMPUB`.

It is possible to specify multiple public key files by specifying the `-p` option more than once. This allows you, for instance, to specify a common key file shared by a group of people, plus a separate key file for your personal correspondents.

Automatic Key Distribution via RIPEM Headers

One final way of distributing your public key component is automatic: If you are using version 1.1 or earlier and your public key is also in your private key file, RIPEM includes your public key in the PEM headers inside every message you encrypt. RIPEM takes advantage of this fact when deciphering, as follows:

When you are deciphering a message, RIPEM needs to find the sender's public component in order to verify the signature on the message. If RIPEM cannot find the public key of the sender via the mechanisms described above, or in a certificate, it resorts to looking in the PEM headers that immediately precede the ciphertext in any RIPEM-enciphered message. If RIPEM finds the sender's public component in the header, it will use this key in attempting to verify the signature. It will also issue a warning message, since if the message is fake, the key inside the message is probably fake as well.

If RIPEM does not find the sender's key, it exits with an error message.

Managing Private Key Components

Unlike public key components, private key components are stored in flat files only. Typically, a given user will have only one RIPEM key, and its private component will be kept by itself in the RIPEM home directory originally specified by the `-H` option during key generation.

During encryption and decryption, RIPEM will need to consult this file to determine your private key component. RIPEM determines the file's name in one of four ways:

- If the RIPEM home directory is specified either on the command line or by environment variable, use the file `privkey` in the home directory. Note that the `-s` option below may still be used but `privkey` will be searched first.
- By using the value of the `-s` (for *secret* key component) command line argument. If you specify `-s myprivatekey` on the RIPEM command line, RIPEM will use `myprivatekey` as the flat file of private keys.
- In the absence of `-s`, by using the value of the RIPEM_PRIVATE_KEY_FILE environment variable.
- If none of the above is specified, by using a default file name. On Unix systems, this file name is the fixed name `.ripemprv` in your home directory. On MS-DOS systems, it is the name `\RIPEMPRV`.

Because the private key component file generated by `ripem -g` identifies your key by your email address, it is possible to create a database of private keys by concatenating the generated private key files created by RIPEM. RIPEM uses only the private component that corresponds to your currently-specified email address. This may be useful if, for some reason, you wish to maintain multiple public keys, identified by different email aliases. Also, because the private key components are encrypted, it is possible to maintain a publicly accessible file of private key components without great loss of security. However, it is generally best for each user to have exactly one public key, and for its private component to be kept in its own file, reasonably secured against access by others.

Changing the Key to Your Private Key

You can change the key to your private component by using the `-c` option. This tells RIPEM to read your current private key file, decrypt it with your current key, prompt you for a new key, reencrypt your private component with the new key-to-the-key, and write out a new private component file. The old private key file is specified as described above. The modified private key file to be created may be specified via the `-S` option, otherwise the private key file in the home directory is rewritten “in place”. Thus, the sequence:

```
$ ripem -c -H ripemhom
Enter password to private key:
Enter new password to private key:
Enter again to verify:
```

reads the encrypted private key from `privkey` in `ripemhom` and changes it to the same private component encrypted with a new key.

Note that the `-c` option changes neither the private component nor the public component of your public key. If you believe that the key to your private component has been compromised, it is probably better to change your public key (by creating a new one) than to simply change the key to your private key. The `-c` option also displays the digest of your self-signed certificate. Therefore, if your need this digest, you can obtain it by changing your key to private key to itself. For more on self-signed certificates, see above.

If you are upgrading from RIPEM version 1.1 or earlier, the `-c` option will create a self-signed certificate for you. See Upgrading From Version 1.1 below.

Encrypting a Message

The `-e` option specifies that RIPEM is encrypting a message.

In encryption mode, RIPEM understands the `-H`, `-u`, `-p`, `-s`, `-y`, `-Y`, and `-R` options described above. The following options are also important:

Specifying Input and Output Files (`-i` and `-o`)

By default, RIPEM reads its input from *standard input*, and writes its output on *standard output*. If standard input and output are files, this is written as <*infile* and >*outfile* on most systems. Alternatively, an input file can be specified via the `-i` option, and an output file via the `-o` option:
`-i` *infile* `-o` *outfile*.

