Everything You Need To Know To Start Using GnuPG with Mutt

By Justin R. Miller <justin@solidlinux.com>

Revision 0.1, Released 23 September 2001

I. Purpose

I'm writing this document for a couple of reasons. Firstly, I believe that everyone should be using encryption and signatures in their email.

Secondly, I believe that if you are already using a mail client such as Mutt (which a lot of people are), then there is absolutely no reason for you not to be using PGP-compatible software.

Thirdly, all of the documentation that I've seen thus far is geared toward someone who is already familiar with PGP, so it just explains things like how to configure Mutt to do such tasks as automatically sign outgoing messages or write a procmail recipe to rewrite old-style inline PGP messages as S/MIME messages. This info means little if you don't understand the technical jargon. By no means do I want to trample the existing documentation for Mutt or GnuPG, the Mutt-i, GnuPG and PGP Howto, nor any other existing documentation. I want simply to provide a resource for the use of Mutt with GnuPG exclusively and to summarize the important points of the whole matter so that one can get started quickly and responsibly.

Fourthly, I am trying to promote both Mutt and GnuPG, as free software projects, for use in everyday communications.

Lastly, I enjoy writing tutorials and I hope that this one is of use to many people. I did a lot of research and I figured a number of things out on my own, and now I want to help people along.

II. Overview

When you are finished with this document, you will be able to send and receive cryptographically signed and/or encrypted email with the Mutt mail reader and GnuPG, the GNU software implementation of OpenPGP (RFC2440). If you want to know why you should use encryption, please

read about that topic first (see the links at the end of this document).

Using GnuPG with Mutt is quite easy, but the main things to be learned are the details and the theory behind the actual actions. This document will explain the need for signing and how to sign messages, how to verify signatures on mail sent to you, how to encrypt messages, and how to decrypt messages. It will also show you, within the scope of Mutt, how to collect and manage the public keys of your correspondents and how to manage your own private key. Lastly, it will show you how to publicize your own public key and how to encourage others to use PGP when communicating with you.

==========

III. Assumptions

Before reading this document, you should have a working knowledge of the Mutt mail reader and you should know the basics of public key cryptography. You should understand that a public key is for others to use when encrypting messages to you and when verifying signatures from you, and that a private key is used for creating signatures and for decrypting messages to you.

==========

IV. Configuring GnuPG

Follow the instructions that came with GnuPG in order to create a private and public key. If you are not sure what values to use, pick the default values for key type and key size and use your name and email address as you would normally address your email.

The only other thing that you might want to do is to configure GnuPG to automatically fetch public keys as necessary. This comes in very handy with Mutt, as you won't have to go and download and import a key manually in another terminal. In order to do this, add this line to your ~/.gnupg/options file:

```
keyserver <keyserver address>
```

Some good keyservers are:

```
wwwkeys.pgp.net
search.keyserver.net
pgp.ai.mit.edu
```

Please note that key retrievals do not happen over HTTP and as such, the keyserver address should not start with 'http://'.

You may test this out by retrieving my public key:

```
gpg --recv-keys 0xC9C40C31
```

You may then confirm that the key was added to your keyring:

```
gpg --list-keys justin
```

==========

V. Protecting Your Private Key

The most important security points to remember in all of this are to protect your private keyring file, and above that, to protect your passphrase. There is still a chance that your private key can be cracked without the passphrase, but it is slim in the hands of one person.

You should limit the number of machines on which your private key is copied. You should only transfer your private key via a floppy disk or

other physical media. If you do not trust the environment on which you will be working to be free from the possiblity of snooping (i.e. a work computer where others have access to your personal files), then you should not introduce your private key to this environment. Either that, or you should create a separate key for use as a "work key" and keep your "home key" in a safe environment.

Theft of your private key by anyone means theft of your identity. In an increasingly digital world, your private key will soon carry as much weight as a photo ID, and in some situations already carries more.

