Mailcrypt: An EMACS Interface to PGP

Version 3.5.9
2010-02-14

Patrick J. LoPresti <patl@lcs.mit.edu>
Leonard R. Budney <lbudney@pobox.com>
Brian Warner <warner@lothar.com>

Copyright © 1995 Patrick J. LoPresti

Copyright © 1998 Leonard R. Budney

Copyright © 2001 Brian Warner

The Mailcrypt program and this documentation are published as free software. You may redistribute and/or modify them under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2, or (at your option) any later version.

Mailcrypt is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with GNU Emacs; see the file COPYING. If not, write to the Free Software Foundation, 675 Mass Ave, Cambridge, MA 02139, USA.

1 Introduction

Mailcrypt is an Emacs Lisp package which provides a simple but powerful interface to cryptographic functions for mail and news. With Mailcrypt, encryption becomes a seamlessly integrated part of your mail and news handling environment.

This manual is long because it is complete. All of the information you need to get started is contained in this Introduction alone.

1.1 Prerequisites

Mailcrypt requires version 19 or higher of GNU Emacs. Mailcrypt has been tested on a variety of systems under both FSF Emacs and XEmacs.

Mailcrypt requires Pretty Good (tm) Privacy, usually known as PGP. This document assumes that you have already obtained and installed PGP and that you are familiar with its basic functions. The best way to become familiar with these functions is to read the *PGP User's Guide*, at least Volume I.

For more information on obtaining and installing PGP, refer to the MIT PGP home page at `http://web.mit.edu/network/pgp.html`.

Although Mailcrypt may be used to process data in arbitrary Emacs buffers, it is most useful in conjunction with other Emacs packages for handling mail and news. Mailcrypt has specialized support for Rmail (see Section "Reading Mail with Rmail" in *The GNU Emacs Manual*), VM (see Section "Introduction" in *The VM User's Manual*), MH-E, and Gnus (see Section "Overview" in *The Gnus Manual*). Information on the general use of these packages is beyond the scope of this manual.

1.2 Installation

If Mailcrypt is not installed on your system, obtain the latest version from the Mailcrypt home page at `http://mailcrypt.sourceforge.net` and follow the instructions in the file `INSTALL`.

Next, decide what version of PGP you are using. Versions 3.5 and higher of Mailcrypt support multiple versions of PGP. To choose a version, add the following lines to your `.emacs` file:

```
(load-library "mailcrypt") ; provides "mc-setversion"
(mc-setversion "2.6")    ; for PGP 2.6 (default); also "5.0" and "gpg"
```

Next, teach your Emacs how and when to load the Mailcrypt functions and install the Mailcrypt key bindings. Almost all Emacs major modes (including mail and news handling modes) have corresponding "hook" variables which hold functions to be run when the mode is entered. All you have to do is add the Mailcrypt installer functions to the appropriate hooks; then the installer functions will add the Mailcrypt key bindings when the respective mode is entered.

Specifically, begin by placing the following lines into your `.emacs` file (or the system-wide `default.el` file):

```
(autoload 'mc-install-write-mode "mailcrypt" nil t)
(autoload 'mc-install-read-mode "mailcrypt" nil t)
```

```
(add-hook 'mail-mode-hook 'mc-install-write-mode)
```

Then add additional lines for your own mail and news packages as described below.

1.2.1 Hooking into Rmail

To hook Mailcrypt into Rmail, use the following lines:

```
(add-hook 'rmail-mode-hook 'mc-install-read-mode)
(add-hook 'rmail-summary-mode-hook 'mc-install-read-mode)
```

Using Emacs version 20.3 or higher, you should use the following lines instead:

```
(add-hook 'rmail-show-message-hook 'mc-install-read-mode)
(add-hook 'rmail-summary-mode-hook 'mc-install-read-mode)
```

1.2.2 Hooking into VM

To hook Mailcrypt into VM, use the following lines:

```
(add-hook 'vm-mode-hook 'mc-install-read-mode)
(add-hook 'vm-summary-mode-hook 'mc-install-read-mode)
(add-hook 'vm-virtual-mode-hook 'mc-install-read-mode)
(add-hook 'vm-mail-mode-hook 'mc-install-write-mode)
```

1.2.3 Hooking into MH-E

To hook Mailcrypt into MH-E, use the following lines:

```
(add-hook 'mh-folder-mode-hook 'mc-install-read-mode)
(add-hook 'mh-letter-mode-hook 'mc-install-write-mode)
```

1.2.4 Hooking into Gnus

To hook Mailcrypt into Gnus, use the following lines:

```
(add-hook 'gnus-summary-mode-hook 'mc-install-read-mode)
(add-hook 'message-mode-hook 'mc-install-write-mode)
(add-hook 'news-reply-mode-hook 'mc-install-write-mode)
```

1.2.5 Hooking into Mew

To hook Mailcrypt into Mew, use the following lines:

```
(add-hook 'mew-message-mode-hook 'mc-install-read-mode)
(add-hook 'mew-summary-mode-hook 'mc-install-read-mode)
(add-hook 'mew-draft-mode-hook 'mc-install-write-mode)
```

Note that Mew already has extensive support for MIME-encoded encrypted and/or signed messages (using the “multipart/encrypted” and “application/pgp-encrypted” formats specified by RFC3156). Using MailCrypt within Mew is most useful for traditional “inline” armored encrypted/signed messages.

1.3 Command Overview

All Mailcrypt commands are (by default) activated by three-character key sequences which begin with `C-c /`. The most common operations are:

Encrypting a Message
: `C-c / e` encrypts a message using the recipient's (or recipients') public key(s). See Section 2.1 [Encrypting a Message], page 4.

Decrypting a Message
: `C-c / d` decrypts a message using your secret key. See Section 2.4 [Decrypting a Message], page 5.

Signing a Message
: `C-c / s` clearsigns a message using your secret key. See Section 2.2 [Signing a Message], page 4.

Verifying a Signature
: `C-c / v` verifies the signature on a clearsigned message using the sender's public key. See Section 2.5 [Verifying a Signature], page 6.

These functions and others are documented in detail in the following chapters.

Any time you are composing or reading mail or news, you can get a summary of the available commands by typing `C-h m`. If you are running Emacs under X, an even easier way to see the available commands is to access the `Mailcrypt` pull-down menu.

2 General Use

By default, Mailcrypt assumes you are using one of the PGP 2.6.x versions. This permits backward compatibility for the millions of satisfied users of Mailcrypt 3.4 worldwide. If you wish to specify a different version of PGP, use the `mc-setversion` function. Its action is the same as setting the variable `mc-default-scheme`. For a list of supported versions, press the tab key. "2.6" means 2.6.x, the original (and default). "5.0" is pgp 5.0. "6.5" is pgp 6.5. "gpg" is GnuPG.

Mailcrypt works by providing two minor modes for interfacing with cryptographic functions: `mc-read-mode` and `mc-write-mode`. `mc-read-mode` provides key bindings for processing messages which you have received; `mc-write-mode` provides key bindings for processing messages which you are about to send. These minor modes will indicate when they are active by placing a characteristic string in the mode line (see Section 6.3 [Mode Line], page 19). They will also add a `Mailcrypt` pull-down menu to the menu bar.

The normal installation procedure (see Section 1.2 [Installation], page 1) will arrange for the appropriate mode to be active when you read and compose mail and news. But you may want to use Mailcrypt's functions at other times; to do so, you can call `mc-install-read-mode` or `mc-install-write-mode` directly. For example, if you were editing a file in Text mode and wanted to digitally sign it, you would type `M-x mc-install-write-mode`, then `C-c / s` (see Section 2.2 [Signing], page 4).

Once one of the Mailcrypt modes is active, you can get a summary of the available functions by typing `C-h m` or by examining the `Mailcrypt` pull-down menu.