Specifying Recipients and Processing Mail Headers (`-r` and `-h`)

The recipient(s) of a message can be specified in two ways: on the command line, or via message headers in the input plaintext.

To specify recipients explicitly on the command line, use the -r option: `-r` *recipient_addr*. *Recipient_addr* must be the recipient's email address, in a form which RIPEM can use to look up the recipient's public key. For instance, suppose your recipient has valid email addresses `bob@egr.biguniv.edu` and `bob@biguniv.BITNET`. If Bob has registered his RIPEM public key only as `bob@egr.biguniv.edu`, then the address `bob@biguniv.BITNET` will not be adequate for RIPEM's purposes, even if it is a valid email address.

The `-r` option can be used multiple times for multiple recipients.

If the message plaintext has been prepared by a mail program, it may already contain mail headers which state the recipients' email addresses in "To:" and "cc:" lines. To take advantage of this situation, you can use the `-h` option. The `-h` option tells RIPEM how to handle plaintext input that contains mail headers. "Mail headers" are defined to be all the lines at the beginning of a message, up to the first blank line.

The syntax is: `-h` *header_opts*, where *header_opts* is one or more of the letters `i`, `p`, and `r`. `i` tells RIPEM to **i**nclude the headers as part of the message to be encrypted. `p` tells RIPEM to **p**repend the headers to the encrypted output. `r` tells RIPEM to examine the message headers, looking for "To:" and "cc:" lines. Any **r**ecipients named on those lines are included as recipients to RIPEM's encryption.

The default is "`-h i`", which causes message headers to be included in the plaintext being encrypted, but no other header processing is done. This is equivalent to treating the message as if it does not contain mail headers at all.

A useful combination is "`-h pr`", which extracts recipients' names from the mail headers at the beginning of the input, copies the mail headers unmodified and unencrypted to the beginning of the output, and then discards the headers before encrypting the rest of the message. This combination is suitable for instances in RIPEM is being used to encrypt a message after it has been prepared by a mail program but before it has been sent.

To find the recipients’ public keys for encrypting the message, RIPEM first searches for a certificate you previously created using the validation mode (see Receiving in Validation Mode below). The value given by `-r` is usually the common name in the full distinguished name. If it can’t find a certificate, RIPEM searches for version 1.1 and earlier style public keys first in `pubkeys` in the RIPEM home directory and then in other files specified with the `-p` option.

When RIPEM has found the public keys, it outputs the recipient’s full distinguished name and certificate status to the debug stream, which is your terminal unless you redirected it using `-Z`. This information is important for two reasons: first, the certificate you made for the recipient may have expired or otherwise become invalid; and more importantly, your must make sure that the full distinguished name is what you expect. If you specify `-r Bob`, RIPEM may find a certificate for “common name = bob, organization = Gadgets” whereas you really want “common name = Bob, organization = Widgets”. In this case, the message is encrypted for the wrong Bob and you should not send it.

Decrypting a Message

The `-d` option specifies that RIPEM is decrypting a message.

During decryption, RIPEM looks at the values of the `-H`, `-i`, `-o`, `-u`, `-p`, `-s`, `-y`, and `-Y` options discussed above.

If RIPEM cannot decrypt the input message, or if the input message fails the signature check, RIPEM will generate no output in the `-o` file or on standard output. Instead, it will issue an error message on the standard error device, which is usually your terminal. In addition, RIPEM returns a value of zero to the operating system if the message decrypts properly, and it returns a non-zero value if there are problems. This is typical behavior for programs under Unix and MS-DOS and allows you to write command scripts which check to see whether decryption proceeded properly.

If RIPEM does decrypt the message properly, it will write the decrypted plaintext to the `-o` file or to standard output. The output contains only the original plaintext (subject to any modifications performed by the `-h` option used by the sender of the message). It does not include any mail headers or other superfluous text added to the encrypted message--for instance, by a mail system--after the encryption.

