GnuPG Commands - Examples

Table of Contents

Key Management

Encryption / Decryption

In This Section

Encryption (gpg [--options] --encrypt *file*)

You encrypt files by using the **--encrypt** command and specifying the file or data to be encrypted..

```
D:\TEMP>gpg --encrypt my-file.txt

You did not specify a user ID. (you may use "-r")

Enter the user ID.  End with an empty line: bobbone@cowtownu.edu
Added 2048g/AB53B492 2001-11-13 "Bob Bone <bobbone@cowtownu.edu>"

Enter the user ID.  End with an empty line:

D:\TEMP>
```

If you don't specify a recipient with your command, GPG prompts you to specify a recipient (whose public key must be on your keyring). Once you specify a recipient, GPG encrypts your file (my-file.txt) to a similarly named file with the extension .GPG (my-file.gpg).

You can avoid bring prompted for a recipient by specifying a **--recipient** as an option.

```
D:\TEMP>gpg --recipient Bob --encrypt my-file.txt

D:\TEMP>
```

Notice that the **-- recipient** option comes before the **--encrypt** command.

In both of the examples we've looked at, GPG encrypts the file (my-file.txt) and produces a similarly named file (my-file.gpg) as output. This new output file is an encrypted (*ciphertext*) version of the original *plaintext* file, but is is a binary file. The contents of this encrypted binary file will look like gobbledygook when opened with a simple text editor like Notepad. Binary files are perfectly fine to send intact to a recipient "as is," however, they can present problems if you want to send the encrypted contents (the *ciphertext*) in the body of an email message.

If you want to work with the encrypted contents (*ciphertext*) of your file in text format, there is a solution. GPG allows you to encrypt your file to a special format known as ASCII Armor. You can send the ASCII Armored contents in the body of an email message.

To encrypt your file and produce an ASCII Armored file as the output, use the **--armor** option. Remember that options

precede commands.

```
D:\TEMP>gpg --recipient bobbone@cowtownu.edu --armor --encrypt my-file.txt

D:\TEMP>
```

When encrypting to ASCII Armor, GPG produces an encrypted file with the extension .ASC (instead of .GPG). .GPG files are binary files; .ASC files are ASCII Armored files. In these examples, both are encrypted with the same strong level of encryption.

If you open up an ASCII Armored file, you'll see that most (but not all) of its contents are gobbledygook. This gobbledygook, however, can be used "as is" in an email because it is simple text -- it is not binary data. The strange looking block of characters in the middle contains the encrypted contents (*ciphertext*) of your original file in ASCII Armor format. You can send that ASCII Armor text block in the body of an email message.

```
D:\TEMP>type my-file.asc

-----BEGIN PGP MESSAGE-----
Version: GnuPG v1.0.7 (MingW32)

hQIOA68nz9GqU7SREAgAxWfwvpziO4N6KquxmeuYD/txfTceyXRZGVqAGFUGmOdE
+K9PCLp/+p3cFC8OcOZg8WReI4wlpYzgS3/XsB4LL9MegSHwjjI9jNsnQOr9EeLA
IgDEb1NeXZ499qnSY1ZvCy/VCF1O7H71y77VQTckpfyHgWvzkaaaheMC0r+JGLZO
0w3NCTERFJ8XaXKz/+qw4gA7xxbpT9nXVXMwEwYgiAviJBJhdYw63oTlRYGgGzPh
H2YVNv2TWnpWp816xi+sbM1ZsJJERnAZSADKFYZzYw4E73VhUlrX5YBY4WN7UmQw
yg73zfJYBuJ8+HymPhUUNH7KFqT5T2Cv4TRJgeWvxAgA3/bSCxncZ640z7KlMCMk
IskJkKRau6jeLJZKheZnyBoYiJLuJw+4FeOIkpk3ZKbWzk18kFT47x5kZA051g/p
A300n5ivHauHQz8jVTXBNF800YtkknB4+H9q5lnVYik0JsPLKGX+/sjEJ01iWaWl
wBC3poSYT+l63wNO73CDhx4VbpOzLgzbyNB6O67iuiQm2D9hLwk8L4YPOoMlfwyM
kUmsZUX709sMBHZN/9aniaVBsLxszHw9xu5OuSz/lHkckplcwb94XDLh1KGGO+1Q
LzbpFYPqe3BANLK5xxlQAAti/uk0XYltVJfUOCzyxl282X3Tp/77FtiGGb8RI1HY
hslojkAQa9gK1+f44Y8LwHH5k7fQr+Q+luqP7inoEQWbpWW4hu80Wkafv/bzI/xu
Z1qGcEVcJGJPP7QwQWUp53FbZuIq742CoxNklwvlnjhEaXa5rG2dmHUREawVzz+q
M8RkPBZIBge0SVY=
=WznL
-----END PGP MESSAGE-----
```

```
D:\TEMP>
```

As you've seen, when GPG encrypts files it produces similarly named files with the extension .GPG or .ASC as output. You can specify the name of the output file yourself, however, with the **--output** option.

```
D:\TEMP>gpg --recipient Bob --armor --output your-file.asc --encrypt my-file.txt

D:\TEMP>
```

In this example GPG encrypts my-file.txt and produces an ASCII Armored file named your-file.asc.