The description of each function below includes which of the modes has a binding for that function.

2.1 Encrypting a Message

The function `mc-encrypt` will encrypt a message in the current buffer. `mc-write-mode` binds this function to `C-c / e` by default.

When this function is called, Mailcrypt will prompt you for a comma-separated list of recipients. If called from a mail composition buffer, the recipient list will default to the Email addresses in the '`To`', '`CC`', and '`BCC`' lines of the message.

If you want to be able to decrypt the message yourself, you need to add yourself to the recipient list. If you always want to do so, set the variable `mc-encrypt-for-me` to `t`. (Note that Mailcrypt overrides the PGP "encrypttoself" flag; use this variable instead.)

If you provide an empty recipient list, Mailcrypt will ASCII-armor the message without encrypting it.

Once you have edited the recipient list to your satisfaction, type `RET` to accept it. You will then be asked whether you want to sign the message; answer `y` or `n`. You can avoid this question by setting the variable `mc-pgp-always-sign`: A value of `t` means "yes", a value of `'never` means "no".

If you elect to sign the message, Mailcrypt will prompt you for the appropriate passphrase unless it is cached (see Chapter 4 [Passphrase Cache], page 15).

Mailcrypt will then pass the message to PGP for processing. Mailcrypt will call the functions listed in `mc-pre-encryption-hook` and `mc-post-encryption-hook` immediately before and after processing, respectively. The encrypted message will then replace the original message in the buffer. You can undo the encryption with the normal Emacs undo command `C-x u` (see Section "Undoing Changes" in *The GNU Emacs Manual*).

If an error occurs, Mailcrypt will display an appropriate diagnostic. If you do not have the public key for one of the specified recipients, Mailcrypt will offer to try to fetch it for you (see Chapter 5 [Key Fetching], page 16).

If you want to use a secret key other than your default for signing the message, pass a prefix argument to `mc-encrypt`. (That is, type `C-u C-c / e`.) Mailcrypt will prompt for a string and will sign with the first key on your secret keyring which matches that string. It will be assumed that you want to sign the message, so you will not be prompted. See the next section, Section 2.2 [Signing a Message], page 4, for information about which key is used by default to sign the message.

2.2 Signing a Message

The function `mc-sign` will clearsign a message in the current buffer. `mc-write-mode` binds this function to `C-c / s` by default.

When this function is called, Mailcrypt will prompt you for the appropriate passphrase unless it is cached (see Chapter 4 [Passphrase Cache], page 15).

Mailcrypt will then pass the message to PGP for processing. Mailcrypt will call the functions listed in `mc-pre-signature-hook` and `mc-post-signature-hook` immediately before and after processing, respectively. The signed message will replace the original message in the buffer. *Do not* edit the message further with the signature attached, because the signature would then be incorrect. If you discover you need to edit a message after you have signed it, remove the signature first with the normal Emacs undo command `C-x u` (see Section "Undoing Changes" in *The GNU Emacs Manual*).

The variable `mc-pgp-user-id` controls which secret key is used for signing. To use a different secret key, pass a prefix argument to `mc-sign`. (That is, type *`C-u C-c / s`*.) Mailcrypt will prompt for a string and will sign with the first key on your secret keyring which matches that string.

The default key for signing is the first one on the secret key ring which matches the string `mc-pgp-user-id`; this defaults to `(user-login-name)`. Note that this differs from PGP's normal default, which is to use the first of *all* of the secret keys. To mimic PGP's behavior, set this variable to `""`. This variable is specific to pgp 2.6.x; `mc-pgp50-user-id` and `mc-gpg-user-id` are the corresponding variables for pgp 5.0 and GnuPG.

If you have multiple secret keys with the same name (perhaps you generate a new key every few years, but keep the expired keys on your secret key ring so you can decrypt old messages), you may want to use a hex keyid in `mc-gpg-user-id` or equivalent. A simple name will cause mailcrypt to use the first matching secret key, which may not be the most recent one. Using a hex keyid will force the encryption program to use that exact secret key for signing. Put something like the following in your `.emacs`:

```
(setq mc-gpg-user-id "0x03A5E108")
```

2.3 Inserting a Public Key Block

The function `mc-insert-public-key` will extract a key from your public keyring and insert it into the current buffer. `mc-write-mode` binds this function to *`C-c / x`* by default.

This function is useful for sending your public key to someone else or for uploading it to the key servers (see Section 9.2 [Key Servers], page 22). The inserted key will be the first one on your public key ring which matches the string `mc-pgp-user-id` (see Section 2.1 [Encrypting a Message], page 4).

You may want to insert a different public key instead; for example, you may have signed someone's key and want to send it back to them. To do so, pass a prefix argument to `mc-insert-public-key`. (That is, type *`C-u C-c / x`*.) You will be prompted for a string; the first key on your public key ring which matches that string will be inserted.

2.4 Decrypting a message

The function `mc-decrypt` will decrypt a message in the current buffer. `mc-read-mode` binds this function to *`C-c / d`* by default.

When this function is called, Mailcrypt will prompt you for the appropriate passphrase unless it is cached (see Chapter 4 [Passphrase Cache], page 15).

The encrypted message will then be passed to PGP for processing. If you are not in a mail buffer, the decrypted message will replace the encrypted form. If you are in a mail buffer, you will be prompted whether to do the replacement.

If you answer *`n`*, you will be placed in a new mail reading buffer to view the decrypted message. This new mail reading buffer will have no corresponding disk file; its purpose is to provide you with all of your usual reply and citation functions without requiring you to save the message in decrypted form. Type *`q`* to kill this buffer.

You can avoid the question of whether to replace the encrypted message by setting the variable `mc-always-replace`. A value of `t` means "yes"; a value of `'never` means "no".

If the encrypted message is also signed, PGP will attempt to verify the signature. If the verification fails because you lack the necessary public key, Mailcrypt will offer to fetch it for you (see Chapter 5 [Key Fetching], page 16).

Look in the `*MailCrypt*` buffer to see the result of the signature verification.

2.5 Verifying a Signature

The function `mc-verify` will verify the cleartext signature on a message in the current buffer. `mc-read-mode` binds this function to `C-c / v` by default.

When this function is called, Mailcrypt will pass the message to PGP for processing and report whether or not the signature verified.

If the signature failed to verify because you lack the necessary public key, Mailcrypt will offer to fetch it for you (see Chapter 5 [Key Fetching], page 16).

2.6 Snarfing a Key

The function `mc-snarf` will add to your keyring any keys in the current buffer. `mc-read-mode` binds this function to `C-c / a` by default.

This function is useful when someone sends you a public key in an Email message.

3 Remailer Support

This is a long chapter describing an advanced feature; you may want to skip it on first reading.

3.1 Remailer Introduction

There are several anonymous remailer services running on the Internet. These are programs that accept mail, strip off information that would identify the origin of the message, and forward the mail to the designated recipient. This simple scheme alone, however, is insecure if the anonymous remailer becomes compromised (or if the remailer was set up by an untrustworthy party in the first place). Whoever controls the remailer will have access to the identities of senders and recipients.

One solution to this is to use *chains* of remailers that send encrypted messages. For example, suppose Bill wishes to send a message to Louis using a chain of remailers A, B, and C. He writes the message (possibly encrypting it for Louis), then encrypts the result (including the fact that Louis is the recipient) using a public key supplied by remailer C. Then he encrypts this result using a public key supplied by remailer B. Then he encrypts this result using a public key supplied by A and sends the message to A.