Receiving in Validation Mode

Using the `-d` option, the first time you receive a RIPEM message from someone, RIPEM will tell you that the public key in their self-signed certificate has not been validated by you and it will display the digest on their self-signed certificate. After verifying their self-signed certificate digest as explained in “Self-Signed Certificates” above, you can use your own private key to create a certificate for them by receiving the message again in validation mode. Use `-d` and all the other flags as before, but also add the `-v` option which specifies the number of months from today that the certificate will be valid. A typical value is 24 for two years. After this time, the status displayed by RIPEM when you use the certificate will change from VALID to EXPIRED. When the certificate expires you should contact the correspondent again to make sure their public key is still valid and repeat the procedure.

Advanced Usage

Specifying Encryption Algorithm (-A)

By default, RIPEM encrypts messages using DES in Cipher Block Chaining (CBC) mode. This is the data encryption algorithm and mode used by Internet PEM-conformant software.

Although DES has proven quite resistant to theoretical attacks of cryptanalysts for 16 years, many cryptologists have expressed concern over DES's relatively small keyspace, which leaves it potentially vulnerable to brute-force attack by a well-funded opponent. (DES keys are 56 bits long.) One obvious solution to the keyspace problem is to use multiple passes of DES.

A few years ago, IBM suggested a particular multi-pass usage of DES called Encrypt-Decrypt-Encrypt (EDE). In EDE usage (sometimes called Triple-DES), each 64-bit block is encrypted with a 56-bit key we'll call *key1*. The result of that encryption is decrypted with a second 56-bit key called *key2*. Finally, the 64-bit result of that decryption is encrypted with *key1*. This use of DES results in a dramatic increase of keyspace from 2^56 keys to 2^112 keys.

RIPEM implements this use of DES, with Cipher Block Chaining applied after each triple encryption, and refers to it as DES-EDE-CBC. When encrypting, specify `-A des-ede-cbc` to select this mode. When decrypting, RIPEM automatically detects the encryption algorithm used and decrypts appropriately.

DES-EDE has not been widely adopted by the cryptographic community. In particular, DES-EDE encryption is not conformant with Internet PEM as of this writing. Therefore, use the default mode (which can also be explicitly requested via `-A des-cbc`) for all but your most critical messages. In addition, consider that there is some performance degradation associated with Triple-DES.

Specifying Debug Mode (`-D` and `-Z`)

RIPEM uses the debug output to show the status of recipient public keys while encrypting a message and to show the status of sender's public key and signature while receiving a message. RIPEM also uses the debug output to show self-signed certificate digests and other important messages which come under the default level.

Users experiencing problems with RIPEM, or simply wishing to examine the inner workings of the program, can use the -D option to increase the debug level which will cause RIPEM to print informative messages while it executes. Debugging options were originally implemented in RIPEM for development purposes, but have been left in place for the benefit of curious users. Specify `-D` *debuglevel* to turn on debug messages. *Debuglevel* is an integer specifying the amount of debug output desired. 1 is the default level for required information on senders and recipients, while 4 is the maximum value currently implemented.

Debug messages are normally written to the standard error output, which is usually your terminal screen. To write debug messages to a file, use the `-Z` *debugfile* option.

Specifying Encryption Mode (`-m`)

By default, in encryption (`-e`) mode RIPEM encrypts a message, encodes it to printable ASCII characters, and signs the message. This processing corresponds to the `-m encrypted` command line option. With non-default values for the `-m` option, RIPEM can perform other types of processing in `-e` mode.

`-m mic-clear` specifies that the message is signed, but not encrypted. The body of the message is left in plaintext, so that the recipient can read it without decryption software of any sort. If the recipient wishes to verify the signature, however, he/she will have to use RIPEM in `-d` mode as usual.

`-m mic-only` also specifies that the message is signed, but not encrypted. However, the body of the message is printably encoded into ASCII characters as per RFC 1113. This encoding expands the size of the message by about 33% and adds no security; it simply helps guarantee that the message will survive hostile mail software verbatim. In practice, `mic-only` mode is infrequently used.

Specifying Your Username (Email Address) (`-u`)

As described above, you can specify your username (or more correctly, your email address) via the `-u` option or via the environment variable `RIPEM_USER_NAME`. The default is for RIPEM to attempt to determine your email address from the running system. On MS-DOS, RIPEM defaults to the name “me.”