==========

VI. Publicizing Your Public Key

People will need your public key both to verify signed messages by you and to send you encrypted messages. You can make it available by putting it on a website, publishing it in print, mailing it via email or postal mail, faxing it, uploading it to a keyserver, or any number of other means.

The more places your public key is put, the better, since there is a lower chance of all of them being modified and distributed falsely. However, just because a key can be obtained from multiple places and all copies may match doesn't mean that the key should be considered valid. A key should only be considered valid (i.e. truly belonging to its owner) if it does so within your web of trust (see next section). To clarify, you can determine whether or not a message originated from a particular key, but you cannot, aside from a validity check, determine whether that key belongs to who it seems to belong to.

You can find my public key in this document, by searching a keyserver for key ID 0xC9C40C31 or my name, or from my website. In addition, Mutt makes it easy to send your public key to someone (once Mutt is configured to work with GnuPG as described below) through the Esc-k key sequence.

You may also add an X-header to your outgoing mail to provide key download information. All of my personal mail includes this header:

```
X-PGP-Key: http://solidlinux.com/~justin/pubkey.asc
```

All of these approaches, as well as signing all outgoing mail, help publicize the fact that you wish to use cryptography for secure email communication. These things will also often generate questions about what the signature attachment means or why you have a file full of garbled characters on your website. This lets you have all the more chances to publicize PGP usage and awareness.

There is considerable debate as whether signing messages that are sent out to mass audiences (via mailing lists and the like) is appropriate. I hope to make available in the near future some arguments for both sides of this debate, but if you'd like more info, you can search the mutt-users and gnupg-users mailing list archives.

==========

VII. Understanding Validity and Your Web of Trust

'Validity' for a particular key refers to the knowledge that the key belongs to the person to whom you expect it to belong. This knowledge comes about based on your trust in the people who have signed the key (including, but not limited to, the key owner).

'Trust' in a person is a property of your particular installation of GnuPG. Trust is a private value that only you have to know about and refers to whether or not you trust the person's signature on a key to be as good as your signature on a key, and the degree to which that trust exists. Initially, key owners have a trust value of 'unknown'. You may

give them a trust value of 'none' if they are known to improperly sign keys. A value of 'marginal' means that they understand key signing and perform it properly. A value of 'full' means that they have an excellent understanding of key signing and that you trust their signature on a key as well as if you had signed the key yourself.

By default, a key is considered valid if it is signed by at least one person to whom you give full trust, or it is signed by at least three people to whom you give marginal trust. This can of course be reconfigured, and a lower number of marginally trusted owners would signify a smaller number of people who would have to conspire against you to pass a key off as valid.

Without the necessary trusted signatures, the key is not considered valid. This does not necessarily mean that the key does not belong to whom you expect it to, but that the software is warning you that it has no way of knowing. Obviously, the web of trust is the weak point in public key cryptography, but when used properly can introduce some level of assurance into the situation.

Allow me to illustrate the importance of key validity with an example. Say that you have met someone on IRC and that she has helped you for a long time with many systems administration issues. She tells you that she is going to send you a script that you should run as root on your machine. If you know this script to come from this person, you would have no problem running it on your machine, due to your trust in her. But, perhaps someone has previously hijacked the communication between you and friend in which you obtained her public key. Then, they could have possibly altered the message to contain a harmful script and passed it off as valid. The end result is that you are harmed and that you possibly blame your friend.

If you had first validated the public key being used, this would have been avoided. You could have done this by signing the key yourself. You also could have insured that the key was signed by at least one other person to whom you have given full trust. Lastly, you could have insured that the key was signed by at least three key owners to whom you have given marginal trust.

In order to sign a public key with your own private key, you must be absolutely sure that the key belongs to a person who is who they say they are. You should use a reliable method of communication to verify the key fingerprint of the key before signing it with your own. You may obtain a key fingerprint by doing this:

```
gpg --fingerprint <key owner name>
```

For example, my key fingerprint is:

```
2231 DFF0 869E E3A5 885A E7D4 F787 7A2B C9C4 0C31
```

If you communicate with the key owner, perhaps in person, or over the phone (if you know their voice), or over certified mail, you know them to be who they say they are, and the key fingerprints match, then you may sign the key. You do this by issuing this command:

```
gpg --sign-key <key owner name>
```

In order to assign trust to a key owner, you must assign the owner a value in your GnuPG trust database. You do this by editing their key:

```
gpg --edit-key <key owner name>
```

Type 'trust' and then follow the directions.