When A receives the message, it decrypts the message with its key to produce something encrypted for B, learns that the next remailer in the chain is B, strips off the information that the message came from Bill, and sends the message on to B. B then decrypts, learns that the next remailer in the chain is C, strips off the information that the message came from A, and sends the result to C. C then decrypts, learns that the destination is Louis, strips off the information that the message came from B, and sends the result to Louis. With this arrangement, only A knows that the original message came from Bill, and only

C knows that the intended recipient is Louis. In general, the sender and recipient can both be known only to someone who has compromised all remailers in the chain.

If Bill wishes, he can include an encrypted "response block" in his message to Louis, which defines a remailer chain that Louis can use to reply to Bill. Louis can use this chain without knowing who Bill is – only the last remailer in the chain need know the final recipient. Bill can also establish a *pseudonym* for use in signing his anonymous messages.

More sophisticated systems split the message into multiple pieces to further disguise the path it takes through the network. Special client programs are used to construct and encrypt the pieces.

Mailcrypt includes facilities for sending messages via remailers, for defining chains of remailers, for generating response blocks, for using pseudonyms, and for interfacing with remailer client programs.

3.2 Types of Remailers

There are currently three classes of remailer networks in use, not counting the original single-hop address-rewriter machines (like the late penet.fi).

Cypherpunk (Type 1)

Machines in the original cypherpunk remailer network accept messages with commands to send a portion of the message out to another machine. By wrapping the final message text in layers of encryption like an onion, the message is sent through chains of remailers, each machine decrypting one layer and sending the rest out to the next hop.

One disadvantage of this scheme is that the message gets smaller by a fairly constant amount on each hop, making traffic analysis easier to perform.

Mailcrypt automates the process of wrapping your final message in layers of encryption for each remailer along the desired chain. Type 1 remailers are handled by `mc-remailer-scheme-type1`.

Mixmaster (Type 2)

The next generation of remailers use a special client program (written in C) to encode the message differently. The message is broken up into multiple identically-sized pieces, which follow separate paths through the network, and are recombined at the far end. At each hop, random padding is added to make sure that *all* inter-node messages are exactly the same size. This makes traffic analysis more difficult. Periodic dummy messages are also sent to further complicate attacks.

The most common type-2 client program for Unix is named “mixmaster”. There are two different versions, with fairly different interfaces. Mailcrypt supports both.

The older version comes from a package named “mix-2.0.3”, and has an interface that modifies a message in-place. The mailcrypt interface to this is through `mc-remailer-scheme-type1` and behaves much like the normal encryption interface: you hit the button and your email is modified in place; the To: header is replaced with the target of the first remailer in the chain. As you can imagine, this API is problematic, as large messages must be split into multiple pieces.

The modern type-2 client program is an updated version of “mixmaster”, from a package named “mixmaster-2.9.0” (distributed at `http://mixmaster.sourceforge.net/`. This version does not modify the message in place; rather it accepts an email message on stdin and sends the resulting series of packets directly. The mailcrypt interface to the new mixmaster is through `mc-remailer-scheme-mixmaster`. When you chose “encrypt for remailer” within this scheme, the message is sent directly to the client program and delivered immediately, bypassing your MUA mode’s normal “send” command.

Mixminion (Type 3, under development)

A new generation of remailer networks is under currently development (at `http://www.mixminion.net/`). One notable improvement is the addition of automatic reply blocks. A major flaw of the type-2 network is that to give someone the ability to reply to a message, you have to give them a type-1 reply block. This makes the response more vulnerable to traffic analysis. Mixminion message can be formed in such a way that the response path is automatically embedded in the outgoing message.

Mixminion uses a client program (called “mixminion”) which behaves just like the modern “mixmaster” client. Through `mc-remailer-scheme-mixminion`, mailcrypt supports a command to send the body of the current message to the mixminion client.

Mixminion is under development, please check the web page for the current status before using it.

The remailer scheme currently in use is determined by the `mc-default-remailer-scheme` variable. Set this to one of `mc-remailer-scheme-type1`, `mc-remailer-scheme-mixmaster`, or `mc-remailer-scheme-mixminion`. For example, use the following in your `.emacs` to make *`C-c / r`* always use the modern mixmaster client:

```
(setq mc-default-remailer-scheme 'mc-remailer-scheme-mixmaster)
```

3.3 Remailer Quick Start

To use Mailcrypt’s remailing facilities, you need to configure them first. Begin with the following steps:

1. Download Levien-format list of remailers from `http://www.tahina.priv.at/~cm/stats/rlist.txt` (as of 2007-03-01) and save the file to ‘`~/.remailers`’. See the variable `mc-levien-file-name` to chnage the file name anad location. Mailcrypt will parse this the first time you access a remailer function.
2. Look over the `.remailers` file and find the ones you want to use. The lines that list remailers and their capabilities look like ones below. It is best to consuts remailer statistics page to determine the most reliable candidats for chaining.

   ```
   $remailer{"frell"} = "..."
   $remailer{"starwars"} = "..."
   ```

3. Add their PGP public keys to your keyring. You can for an armored keyring full of remailer public keys. Note that Mailcrypt *requires* that you have the public keys

of all the remailers you want to use, and therefore that the remailers support PGP encryption.

The list of remailers and their keys (as of 2007-03-01) can be found at `http://www.noreply.org/echolot/thesaurus/`. With gpg, after each *key* has been saved to a file, the entries can be imported into separate public keyring with command:

```
gpg --no-options --no-default-keyring \
    --keyring ~/.gnupg/pubring-remailer.gpg \
    --import *.key
```

In order for gpg to use separate keyring, the new keyring file must be configured to `~/.gnupg/gpg.conf` by adding this line:

```
keyring pubring-remailer.gpg
```

> *Note:* Downloading the remailer list and their keys need only be done once, although repeating them from time to time is probably a good idea, since remailers come and go.

Now test the remailer functions. First compose an outgoing Email message (using *C-x m*, for example) addressed to yourself. Type *C-c / r*. Choose a remailer; use `TAB` to get completion on its name. The buffer will be rewritten for anonymous mailing through that remailer.

Note that you can only select a single remailer when you rewrite the message. To send the message through multiple hops, either rewrite the message multiple times, or define a *chain* of remailers that can be referred to like a single remailer. Remailer chains are described in the next section.

3.4 Remailer Chains

`mc-write-mode` binds the function `mc-remail` to the key *C-c / r*. Depending upon the current remailer scheme, this function may pass the message directly to a remailer client, or may simply rewrite the message for a remailer or chain. For “type1” remailers, the resulting buffer is just a new Email message, so it can itself be rewritten for another remailer; this is one way to manually construct a remailer chain.

Mailcrypt also has powerful facilities for defining automatic chains. We will start with an example. Suppose you have put the following into your `.emacs` file:

```
(setq mc-remailer-user-chains
      '(("Foo" "alumni" "robo")
        ("Bar" (shuffle-vector ["replay" "flame" "spook"]))
        ("Baz" "Foo" "Bar" "rahul" "Bar")
        ("Quux" 4)))
```

This code defines four chains. The first is named `"Foo"` and consists of `"alumni"` and `"robo"`, in that order. The second is named `"Bar"` and consists of `"replay"`, `"flame"`, and `"spook"` in some random order (a different order will be chosen each time the chain is used). The third is named `"Baz"` and consists of 9 remailers: The two from `"Foo"`, followed by a permutation of the three from `"Bar"`, followed by `"rahul"`, followed by another permutation of the three from `"Bar"`. Finally, the fourth is named `"Quux"` and consists of a random permutation of the four best remailers as ordered in the `~/.remailers` file.

Now whenever you are prompted for a "remailer or chain", the chains "Foo", "Bar", "Baz", and "Quux" will be available, including `TAB` completion on their names. By capitalizing their names, you guarantee they will show up near the top of the completion list if you type `TAB` on an empty input.