RIPEM makes use of this information in all three modes: key generation, encryption, and decryption. If you have a number of email addresses that you wish to be regarded as equivalent, you can specify your username as a list of comma-separated email addresses. During key generation and encryption, RIPEM will use the first name in the list as your username. During decryption, RIPEM will search your private key file for your private component under each one of the comma-separated names until it succeeds in finding a matching key in the file.

These features make it easier to use RIPEM if you have mail forwarded to a primary mail account from a number of other email addresses. To make use of the multiple username capability, you must edit your private key file to include multiple `User:` lines.

Specifying the Key to Your Private Component (`-k, -K`)

By default, RIPEM prompts you interactively when it needs to know the key to the private component of your public key. However, with some loss in security, it is possible to inform RIPEM of this key by other means. This capability may be useful for instances when RIPEM is invoked by another program, or for when you are testing or benchmarking.

You can specify the key to your private key via the `-k` option; specify `-k` *keytokey* on the command line. As a special case, you can specify `-k -`. Specifying `-` as your password on the command line causes RIPEM to read the password as the first line from standard input.

If -k is not specified, RIPEM will check the value of the environment variable `RIPEM_KEY_TO_PRIVATE_KEY`. If this variable exists and has a non-null value, its value will be used as the key to the private key; otherwise, RIPEM will prompt you on your terminal for this key.

When using `-c` to change your key to private key, you can also use upper case `-K` to specify the new key instead of letting RIPEM prompt you for the new key. This is useful so that all values can be supplied on the command line when invoking RIPEM from another application.

Specifying the key to your private key via `-k` or via the environment variable is generally less secure than typing it interactively. Although RIPEM erases its command line arguments shortly after startup, there is a brief window of time during which other users on a Unix system could view your command line arguments by using the `ps` command. Likewise, other users could determine the value of your `RIPEM_KEY_TO_PRIVATE_KEY` variable by various means, especially if they had physical access to your terminal. Therefore, these options should be used with caution, if at all.

Using UNIX Mail Programs and Utilities

This section suggests techniques for using RIPEM in conjunction with popular Unix mail programs. Use of the C-Shell is assumed.

It is possible, of course, to compose a message in a text editor, save the message to a file, run RIPEM to encrypt the message, start your mailer, insert the encrypted file into a message, and then send the message. In fact, the encryption and mailing can be done on separate systems, with appropriate file transfers. However, on most Unix systems it is possible to eliminate several of these tedious steps.

Setting Up Your Environment

It is recommended that Internet-connected Unix users include the following in their `.cshrc` file:

```
setenv RIPEM_HOME_DIR  ~/.ripemhome       (must already be created)
setenv RIPEM_SERVER_NAME        ripem.msu.edu  (for 1.1 compatibility)
setenv RIPEM_USER_NAME          (Your email address; e.g., smith@bigu.edu)
```

Create a shell script to encrypt RIPEM messages. Place the following lines in a file named `ripem-encrypt`, put this file in a directory mentioned in your path, and give it execute permission. (This file is available in the RIPEM distribution, in the `util` directory.)

```
#!/bin/sh
tempfile=/tmp/msg-`whoami`
ripem -e -h pr -i $1 -o $tempfile
cp $tempfile $1
rm $tempfile
```

Create a shell script to decrypt RIPEM messages. As above, place these lines in the file `ripemd`:

```
ripem -d | more
```

Create a shell script to help reply to encrypted RIPEM messages. Place this in `ripemr`:

```
ripem -d -h pr | quote -h
```

Include the following lines in the file `.mailrc` in your home directory. (A sample `.mailrc` can be found in the RIPEM distribution, in the `util` directory.)

```
set editheaders
set EDITOR=ripem-encrypt
```

Creating a RIPEM Public Key

The initial generation of keys can proceed something like this:

```
cd ~
ripem -g -R eks
```

Type random garbage at the keyboard when prompted.

Type in a secret password when prompted for the password to your private key. Type it again when prompted for verification.

Now you can create signed messages and send them to your correspondents. They will ask you for your self-signed certificate digest and validate your public key.