After all of this is done, if you trust someone fully to understand the significance of key signing, you can thereby, via your web of trust, validate all keys signed by them.

==========

VIII. Understanding The Need For Message Signatures

The first reason for message signatures is authenticity. It is a largely unknown fact that the From: header of email can be trivially forged. A quick glance can lead you to believe that the mail was sent by someone who might not have had anything to do with the message. A signature will help you determine if the mail was really sent by who you think it was sent by.

The second reason for signatures is integrity. Someone who has access to any piece of mail that is waiting for you (perhaps at your ISP) can change the message. This could affect something as simple as where you should meet someone to something as serious as which bank account you should transfer money to. A signature can tell you if the message was modified between the sender and your mail client.

==========

IX. Understanding How Signatures Are Made

Signatures typically are attached to the message. Detached signatures that are placed in an alternate download location are possible, but are not practical for something such as email, where all-in-one communication is the goal. The outdated standard for making attached signatures was to paste it at the bottom of the message, but the new standard involves an actual MIME attachment to the message. Regardless, Mutt can verify either style.

The important thing to understand is that the signature for each piece of mail is different and is based on both the sender's key and the content of the message. If all signatures from a person were the same, then one could just copy the signature from a mailing list post and start putting it on their own mail. Therefore, the signature is based partly on the contents of the data being signed.

Another important point to note that has been brought up recently in security forums is a weakness of signatures. Because signatures sign only the body of an email, a malicious hacker could intercept a message from one person and redirect it to another while still keeping the signature intact. For example, if your boss emails a co-worker and says "You're fired!", you could then send this message on to someone else who would think it was addressed to them and see that the signature belongs to their boss. The approach to get around this is to always make it clear in the message body to whom the message is addressed and to put things in context so that they could not be misconstrued by a third party.

==========

X. Understanding How Encryption Is Done

Aside from the obvious fact that the message is encrypted, an important thing to note about encryption is that even the sender cannot read the encrypted message. Fortunately, Mutt has two ways around this. There is an option that lets you save a plaintext copy of your outgoing messages when saving to a 'sent' folder (see Mutt's 'fcc_clear' option). You may also configure Mutt to use GnuPG's '--encrypt-to' option to additionally encrypt all encrypted mail to you so that encrypted mail in your 'sent' folder can be decrypted by you. Without one of these options, only the intended recipient can decrypt the message that was sent out, even if a copy resides in your 'sent' folder.

In order to encrypt a message to someone, you need their public key. Once you've got that, you name them as the recipient and the message is encrypted in an (as of yet) unbreakable code.

Remember that encryption alone does not guarantee authenticity, as

anyone with the recipient's public key can send them encrypted mail. Adding a signature to encrypted mail ensures that it originated from you and was encrypted by you.

==========

XI. A Practical Example of Manual Key Management

In order to understand what Mutt is going to do for you, you need to know what happens behind the scenes. In order to verify a signature or encrypt a piece of mail for someone, you need their public key. You can either download it from a person's website or request it from a keyserver, or perhaps the person will email it to you (by the way, Mutt of course has a handy shortcut to let you get keys out of an email with the Control-k keystroke). A key looks something like this (below is my public key):