Now for the gritty details. `mc-remailer-user-chains` is a list of chain definitions. A chain definition is a list whose first element is the name (a string) and whose remaining elements form a *remailer list*. Each element of a remailer list is one of the following:

1. A raw remailer structure. This is the base case, but you will probably never want nor need to deal with these directly.
2. A string naming another remailer chain to be spliced in at this point.
3. A positive integer N representing a chain to be spliced in at this point and consisting of a random permutation of the top N remailers as ordered in the `~/.remailers` file.
4. An arbitrary Emacs Lisp form, which should return another remailer list which will be spliced in at this point and recursively evaluated. Mmmm, Lisp.

So, in the example "Bar" above, `shuffle-vector` is actually a Lisp primitive which returns a random permutation of the argument vector. (Which brings up a side note: A remailer list can be a vector instead of a list if you like.)

So where do the definitions for "replay" etc. come from?

There is another variable, `mc-remailer-internal-chains`, which has the same format as `mc-remailer-user-chains`. In fact, the concatenation of the two is always used internally when resolving chains by name. The "internal chains" are normally generated automatically from a Levien-format remailer list, which lives in `~/.remailers` by default and is parsed at startup time. The parser creates several chains, each containing a single remailer, and names each chain after the respective remailer.

Thus "replay" (for example) is actually the name of a *chain* whose single element is the remailer at <remailer@replay.com>. So "replay" is a valid name of a chain to include in the definition of another chain, as was done above in the definition of "Bar".

3.5 Response Blocks

Mailcrypt can generate a response block for you. Just type *`C-c / b`* in an outgoing mail buffer. That will prompt you for a chain to use, and will insert the response block at point. Note that you can use any chain you want for your response block; it need not be related to the chain you (later) use to remail the message.

If instead you type *`C-u C-c / b`*, you will be dropped into a recursive edit of the innermost part of the response block. This text is what you will see at the top of the message when the response block is used. This text is the only way to identify the response block, since it will be used to mail you through anonymous remailers.

You probably won't need to use the *`C-u`* feature, since by default the response block contains the date, '`To`' field, and '`From`' field of the message you are composing. However, if you want your response block to point to a USENET newsgroup instead of your Email address, you may edit the innermost part of the response block to have a '`Newsgroups`' line instead of a '`To`' line.

Inserting a response block also updates the '`Reply-to`' hashmark header field. So, when your recipient replies to your message, the reply will automatically be addressed properly.

This only works if the last remailer in the chain used to encrypt the *message* supports hashmarks (the response block chain doesn't matter). If the last remailer does not support hashmarks, Mailcrypt will generate an error when you try to use the chain.

Note that you should insert your response block before you encrypt the message for remailing. Also, see Section 3.10 [Remailer Security], page 14.

3.6 Pseudonyms

Mailcrypt supports pseudonyms. Type *`C-c / p`* in an outgoing message buffer and you will be prompted for a pseudonym to use. Your pseudonym will show up in the '`From`' line that the recipient sees. Your pseudonym may either be a complete '`From`' line (including an Email address), or just a full name (with no Email address). In the latter case, the Email address will automatically be set to <x@x.x>, an invalid address designed to prevent sendmail from going rewrite-happy.

If you have one or more pseudonyms which you normally use, and you aren't afraid of revealing them if your account is compromised, you can set up a default list of pseudonyms with lines like the following in your `.emacs` file:

```
(setq mc-remailer-pseudonyms
      '("Elvis Presley" "Vanna White" "Charles Manson"))
```

Then those names will be available for completion when you are prompted for your pseudonym.

You should insert your pseudonym before you insert a response block, so that the response block will contain the '`From`' line as well as the '`To`' line. That way you can tell who you were pretending to be when you get a reply to your message.

Note: Many remailers do not support pseudonyms. In addition, the Levien format does not (yet) indicate which do and which do not, so Mailcrypt can't warn you when your pseudonym isn't going to work. The only way to be sure is to send yourself a test message, and to try different remailers until you find one or more which work. On the bright side, only the last remailer in the chain needs to provide such support; none of the others matter.

3.7 Remailing Posts

Mailcrypt knows how to rewrite USENET posts for anonymous or pseudonymous remailing. Just compose your post or followup normally, and use *`C-c / r`* to rewrite it for a remailer chain. You don't even need to start your newsreader to make a post; you can just compose a message in mail mode and replace the '`To`' line with a '`Newsgroups`' line before doing *`C-c / r`*.

Mailcrypt will generate an error if the last remailer in the chain does not have both the `post` and `hash` (hashmarks) properties. The hashmarks are used to preserve '`References`' and similar headers, so your anonymous or pseudonymous followups will thread properly. The variable `mc-remailer-preserved-headers` controls which headers are preserved when rewriting a message, but you should not need to change it since the default value is reasonable.

Before rewriting, you can use *`C-c / p`* to insert your pseudonym, and *`C-c / b`* to insert your response block, just like when composing mail. In this case, the response block will include the ‘`From`’ line and the ‘`Newsgroups`’ line (which is the news analogue to the ‘`To`’ line).

3.8 Mixmaster Support

(note: this chapter describes mailcrypt’s support for the “old” mixmaster interface. For a description of mailcrypt’s support for the modern mixmaster client see Section 3.9 [Mixmaster and Mixminion], page 13.)

Mixmaster is a newer type of remailer which provides excellent security against traffic analysis and replay attacks. (For more information on these attacks and Mixmaster, see Lance Cottrell’s FAQ at `http://mixmaster.sourceforge.net/faq.shtml`.

If you do not use Mixmaster, you may skip this section entirely; Mailcrypt’s default configuration treats Mixmaster as if it did not exist.

If you have the Mixmaster executable installed, you can tell Mailcrypt to use it by placing lines like the following into your `.emacs` file:

```
(setq mc-mixmaster-path "mixmaster")
(setq mc-mixmaster-list-path "/foo/bar/baz/type2.list")
```

`mc-mixmaster-path` is a string representing the Mixmaster executable. `mc-mixmaster-list-path` is the complete path to the `type2.list` file.

Once these variables are defined, (and if the remailer scheme is set `mc-remailer-scheme-type1`, see Section 3.2 [Types of Remailers], page 7), Mailcrypt will automatically try to use the Mixmaster executable whenever possible. Specifically, when you rewrite a message for a chain, Mailcrypt will find maximal length sub-chains which have the `mix` property and will use the Mixmaster executable to rewrite for those sub-chains.

This allows arbitrary intermingling of Mixmaster and normal (also called *Type 1*) remailers, but you should note that this is *not recommended.* The recommended procedure is to have a single Mixmaster sub-chain which is most or all of the whole chain.

There are advantages and disadvantages to having the Mixmaster sub-chain at the end of the whole chain. The primary advantage is that Mixmaster remailers support multiple recipients. The primary disadvantages are that they do not support pseudonyms nor posting.

So here, as always, it is the last element of the chain which needs to support the special features you want. In general, the remaining elements do not matter, and the superior security of Mixmaster remailers is a good argument for using them for the bulk of your chains.

Mixmaster remailers also have a "Type 1 compatibility mode" which you might want to invoke to use a pseudonym or make a post. You can do this with the function `mc-demix`. Here is an example of its use:

```
(setq mc-remailer-user-chains
      '(("Foo" "vishnu" "spook")
        ("Bar" "Foo" (mc-demix "replay"))))
```

This makes "Bar" a chain of three remailers, and guarantees that the last one ("replay") will be used in compatibility mode.

Note that Mixmaster remailers cannot be used for response blocks. Mailcrypt will ignore the `mix` property when generating a response block.

3.9 Mixmaster and Mixminion

Mailcrypt now contains preliminary support for the modern mixmaster client, as well as the experimental mixminion client. This support is accessed through the `mc-remailer-scheme-mixmaster` and `mc-remailer-scheme-mixminion` scheme settings.