Encrypting an Email Message Using "mail"

If you are using the "`mail`" package that comes with many Unix systems, you can use the following procedure to compose, encrypt, and send a message. In this example, your input is in bold.

$ **mail smith@bigu.edu**

```
Subject: Greetings
This is a test message
~e
Enter password to private key: (Type your password here.)
(Type Control-D)
$
```

The `~e` command to `mail` was originally designed to **e**dit the message being composed. Nowadays, however, it is rarely used (in favor of the `~v` **v**isual edit command). The tricks described above effectively turn `~e` into an "**e**ncrypt" command on SunOS and some other Unix systems. On some Unix systems, however, the `mail` command does not interpret the `editheaders` option. On those systems, you need to use a different approach, which requires you to type the recipient's address twice:

```
$ mail smith@bigu.edu
Subject: Greetings
This is a test message
~| ripem -e -r smith@bigu.edu
Enter password to private key: (Type your password here.)
(continue)
(Type Control-D)
$
```

Decrypting an Email Message Using "mail"

```
$ mail
Mail version ......
"/usr/spool/mail/jones": 1 message 1 new
>N  1 smith@bigu.edu       Wed Sep 30 22:38    29/1119  Greetings
& pipe ripemd
Pipe to: "ripemd"
Enter password to private key: (Type your password here.)
(The plaintext message is displayed.)
"ripemd" 29/1119
& q
$
```

Encrypting an Email Message Using "Mush"

`Mush` is a mail package that is compatible with `mail`, but provides additional capabilities. The procedure described above for encrypting messages with mail also works with mush.

Decrypting an Email Message Using "Mush"

The procedures described for `mail` also work with `mush`. However, `mush`'s greater power allows you to configure it to be easier to use than `mail`, especially in *curses* mode. Configure `mush` by creating a file named `.mushrc` in your home directory and placing the following lines in it:

```
set edit_hdrs
set editor=/home/scss3/mrr/bin/ripem-encrypt
set visual=/usr/ucb/vi
bind-macro D :pipe ripemd\n
bind-macro R r~f\n~\|ripemr\n~v\n
```

To decrypt and display a message in curses mode, simply type the letter `D` while the cursor is positioned on the corresponding message index line:

```
  56  U  Mark Riordan <mrr@cl-next Nov  3, (13/460) "test"
```
(press **`D`***)*
```
:pipe ripemd
```
(This is generated automatically by the `mush` macro.)
```
Enter password to private key:
```
(Type the password.)
(The plaintext is displayed.)
```
  56    Mark Riordan <mrr@cl-next Nov  3, (13/460) "test"
mrr 56: ...continue...
```

To reply to an encrypted message, type `R` while the cursor is positioned on the corresponding message line. The `R` macro decrypts the message, quotes the text of the message with the traditional "> " line prefix, and enters a visual editor. For this procedure to work, you must have compiled the `quote` program--located in the `ripem/util` directory--and installed it in a directory on your path.

```
  56  U  Mark Riordan <mrr@cl-next Nov  3, (13/460) "test"
```
(press **`R`***)*
```
To: mrr@museum.cl.msu.edu
Subject: Re: test 5

~f
Including message 56 ... (30 lines)
(continue editing letter)
~|ripemr
```
(All of this is generated automatically by the macro.)
```
Enter password to private key:
```
(Type the password, to decrypt the message.)
(`Mush` calls up a visual editor on a quoted copy of the plaintext of the message to you. Compose your reply and exit the editor. To exit vi, for instance, type **`ZZ`***.)*
```
(continue editing letter)
~e
```
(Type this to encrypt your reply)
```
Enter password to private key:
```
(Type the password, to encrypt your reply.)
```
(continue editing letter)
```
(Type ***control-D*** *to finish the outgoing letter and have `mush` send it.)*

Using RIPEM with ELM's MIME Features

The popular Unix mailer Elm has recently been extended to provide MIME (Multipurpose Internet Mail Extension) capabilities. (MIME is not particular to Elm, and MIME support is or will soon be available in other mailers as well.) RIPEM can be used with Elm's MIME support, though somewhat awkwardly. Below are preliminary instructions for interfacing RIPEM with Elm.