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v1.0.6 (GNU/Linux)
Comment: For info see http://www.gnupg.org

mQGiBDsNpPoRBAC5A1vMkEiE2Z4AE9YYn3FOYi0K/ZDjUIz2/5wK4vX5IN149Yo5
HzOH0ftJK2oTBRaKJXF1DvUO+eyIRn02BiWoLB/Dy624LEZ7psyfGLB9v/OO1T8k
2RxAHVZefMOlJZhuAXpts2dtzkc0hBApo80Jgka1LW+i1uR/0EMj98hGwwCgl8cE
k2xD4PZGNygAQ8NkVjkkVr8EALVGxfr4+WmmOn/RFyCnFppyYnTkBIPV0/5fUSzr
z53AYXXCfbHx8eF8rRxCjmFdzSu212s7ISOqUXCahLSDcsWhjaavMzD/pA/GvWhO
vTaVprdqT6pipQTvFXE4IgD1PRwl4KCu6BsJMeVWXQkjmh36y1FIA2Ub43VQnZJE
ZBitBACaS+S7+fVFvyvK654Chh0K3Klc1zUL9TE5l0Os2veFLS2s9d6y6fpEhnB0
iABWPPXZRuPFjM8GG1cw/KtQDo/H2y2PtwG3mElF3fdZklGurvCTffWJyE2UZBz0
s1owo2LJF978mzKonLOc2sYT7Ix5mdsgdBAjZ4kH8EP7t3NukrQjSnVzdGluIFIu
IE1pbGxlciA8anVzdGluQHZveGVsLm5ldD6IVwQTEQIAFwUCOw2k+gULBwoDBAMV
AwIDFgIBAheAAAoJEPeHeivJxAwxVeEAnRdggrNekDQMyTmd5C1UjITxIhEoAJ9m
q4RDI8Fw8waK4xIGkT32gcR7H4hGBBARAgAGBQI7DcFUAAoJEN+hndnKoo+RBgMA
n3VjFLMZUJO86aqg4MEoQaouK5L4AJ9X5s07QlGNqRruAAVvyQ2tCIz7KbQoSnVz
dGluIFIuIE1pbGxlciA8anVzdGluQHNvbGlkbGludXguY29tPohXBBMRAgAXBQI7
DsfIBQsHCgMEAxUDAgMWAgECF4AACgkQ94d6K8nEDDFB/QCfYengAcA03vdik/vp
ktnSg88R8Q8An1iQ8g9UKNIBPzbhLYOSOa441y9guQENBDsNpP4QBAC6KTosKykU
PZQY+bd+hTPBHArGLFXGHC5RJtCVEU7iZHt4CEEEveAt6pAzoM/IBEIl0kHoG9Zq
KUE8DY8Wnc/c0fLBFuLxa5G5J+6NBXrQwf7mOhbpFjvUPegnJJpUaTJRCYLc4sBt
jMfaAUZJR5xRXVipLk0e8EwIuDDR1eVwbwADBwP7BU/PbNPMkGcxdrCyznBEvZAA
txqvplJwPZbOpMJ6ck/ckeXoy6G8f3vGUyGqg8bwS8SC+pxOkaBVDdpLXcfcJwJJ
R88HXIjfYZdalnMDELDHAvSsHxDsjs49tLzXRRQ0NpGPEGFiXHzMv43Ke8oYErqd
BvyyxvcrgZ6DdScYwWOIRgQYEQIABgUCOw2k/gAKCRD3h3orycQMMX4zAKCBlYSH
yeddCzek9yaiPoOD+6s3zQCfU0FwlY6o+vDLhSRerSfoA90g7gk=
=ERSn
-----END PGP PUBLIC KEY BLOCK-----
```

Once you've got the key in a file on your computer, to add it to your keyring, or your store of available keys, you would do this:

```
    gpg --import <key file>
```

GnuPG will take care of the rest. The nice thing about the whole system is that extraneous text before the beginning and ending markers is ignored, so keys can be embedded with text files that contain other information, and GnuPG will still be able to import the key(s) from those files.

You can verify that a key is now in your keyring by listing all of your keys:

```
    gpg --list-keys
```

You can also list particular keys by key ID or a word from the owner's name:

```
    gpg --list-keys <owner's first name>
```

Please note that this is the bare minimum way to use public keys. You

should also take into account your web of trust and later edit the trust value for the key's owner and/or sign the key if you really want to rely on communications using it.

==========

XII. A Practical Example of Manual Signing and Verification

Take for example a sample piece of text:

```
Here is some sample text.
```

If I saved this to a file called sample.txt and wanted to sign it before distributing it, I would do this:

```
gpg --clearsign sample.txt
```

After entering my passphrase, this will output a new, signed version at sample.txt.asc:

```
-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA1

Here is some sample text.
-----BEGIN PGP SIGNATURE-----
Version: GnuPG v1.0.6 (GNU/Linux)
Comment: For info see http://www.gnupg.org

iD8DBQE7fE+B94d6K8nEDDERAnWcAJ9ZOEiWcmfuDokPfvJtch/ERbZZFwCfeNKS
3UFUmgeigIIJQcZbhocMJUQ=
=FCcr
-----END PGP SIGNATURE-----
```

With Mutt, your text can be signed automatically, the passphrase can be cached for a customizable amount of time, and the recipient will get your message along with a MIME attachment of the signature.

In order to verify a signed file, I would do this:

```
gpg --verify sample.txt.asc
```

The output, providing I have the signer's (my own) public key, looks like this:

```
gpg: Signature made Thu 16 Aug 2001 06:56:01 PM EDT using DSA key ID C9C40C31
gpg: Good signature from "Justin R. Miller <justin@solidlinux.com>"
```

Mutt can automatically verify signatures, retrieving public keys if necessary, and will show signed messages as just the message text with some status bar info regarding the signature validity. It looks almost like regular mail, and you don't have to do a thing to verify the message's integrity besides have the sender's public key (or have it automatically fetched).

==========

XIII. A Practical Example of Manual Encryption and Decryption

In order to encrypt our file, we would run this:

```
gpg --encrypt --armor sample.txt
```

The '--armor' option causes the encrypted file to be outputted as text instead of binary data. After choosing the public key of the intended recipient, an encrypted file might look like this:

```
-----BEGIN PGP MESSAGE-----
Version: GnuPG v1.0.6 (GNU/Linux)
Comment: For info see http://www.gnupg.org
```

```
wh/HTTQIPdAOfSz9ZwsabH5dS9llSpeHHNGpO2cyR90owL3/p59IqG/7YycnaT6m
s5v45hAdGVVwHjwkUlqJElKgqD02S3i7qbOxOfE5vdeMhdew96/Axvi4QW94gY9R
ZTKjfes5QstgMHswNtn8MHIjZBShVCmRzTh04LjC8R3u/hWSvOZZes2+dJIcZOUE
AMvRRpatVVEb5AzdPRO1QaSHVwKQb+1DxeOWUviVOzp6CkPfmJPeazHRL6QjMZAM
AW5SCo4WME7Gupx931TxckRhCxDjoC63fz5sZzDB6mZhDZRxJf8HcG0L0sz4oCXr
hQEOA5Bet+UpglfCEAQAxojHuWKt/kcsdN8wZc4f0IJUU4WWEXWNoXUx+rN2Qkah
QXNDInfL5U98QkLtiPMLIifkm7YjQtqx0/9Sm8ngpFKb0lkBve/VUt+Z6bRp+2iC
m1KtBnoml6mYKCg05bRY6jrd2/0iT5sVBEHnaA==
=jW1T
-----END PGP MESSAGE-----
```

With Mutt, you can just indicate that you would like to encrypt a message, choose the intended recipient's public key from a menu, and the message is sent without you ever having to see the encryption.

Since I cannot decrypt the above file (as it wasn't intended for me), I will just show the steps necessary to decrypt a file:

```
    gpg --decrypt <encrypted file>
```

After entering my passphrase and if the message was intended for me, a new file will be written containing the decrypted text.

With Mutt, the passphrase can be cached and messages will automatically be decrypted and displayed in the message view screen, along with a brief indication of whether encryption and/or a signature was involved.

==========

XIV. Configuring Mutt For Use With GnuPG

In order to use Mutt with GnuPG, PGP support must be compiled in when building Mutt (this happens by default). Also, you need to add some directives to your .muttrc (or source a file containing these directives) which tells Mutt how to interact with GnuPG. Here are the commands that I use:

```
set pgp_decode_command="gpg %?p?--passphrase-fd 0? --no-verbose --batch --output - %f"
set pgp_verify_command="gpg --no-verbose --batch --output - --verify %s %f"
set pgp_decrypt_command="gpg --passphrase-fd 0 --no-verbose --batch --output - %f"
set pgp_sign_command="gpg --no-verbose --batch --output - --passphrase-fd 0 --armor --
detach-sign --textmode %?a?-u %a? %f"
set pgp_clearsign_command="gpg --no-verbose --batch --output - --passphrase-fd 0 --armor
--textmode --clearsign %?a?-u %a? %f"
set pgp_encrypt_only_command="pgpewrap gpg --batch --quiet --no-verbose --output - --
encrypt --textmode --armor --always-trust --encrypt-to 0xC9C40C31 -- -r %r -- %f"
set pgp_encrypt_sign_command="pgpewrap gpg --passphrase-fd 0 --batch --quiet --no-verbose
--textmode --output - --encrypt --sign %?a?-u %a? --armor --always-trust --encrypt-to
0xC9C40C31 -- -r %r -- %f"
set pgp_import_command="gpg --no-verbose --import -v %f"
set pgp_export_command="gpg --no-verbose --export --armor %r"
set pgp_verify_key_command="gpg --no-verbose --batch --fingerprint --check-sigs %r"
set pgp_list_pubring_command="gpg --no-verbose --batch --with-colons --list-keys %r"
set pgp_list_secring_command="gpg --no-verbose --batch --with-colons --list-secret-keys
%r"
set pgp_autosign=yes
set pgp_sign_as=0xC9C40C31
set pgp_replyencrypt=yes
set pgp_timeout=1800
set pgp_good_sign="^gpg: Good signature from"
```

Aside from the basic commands necessary to inter-operate with GnuPG, these directives also tell Mutt to automatically sign all outgoing messages using key ID 0xC9C40C31 (my key), to encrypt all encrypted mail to me as well (for storage in my 'sent' folder, and to cache my passphrase for a half hour.

==========

XV. Trying It All Out

Mutt should be ready to go at this point. The first time you send a message, Mutt will ask you for your passphrase, which it will then cache, then it will sign all outgoing messages. When you receive a signed message, Mutt will use the sender's public key (fetching it if it has to) to verify the signature. When encrypting messages, it will ask for your passphrase (if not cached) and will present you with a menu of public keys that it believes match the intended recipient. Lastly, when decrypting messages, it will ask you for your passphrase if the cache has expired or it hasn't been entered yet.

==========

XVI. Links For More Information

The Mutt E-Mail Client Home Page
(http://www.mutt.org/)

The GNU Privacy Guard Home Page
(http://www.gnupg.org/)

The Mutt-i, GnuPG and PGP Howto
(http://linuxdoc.org/HOWTO/Mutt-GnuPG-PGP-HOWTO.html)

The International PGP Home Page
(http://www.pgpi.org/)

Why You Should Use Encryption
(http://www.goingware.com/encryption/)

==========

XVII. Contributors

I'd like to thank the following people for assistance in revising this document:

- Dave Chapeskie, for many clarifications, corrections, recommendations, and security tips

- Thomas Roessler, for a very clear explanation of validity and trust

I'd like to thank the following people for suggestions and feedback that I may incorporate into a later revision of this document:

- Andrew McDonald
- Rick Moen
- Andy Smith

Lastly, I would like to thank you for reading this far!

==========

XVIII. Feedback

I hope that this document has been of use to you. Please let me know if there is any way that I can improve it. You can contact me via email at justin@solidlinux.com, and I would of course prefer that you use encryption when doing so using my public key, found elsewhere in this document or at http://solidlinux.com/~justin/pubkey.asc.

$Id: mutt-gnupg-howto,v 1.1.1.1 2002/03/04 00:03:18 incanus Exp $