Both of these clients operate in a mode where the completely absorb the message body. When run, they accept the text on stdin, decide upon a remailer chain, construct the message packets, then transmit the packets directly. Unlike the type1 (and older mixmaster) interface, once these client programs finish, the message has been sent and the user's MUA buffer is no longer needed. Note, in particular, that this bypasses the MUA's "send" command.

To make sure that the user does not accidentally use the MUA mode to send the message (non-anonymously), Mailcrypt will attempt to erase the message buffer and To: header. Some MUA modes may also have support code which will delete the message buffer altogether.

Mailcrypt will also ask "Do you really want to send this message" before running the client program. This is your last chance to avoid sending the message.

The modern client programs are designed to convey message *bodies*, not the headers-plus-body that usually make up an email message. Email headers, even with the obvious ones like From: and Message-Id: stripped out, are too likely to leak information, making it possible for the message recipient to figure out who sent the message. Any extra information about the kind of system used to construct the message serves to reduce the "anonymity set", the set of possible message senders. A smaller anonymity set means less anonymity.

The client programs have command-line arguments to add certain headers (like Subject:) back into the message. Different clients (and different versions of the same client) offer different options. Where possible, Mailcrypt will take the headers from your message and pass them to the remailer client program. Note that most clients only allow a single recipient to be named.

Mailcrypt support for modern remailer clients is still preliminary. It does not yet provide control over several options offered by the clients. Features not useable include:

Mixmaster

Newsgroup-posting

Nym support

Automatic sign/encrypt

reply-chain generation

file attachments

Mixminion

reply block generation

In addition, both clients offer ways to influence the chain of remailers to use for the message. Mailcrypt does not currently offer a way to access this control: messages will be sent using the default client settings (which usually means a chain of four reliable hops).

3.10 Remailer Security

Keep in mind that there is only one person fully qualified to protect your privacy: *you*. You are responsible for obtaining a list of remailers and their public keys; you are responsible for choosing which of them to use and in what order. There are public lists of remailers and keys (the Quick Start section above relies on them), but you pay for the convenience by putting your trust in a single source. This is one reason Mailcrypt does not access these public lists automatically; you need to get into the habit of watching what goes on behind the scenes. You should also try to learn something about the remailers themselves, since you are relying on them to help protect your privacy.

How many remailers should you include in your chain, and how should you choose them? That depends on whom you perceive as a threat. If the threat is your ex-spouse or your boss, even a single remailer is probably adequate (more won't hurt, but will cost in latency). If the threat is the Church of Scientology, you probably want to use a fair number of remailers across multiple continents. If the threat is a major world government, well, best of luck to you.

Also, there is a huge difference between chains suitable for regular messages and chains suitable for response blocks. Some remailers don't even keep mail logs (at least, their operators claim they do not), so it may be literally impossible to trace a message back to you after the fact if you chain it through enough remailers. Response blocks, on the other hand, have your identity buried in there *somewhere*. In principle, at least, it is possible to compromise the keys of all the remailers in the chain and decrypt the response block. So you should either use very long and strong chains for your response blocks, avoid using response blocks at all, or only use response blocks which themselves ultimately point to a newsgroup.

3.11 Verifiable Pseudonyms

Here is a plausible sequence of operations when using the remailer support in Mailcrypt:

1. You create a public/private PGP key pair. You give it a User ID which is your pseudonym. You upload the public key to the key servers or otherwise distribute it. (Be aware that anyone who compromises your account can read the IDs on your secret keyring, thus discovering your verifiable pseudonyms.)
2. You compose an Email message, Email reply, news post, or news followup.
3. You insert your pseudonym with `C-c / p`.
4. (Optional) You insert your response block with `C-c / b`.
5. You type `C-c / s` to sign the message. The `mc-sign` function understands pseudonyms.
6. You type `C-c / r` to rewrite the message for remailing. (Or use `C-u C-c / r` to view each step of the rewriting as it happens.)
7. You type `C-c C-c` to send the message.

Now the recipient(s), reading your message through mail or news, can verify your pseudonymous signature; thus you have started to create a verifiable pseudonymous identity. If you use it consistently, it will develop a reputation of its own. With Mailcrypt, using a pseudonym is almost as easy as using your real name (and your followups in news will even thread properly). Welcome to the new age of letters. . .

3.12 Remailer Tips

This is a collection of tips for using Mailcrypt's remailer support.

- Read and understand the `.remailers` file. Do a web search for "anonymous remailer list", ask around in `news:alt.privacy.anon-server`, or, as a last resort, `news:alt.security.pgp`. Check the documentation (*C-h v*) for the variable `mc-levien-file-name` for a description of Levien format.
- Mailcrypt needs to be able to encrypt a message to each remailer in the chain, so it needs access to their public keys, in a keyring usable by the currently selected backend. Keyrings containing keys for all the well-known remailers are usually available from the same places as the remailer lists.
- The relevant remailer properties are `pgp` (required), `hash` (required if you use hashmark headers), and `post` (required for posting to USENET). Remailers which do not support PGP won't even show up in the completion list.
- The only remailer which needs special properties (e.g., posting, hashmarks, pseudonym support) is the last one in a chain. Any remailer can be used at the beginning or in the middle. So if you find a few remailers which support the feature(s) you require, and you always use them at the end of your chains, then you can be confident that even the longest chains will work.
- If you update your `~/.remailers` file, you can reread it with *M-x mc-reread-levien-file*.
- Remember the natural order of operations. First you compose your message. Then you insert your pseudonym with *C-c / p*. Then you insert your response block with *C-c / b*. Then you sign (*C-c / s*) or sign and encrypt (*C-c / e*) the message. Then you rewrite it for a remailer or chain (*C-c / r*). Then you send it. All but the first and last two of these are optional. (Well, strictly speaking, they are all optional, but you get the idea.)
- Find and read some of the excellent remailer documentation available on the Internet. For some good starting points, see Chapter 9 [References], page 21.

4 Passphrase Cache

Mailcrypt can remember your passphrase so that you need not type it repeatedly. It will also "forget" your passphrase if it has not been used in a while, thus trading some security for some convenience. You can tune this tradeoff with the variable `mc-passwd-timeout`, which is a duration in seconds from the last time the passphrase was used until Mailcrypt will forget it. The default value is 60 seconds.

So, for example, to make Mailcrypt remember your passphrase for 10 minutes after each use, you would use the following line in your `.emacs` file:

```
(setq mc-passwd-timeout 600)
```

A value of `nil` or 0 will disable passphrase caching completely. This provides some increase in security, but be aware that you are already playing a dangerous game by typing your passphrase at a Lisp interpreter.

Mailcrypt understands multiple secret keys with distinct passphrases.

To manually force Mailcrypt to forget your passphrase(s), use the function `mc-deactivate-passwd`. Both `mc-read-mode` and `mc-write-mode` bind this function to *`C-c / f`* by default.

> **Warning:** Although Mailcrypt takes pains to overwrite your passphrase when "forgetting", it cannot prevent the Emacs garbage collector from possibly leaving copies elsewhere in memory. Also, your last 100 keystrokes can always be viewed with the function `view-lossage`, normally bound to *`C-h l`*. So be sure to type at least 100 characters after typing your passphrase if you plan to leave your terminal unattended.

5 Key Fetching

Mailcrypt knows how to fetch PGP public keys from the key servers (see Section 9.2 [Key Servers], page 22). The function `mc-fetch-key` is bound by default to *`C-c / k`* in both `mc-read-mode` and `mc-write-mode`. Additionally, `mc-encrypt`, `mc-decrypt`, and `mc-verify` will offer to call this function to automatically fetch a desired key. If you call it manually, it will prompt you for the User ID of the key to fetch.