Edit your local or system `mailcap` file to add the following lines:

```
# This entry is for reading a mail message encoded by RIPEM
application/ripem;      ripem -d ; copiousoutput
```

This is necessary only on the receiving end.

On the sending end:

- Compose the message and encode it, saving the result in a temporary file, say, `mymsg`.

- Start the mail message in Elm and include the temporary file as a MIME message with a line of the form:
  `[include mymsg application/ripem]`

- Send the message.

Upon receipt, Elm will recognize it as a MIME message and start RIPEM to decode it. If the receiving mailer does not understand MIME, the usual methods of decrypting RIPEM messages can be used.

Using RIPEM with the MH Mailer

The RIPEM source distribution contains two Perl scripts that facilitate reading and sending encrypted messages with RIPEM. `display-ripem` decrypts and displays RIPEM messages; `send-ripem` encrypts and sends messages. These utilities, written by Marc VanHeyningen, can be found in the `ripem/util` directory. See the source code for documentation.

Using RIPEM with EMACS

Jeff Thompson has written functions for the powerful EMACS editor to facilitate use of RIPEM. These functions are in the file `ripem/util/ripem.el` in the RIPEM source distribution. To enable the use of these functions, edit your ~/.emacs file and add to it a line like:

```
(load "/home/local/ripem/util/emacs/ripem.el")
```

(Modify the file name in quotes to be the correct path to the `ripem.el` file.)

To encrypt a message, enter the message, including the To: and cc: lines, into an EMACS buffer and type:

***Esc* `x ripem-encrypt`**

To decrypt a message, load it into an EMACS buffer and type:

***Esc* `x ripem-receive`**

To receive a message in validation mode, load it into an EMACS buffer and type:

***Esc* `x ripem-validate-and-receive`**

Online help is available by typing:

Ctrl-h* `f` *function_name

where *function_name* is one of: `ripem-generate`, `ripem-encrypt`, `ripem-sign`, `ripem-sign-clear`, `ripem-receive`, `ripem-validate-and-receive`, `ripem-change-password`, or `ripem-list-users`.

Upgrading From RIPEM 1.1

The most important part of upgrading from RIPEM version 1.1 or earlier is to create a self-signed certificate. This is done by changing your key to private key. Follow these steps:

- Create your RIPEM home directory. From now on, use `-H` to specify this directory to RIPEM, or set the RIPEM_HOME_DIR environment variable. For Unix, a good home directory is `~/.ripemhome` or for MS-DOS a good directory is `\ripemhom`.
- Use `ripem -c` to change your password. The old and new passwords may be the same. Specify `-p` and `-s` for your input public and private key files as you did for RIPEM 1.1, but do *not* specify `-P` or `-S` output files. Instead use `-H` to specify the RIPEM home directory. Note RIPEM does not change your public or private key. It simply places your private key under a different protection key. RIPEM creates a new private key file in the home directory so you don't need the file given by `-s` anymore.
- RIPEM will notice that you don't already have a self-signed certificate, and will create one for you and add it to `pubkeys` in your RIPEM home directory. The validity period defaults to 24 months from now, but you may change the validity length with `-v`.
- RIPEM displays your self-signed certificate digest which you should record for giving out to others.

You do not need to put your self-signed certificate on a server or in your "finger". The correct way to disseminate it is to prepare a RIPEM signed message and send it to your correspondents. They will ask you for your self-signed certificate digest and will validate your public key. See Self-Signed Certificates above for a description of how to establish a trusted channel.

You can still use `-d` to decrypt version 1.1 and earlier messages if you indicate the public key sources using `-p` or `-y`. See Managing Keys According to RIPEM 1.1 and Earlier above. When RIPEM displays the sender's name it will warn you that the sender's public key is not validated.

Also, you can still send messages to users of version 1.1 or earlier. They still have access to your public key according to the old model, and current RIPEM messages are backwards compatible. When encrypting a message using `-e` you can specify sources of unvalidated keys for the recipients using `-p` or `-y`. RIPEM will show that the keys are not validated when it displays the recipient information.

Preferably, each of your correspondents should upgrade to the current RIPEM and you should receive a message from each of them in validation mode to create a certificate for them. Once you have done this, you should not need to use `-p` or `-y`.