The variable `mc-pgp-fetch-methods` is a list of ways to attempt to fetch a key. (More precisely, it is a list of functions to be called, each of which will attempt to fetch the key.) The methods will be tried in the order listed. The default list is:

```
'(mc-pgp-fetch-from-keyrings
  mc-pgp-fetch-from-finger
  mc-pgp-fetch-from-http)
```

For a description of these functions, see the following sections.

If you are not directly on the Internet, you probably want to obtain a copy of the global public key ring from the keyservers, install it somewhere under the name `public-keys.pgp`, and do:

```
(setq mc-pgp-fetch-methods '(mc-pgp-fetch-from-keyrings))
(setq mc-pgp-fetch-keyring-list '("/blah/blah/blah/public-keys.pgp"))
```

This will allow you to fetch keys from your local copy of the global key ring instead of sending requests to the key servers directly (see Section 5.1 [Keyring Fetch], page 17). Alternately, if your organization has a proxy HTTP server, you can configure Mailcrypt to use that. See Section 5.3 [HTTP Fetch], page 17.

If the key is found, you will be shown the result of running PGP on it locally. This allows you to inspect the signatures on the key *relative to your own keyring* before you consent to having it added. **Inspect the signatures carefully!** Key distribution is often the Achilles' heel of public key protocols. If you blindly use keys obtained from the key servers, you are asking for trouble.

All of the methods use `mc-pgp-fetch-timeout` as a timeout in seconds; the default value is 30.

5.1 Keyring Fetch

The function `mc-pgp-fetch-from-keyrings` will attempt to fetch a key from a set of keyrings on the locally accessible filesystem. This is useful if your organization maintains a large common public keyring whose entire contents you do not wish to duplicate on your own ring. It is also useful if you download a copy of the global public ring from the key servers (see Section 9.2 [Key Servers], page 22).

The variable `mc-pgp-fetch-keyring-list` controls this behavior. It is a list of file names of public keyrings which this function will search, in order, when seeking a key. The default value is `nil`, meaning this search will always fail.

5.2 Finger Fetch

The function `mc-pgp-fetch-from-finger` will attempt to fetch a key by fingering an address and parsing the output for a PGP public key block.

5.3 HTTP Fetch

The function `mc-pgp-fetch-from-http` will attempt to fetch a key by connecting to a key server (see Section 9.2 [Key Servers], page 22) which has a World Wide Web interface.

The variables `mc-pgp-keyserver-address`, `mc-pgp-keyserver-port`, and `mc-pgp-keyserver-url-template` control the fetching process. The default is to use Brian LaMacchia's key server at MIT. If this default should stop working, or if you want to help with network congestion and machine load, you can choose a different server. As of this writing, any of the following sequences of Emacs Lisp in your `.emacs` file will work; choose one:

```
;; Key server at MIT (Massachusetts, USA)
;; This is the default; these lines are only for reference
;(setq mc-pgp-keyserver-address "pgp.ai.mit.edu")
;(setq mc-pgp-keyserver-port 80)
;(setq mc-pgp-keyserver-url-template
;      "/htbin/pks-extract-key.pl?op=get&search=%s")

;; Key server at UPC (Barcelona, Spain)
(setq mc-pgp-keyserver-address "goliat.upc.es")
(setq mc-pgp-keyserver-port 80)
(setq mc-pgp-keyserver-url-template
      "/cgi-bin/pks-extract-key.pl?op=get&search=%s")

;; Key server at Cambridge University (Cambridge, England)
(setq mc-pgp-keyserver-address "www.cl.cam.ac.uk")
(setq mc-pgp-keyserver-port 80)
(setq mc-pgp-keyserver-url-template
      "/cgi-bin/pks-extract-key.pl?op=get&search=%s")

;; Key server at UIT (Tromso, Norway)
(setq mc-pgp-keyserver-address "www.service.uit.no")
(setq mc-pgp-keyserver-port 80)
(setq mc-pgp-keyserver-url-template
      "/cgi-bin/pks-extract-key.pl?op=get&search=%s")
;; Key server at CMU (Pennsylvania, USA)
(setq mc-pgp-keyserver-address "gs211.sp.cs.cmu.edu")
(setq mc-pgp-keyserver-port 80)
(setq mc-pgp-keyserver-url-template "/cgi-bin/pgp-key?pgpid=%s")
```

If your organization has a firewall, you might not be able to access the World Wide Web directly. Your organization may have a proxy HTTP server set up, however. In that case, you should place code like the following in your `.emacs` file. You can use any of the above key servers instead of the one at MIT, of course.

```
     ;; Mailcrypt configuration for accessing key server through HTTP proxy
     (setq mc-pgp-keyserver-address "your.proxy.com")
     (setq mc-pgp-keyserver-port 13013)  ; Your proxy's port
     (setq mc-pgp-keyserver-url-template
            "http://pgp.ai.mit.edu/htbin/pks-extract-key.pl?op=get&search=%s")
```

Note that fetching from a key server can be somewhat slow, so be patient. (At least it beats the tar out of the Email interface.)

5.4 GnuPG Fetch

GnuPG happens to have a built-in HKP keyserver interface which is completely independent from MailCrypt's own key fetching support. If your `.gnupg/gpg.conf` (`.gnupg/options` for older versions) file includes a line like:

'`keyserver wwwkeys.pgp.net`'

then any operation that needs an otherwise-unavailable public key (which generally means signature verification) will automatically contact the keyserver and try to retrieve the key. It sends the hex keyid to the server, not a string, so it could only be used at encryption time if you already know the keyid of your recipients.

You can also tell GPG to explicitly request a key (by hex keyid) with '`--recv-keys`', or to send your own key with '`--send-keys`'. Check the GnuPG manual for details.

It is also possible to fetch keys with `mc-fetch-key`, although its behaviour is a bit different from the one described in the pgp section, if `mc-default-scheme` is set to `'mc-scheme-gpg`. When called interactively, it will prompt for a key ID to fetch from a keyserver. You can either set the server to query with

```
     ;; Key server at DFN (Germany)
     ;; You should choose another one in your region.
     (setq mc-gpg-keyserver "blackhole.pca.dfn.de")
```

in your `.emacs` file or let GPG use its default defined in its configuration file. Every string that can be passed to the gpg '`--keyserver`' option is allowed for `mc-gpg-keyserver`. At the moment it is *not* possible to pass a search string to the function. Please use the '`--search-key`' command option if you have a newer version of gpg. Maybe someday we will write a frontend for this.

If you want to finger a key from a server use the `mc-gpg-fetch-from-finger` function. It expects an input of the form '`USER@HOST`'. The variable `mc-gpg-finger-timeout` defines the timeout in seconds for the operation.

6 Miscellaneous Configuration

This chapter documents some additional Mailcrypt configuration options which could not be naturally described elsewhere.

6.1 Alternate Keyring

By default, Mailcrypt will use the same public keyring that PGP would use if executed from the shell.

You can cause Mailcrypt to use a specific public keyring by setting the variable `mc-pgp-alternate-keyring`. If this variable is set, Mailcrypt will use that keyring for all functions which would otherwise have used the default. This includes adding keys, extracting keys, verifying signatures, and encrypting messages.

This feature might be useful if you maintain multiple keyrings; you can switch between them by setting this variable. Depending on your tastes, you might want to configure fetching from a keyring as well (see Section 5.1 [Keyring Fetch], page 17).

6.2 Comment Field

By default, Mailcrypt will supply a "comment" option to PGP, resulting in output which looks something like this:

```
----- BEGIN PGP FOOBAR -----
Version: 2.6.3
Comment: Processed by Mailcrypt 3.5.9, an Emacs/PGP interface

...
----- END PGP FOOBAR -----
```

To change the comment to one of your own, set the variable `mc-pgp-comment`. Set it to `nil` to use PGP's default, which is probably either no comment or something defined in `config.txt`. `mc-pgp50-comment` and `mc-gpg-comment` are the corresponding variables for the other versions.

6.3 Mode Line

`mc-read-mode` and `mc-write-mode` will each indicate they are active by placing the string ‘`MC-r`’ or ‘`MC-w`’ in the mode line, respectively.

You can change these strings by setting the variables `mc-read-mode-string` and `mc-write-mode-string`. So, for example, to get rid of the mode indicators entirely, you might put the following lines into your `.emacs` file:

```
(setq mc-read-mode-string "")
(setq mc-write-mode-string "")
```

6.4 Key Bindings

The Mailcrypt key bindings are defined by the keymaps `mc-read-mode-map` and `mc-write-mode-map`. To change the key bindings, you just need to set these variables in your `.emacs` file.

For example, if you wanted *C-c C-m* to be the Mailcrypt prefix (instead of *C-c /*) in `mc-read-mode`, you would put the following code in your `.emacs` file:

```
(setq mc-read-mode-map (make-sparse-keymap))
(define-key mc-read-mode-map "\C-c\C-mf" 'mc-deactivate-passwd)
(define-key mc-read-mode-map "\C-c\C-md" 'mc-decrypt)
(define-key mc-read-mode-map "\C-c\C-mv" 'mc-verify)
(define-key mc-read-mode-map "\C-c\C-ma" 'mc-snarf)
(define-key mc-read-mode-map "\C-c\C-mk" 'mc-fetch-key)
```

For more information on Emacs key bindings, see Section “Customizing Key Bindings” in *The GNU Emacs Manual*.

6.5 Nonstandard Paths

The information in this section should be unnecessary, but is provided "just in case".

Mailcrypt will look for the PGP executable in your standard search path under the name `pgp`. To use a different name (or to provide a complete path), set the variable `mc-pgp-path`.

PGP 5.0 includes four separate executables, usually installed as "pgpe", "pgps", "pgpv", and "pgpk". The variables `mc-pgp50-pgpe-path`, `mc-pgp50-pgps-path`, `mc-pgp50-pgpv-path`, and `mc-pgp50-pgpk-path` tell Mailcrypt where to find them if they are not on your search path.

GnuPG is normally installed as "gpg". `mc-gpg-path` tells Mailcrypt where to find the executable if it is not on your path.

In order to keep your identities straight, Mailcrypt needs to know where your secret keyring resides.

Mailcrypt figures this out heuristically by assuming that the file `secring.pgp` is in the same directory as your public key ring. It determines the location of the latter by doing a dry run of PGP with ‘`+verbose=1`’ and parsing the output.

If this heuristic is failing for you, you can manually tell Mailcrypt where your secret key ring is by setting the variable `mc-pgp-keydir`, like this:

```
(setq mc-pgp-keydir "/users/patl/.pgp/")
```

Note that the trailing slash is *required*.

If the heuristic fails, please report it as a bug (see Chapter 10 [Credits], page 24).

Note that if you have changed the default location of your secret keyring, Mailcrypt will be unable to locate it. You can work around this by either setting `mc-pgp-keydir`, or by making a symbolic link to your secret keyring from `secring.pgp` in your default public keyring directory.

7 Tips

Here are some random tips.

- PGP provides quite good security when used correctly. You are far more likely to use it correctly if you have read the directions. Read the *PGP User's Guide*!

- 60 seconds is a relatively safe but somewhat inconvenient value for `mc-passwd-timeout`. If your paranoia permits, consider increasing it to five or ten minutes (see Chapter 4 [Passphrase Cache], page 15).
- If Mailcrypt ever does something you wish it had not, *DON'T PANIC*. Just use the normal Emacs undo command, `M-x undo` or `C-x u`, to restore your buffer (see Section "Undoing Changes" in *The GNU Emacs Manual*). Mailcrypt keeps almost no state except what you see in your buffer, so any action can be undone this way.
- All Mailcrypt operations place PGP's output in the `*MailCrypt*` buffer. Check it occasionally for status and warning messages.
- Add yourself to the Mailcrypt announcements mailing list (see Section 9.3 [Mailing List], page 23). That way you can find out about new versions of Mailcrypt automatically, and we can enjoy the feeling that people are actually using our package.

8 Limitations

Mailcrypt is a powerful program, but it is not a complete PGP interface. Perhaps some future version will be; in the meantime, you will need to use the command-line interface for some operations. Things which the current version does not support include:

Complete Key Management

Mailcrypt's key management support is limited to adding and extracting keys from keyrings. It does not support key generation, key removal, key revocation, ID and trust parameter editing, or key signing. It also ignores PGP's warnings when you use a key which is not fully certified. (Of course, you can see these warnings by viewing the `*MailCrypt*` buffer; see Chapter 7 [Tips], page 20.)

Encryption with Conventional Cryptography

Mailcrypt supports decryption but not encryption with "conventional" (i.e., non-public key) cryptography.

Detached Signatures

Mailcrypt does not support the creation nor the verification of detached signatures.

"*For your eyes only*" *Decryption*

Mailcrypt will be unable to decrypt a file which was encrypted with the "for your eyes only" ('`-m`') option. This is actually a bug in PGP, which provides no portable way to avoid its paging behavior.

9 References

This chapter contains information and pointers to information about topics related to PGP and Mailcrypt.

9.1 Online Resources

`http://sourceforge.net/users/patl`
"Mailcrypt: An Emacs/PGP Interface", by Patrick J. LoPresti. The author of original Mailcrypt.

`http://world.std.com/~franl/crypto.html`
"Cryptography Web Sites, Publications, FAQs, and References", by Fran Litterio. This page is simply excellent. It makes all the other References in this chapter redundant, but we will include them anyway for redundancy.

`http://www.faqs.org/faqs/by-newsgroup/alt/alt.security.pgp.html`
This is a site for the `alt.security.pgp` FAQ lists.

`news:alt.security.pgp`
The `alt.security.pgp` newsgroup is a good place to go for discussion about PGP, as well as any topic which any fool anywhere ever thinks is related to PGP. It is also a good last resort for getting answers to questions, but please read the FAQ lists first.

`http://www.farcaster.com/`
Brian LaMacchia put together a World Wide Web interface to the public key servers (see Section 9.2 [Key Servers], page 22). Mailcrypt uses this interface by default when attempting to fetch keys via HTTP (see Section 5.3 [HTTP Fetch], page 17); most people get to his interface through this page.

`ftp://ftp.csua.berkeley.edu/pub/cypherpunks/Home.html`
The Cypherpunks are dedicated to taking proactive measures to ensure privacy in the digital age. They wrote the software for, and operate many of, the anonymous remailers currently in existence.

`http://www.advogato.org/person/raph/`
Raph Levien (raph.levien at gmail.com) previously maintained a remailer list. If you are impressed by how easy it is to configure Mailcrypt's remailer functions, Raph is the one to thank.

`http://en.wikipedia.org/wiki/Lance_Cottrell`
Lance Cottrell is the author of Mixmaster.

`http://www.gnupg.org/`
Homepage for the GNU Privacy Guard. This is a GPL-ed replacement for PGP.

9.2 Key Servers

Key servers are machines with a publicly accessible interface to an enormous global public keyring. Anyone may add keys to or query this keyring. Each key server holds a complete copy of the global keyring, and they arrange to keep one another informed of additions they receive.

This means you can tell any key server to add your public key to the global keyring, and all of the other servers will know about it within a day or so. Then anyone will be able to query any key server to obtain your public key.

To add your key to the keyservers, send an Email message to `pgp-public-keys@pgp.ai.mit.edu` with a subject line of '`ADD`' and a body containing your public key block. With Mailcrypt installed, you can just type *`C-c / x`* to insert your public key block (see Section 2.3 [Inserting Keys], page 5) into the body of the message.

For help with the Email interface to the key servers, send a message with a subject line of '`HELP`'. For a World Wide Web interface to the key servers, see Brian LaMacchia's home page at `http://www-swiss.ai.mit.edu/~bal/`.

Some other key servers include:

- pgp-public-keys@jpunix.com
- pgp-public-keys@kub.nl
- pgp-public-keys@uit.no
- pgp-public-keys@pgp.ox.ac.uk

For a complete list, consult any good online repository of PGP information (see Section 9.1 [Online Resources], page 22).

It is strongly recommended that you submit your key to the key servers, since many humans and programs (including Mailcrypt) may look for it there. Besides, it takes mere seconds and the pain passes quickly.

9.3 Mailing List

New releases of Mailcrypt are announced on the SourceForge mailing lists. They are where discussion about bugs and new features take place.

Please see `http://mailcrypt.sourceforge.net/` for subscription instructions and archives.

9.4 Politics

Cryptography in general, PGP in particular, and free software are politically somewhat controversial topics. Heck, in the U.S. Congress, freedom of speech is a controversial topic. Anyway, here are some organizations you should definitely watch and preferably send lots of money.

The Electronic Frontier Foundation

The EFF (`http://www.eff.org/`) works to protect civil liberties in cyberspace. They also maintain an impressive collection of on-line resources. If you like Mailcrypt so much that you wish you had paid for it, this is the number one place we would want to see your money go. The EFF newsgroups, `comp.org.eff.news` and `comp.org.eff.talk`, are required reading for the well-informed.

The League for Programming Freedom

The LPF (`http://www.lpf.org/`) works to fight software patents, which threaten to make free software like Mailcrypt impossible.

The Center for Democracy and Technology

The CDT (`http://www.cdt.org/`) has essentially the same goals as the EFF, but is more of a lobbying group.

Mailcrypt's remailer support was inspired by the Communications Decency Act of 1995 (see `http://www.cdt.org/speech/cda/` and `http://wikipedia.org/wiki/Communications_Decency_Act`) and by the International "Church" of Scientology (see `http://wikipedia.org/wiki/Church_of_Scientology`).

10 Credits

Mailcrypt was written by Jin Choi (jin@atype.com) and Pat LoPresti (patl@lcs.mit.edu). PGP 5 support was added by Len Budney. GnuPG, modern mixmaster, and mixminion support were added by Brian Warner.

Mailcrypt is hosted on SourceForge, at `http://mailcrypt.sourceforge.net/`. Please send us your bug reports and comments. Also see Section 9.3 [Mailing List], page 23.

This documentation was mostly written by Pat LoPresti, but borrows heavily from an earlier version by Hal Abelson (hal@mit.edu).

Mailcrypt would not be as robust nor as featureful if it were not for our outstanding set of Beta testers:

- Samuel Tardieu <sam@inf.enst.fr>
- Richard Stanton <stanton@haas.berkeley.edu>
- Peter Arius <arius@immd2.informatik.uni-erlangen.de>
- Tomaz Borstnar <tomaz@cmir.arnes.si>
- Barry Brumitt <belboz@frc2.frc.ri.cmu.edu>
- Steffen Zahn <Steffen.Zahn%robinie@sunserv.sie.siemens.co.at>
- Mike Campbell <mcampbel@offenbach.sbi.com>
- Mark Baushke <mdb@cisco.com>
- Mike Long <mike.long@analog.com>

Index

This index has an entry for every key sequence, function, and variable documented in this manual.

C

`C-c / a` 6
`C-c / b` 10
`C-c / d` 5
`C-c / e` 4
`C-c / f` 15
`C-c / k` 16
`C-c / p` 11
`C-c / r` 9
`C-c / s` 4
`C-c / v` 6
`C-c / x` 5

M

`mc-always-replace` 5
`mc-deactivate-passwd` 15
`mc-decrypt` 5
`mc-default-remailer-scheme` 8
`mc-demix` 12
`mc-encrypt` 4
`mc-encrypt-for-me` 4
`mc-fetch-key` 16
`mc-gpg-comment` 19
`mc-gpg-path` 20
`mc-insert-public-key` 5
`mc-install-read-mode` 3
`mc-install-write-mode` 3
`mc-levien-file-name` 15
`mc-mixmaster-list-path` 12
`mc-mixmaster-path` 12
`mc-passwd-timeout` 15
`mc-pgp-alternate-keyring` 19
`mc-pgp-always-sign` 4
`mc-pgp-comment` 19
`mc-pgp-fetch-from-finger` 17
`mc-pgp-fetch-from-http` 17
`mc-pgp-fetch-from-keyrings` 17
`mc-pgp-fetch-keyring-list` 17
`mc-pgp-fetch-methods` 16
`mc-pgp-keydir` 20
`mc-pgp-keyserver-address` 17
`mc-pgp-keyserver-port` 17
`mc-pgp-keyserver-url-template` 17
`mc-pgp-path` 20
`mc-pgp-user-id` 5
`mc-pgp50-comment` 19
`mc-pgp50-pgpe-path` 20
`mc-pgp50-pgpk-path` 20
`mc-pgp50-pgps-path` 20
`mc-pgp50-pgpv-path` 20
`mc-post-encryption-hook` 4
`mc-post-signature-hook` 4
`mc-pre-encryption-hook` 4
`mc-pre-signature-hook` 4
`mc-read-mode` 3
`mc-read-mode-map` 19
`mc-read-mode-string` 19
`mc-remail` 9
`mc-remailer-encrypt-for-chain` 9
`mc-remailer-internal-chains` 10
`mc-remailer-preserved-headers` 11
`mc-remailer-pseudonyms` 11
`mc-remailer-user-chains` 9
`mc-reread-levien-file` 15
`mc-setversion` 3
`mc-sign` 4
`mc-snarf` 6
`mc-verify` 6
`mc-write-mode` 3
`mc-write-mode-map` 19
`mc-write-mode-string` 19

Table of Contents

1 Introduction 1

- 1.1 Prerequisites 1
- 1.2 Installation 1
  - 1.2.1 Hooking into Rmail 2
  - 1.2.2 Hooking into VM 2
  - 1.2.3 Hooking into MH-E 2
  - 1.2.4 Hooking into Gnus 2
  - 1.2.5 Hooking into Mew 2
- 1.3 Command Overview 2

2 General Use 3

- 2.1 Encrypting a Message 4
- 2.2 Signing a Message 4
- 2.3 Inserting a Public Key Block 5
- 2.4 Decrypting a message 5
- 2.5 Verifying a Signature 6
- 2.6 Snarfing a Key 6

3 Remailer Support 6

- 3.1 Remailer Introduction 6
- 3.2 Types of Remailers 7
- 3.3 Remailer Quick Start 8
- 3.4 Remailer Chains 9
- 3.5 Response Blocks 10
- 3.6 Pseudonyms 11
- 3.7 Remailing Posts 11
- 3.8 Mixmaster Support 12
- 3.9 Mixmaster and Mixminion 13
- 3.10 Remailer Security 14
- 3.11 Verifiable Pseudonyms 14
- 3.12 Remailer Tips 15

4 Passphrase Cache 15

5 Key Fetching 16

- 5.1 Keyring Fetch 17
- 5.2 Finger Fetch 17
- 5.3 HTTP Fetch 17
- 5.4 GnuPG Fetch 18

6 Miscellaneous Configuration 19
- 6.1 Alternate Keyring 19
- 6.2 Comment Field 19
- 6.3 Mode Line 19
- 6.4 Key Bindings 19
- 6.5 Nonstandard Paths 20

7 Tips 20

8 Limitations 21

9 References 21
- 9.1 Online Resources 22
- 9.2 Key Servers 22
- 9.3 Mailing List 23
- 9.4 Politics 23

10 Credits 24

Index 25