When using the **--encrypt** command, you may receive a warning from GPG about the "trust" in a key's owner:

```
D:\TEMP>gpg --recipient Bob --encrypt my-file.txt

gpg: checking the trustdb
gpg: checking at depth 0 signed=0 ot(-/q/n/m/f/u)=0/0/0/0/0/4
gpg: AB53B492: There is no indication that this key really belongs to the owner
2048g/AB53B492 2001-11-13 "Bob Bone <bobbone@cowtownu.edu>"
             Fingerprint: C8C5 2C0A B2A4 8174 01E8  12C8 F3CC 3233 3FAD 9F1E

It is NOT certain that the key belongs to its owner.
If you *really* know what you are doing, you may answer
the next question with yes

Use this key anyway? Y

D:\TEMP>
```

This message tells you that you have not yet signed the recipient's public key in order to establish a level of trust for that key. Although you ought to consider signing the key in order to set the trust level for the key, you can simply answer "yes" when GPG's confirms that you want to use the key and GPG will encrypt the file or message using that public key. (You can suppress this warning by adding the **--always-trust** option to your Options file.) For more information on signing keys and establishing trust levels for keys, see the **Understanding Signatures & Trust** and **Signing Keys** sections below.

(For more information on encrypting messages and files, see the **GNU Privacy Handbook**.)

Symmetric Encryption (gpg [--options] --symmetric *file*)

You can encrypt files using symmtric encryption (as opposed to public key encryption) with the **--symmetric** command. You will be prompted for a passphrase to protect the key used to encrypt the file.

```
D:\TEMP>gpg --symmetric my-file.txt

Enter passphrase: My_31337_Passphrase
Repeat passphrase: My_31337_Passphrase

D:\TEMP>
```

With symmetric encryption, you encrypt and decrypt files with the same key (which GPG generates and protects with the passphrase you supply). By contrast, the **--encrypt** command uses asymmetric encryption: you encrypt files with other people's public keys, and they decrypt with their secret (or private) keys. (For more information on symmetric vs. asymmetric encryption, see the **GNU Privacy Handbook**.) Symmetric encryption is useful if you don't plan to deliver or distribute the files to other people. For example, you may simply want to protect sensitive files on your own hard drive (not distribute them to other people).

You can combine the --symmetric command with the **--output** or **--armor** options, just like the **--encrypt** command.

(For more information on using symmetric encryption, see the **GNU Privacy Handbook**.)

Decryption (gpg [--options] --decrypt *file*)

To decrypt an encrypted file, use the **--decrypt** command. The **--decrypt** command should be used no matter whether you have received that file from someone else (who encrypted with the **--encrypt** command it using your public key), or whether you encrypted the file yourself with symmetric encryption by using the **--symmetric** command.

If the file was encrypted to your public key with the **--encrypt** command, GPG asks you for the passphrase for your secret key (often called a private key).

```
D:\TEMP>gpg --decrypt my-file.gpg
```

```
You need a passphrase to unlock the secret key for
user: "Bob Bone <bobbone@cowtownu.edu>"
2048-bit ELG-E key, ID AB53B492, created 2001-11-13 (main key ID 3FAD9F1E)

Enter passphrase: My_31337_Passphrase

gpg: encrypted with 2048-bit ELG-E key, ID AB53B492, created 2001-11-13
     "Bob Bone <bobbone@cowtownu.edu>"

This is my file.

I have many such files.

But this is the file I'm working with now.

D:\TEMP>
```

If you encrypted the file yourself with symmetric encryption (**--symmetric**), GPG asks for the passphrase that you assigned to the file.

```
D:\TEMP>gpg --decrypt my-file.gpg

gpg: CAST5 encrypted data
Enter passphrase: My_31337_Passphrase

This is my file.

I have many such files.

But this is the file I'm working with now.

D:\TEMP>
```

If you don't specify an output file for the decrypted (*plaintext*) contents, GPG merely displays the decrypted contents inline. You can specify an output file for the decrypted contents with the **--output** option.

```
D:\TEMP>gpg --output my-file.txt --decrypt my-file.gpg

You need a passphrase to unlock the secret key for
user: "Bob Bone <bobbone@cowtownu.edu>"
2048-bit ELG-E key, ID AB53B492, created 2001-11-13 (main key ID 3FAD9F1E)

Enter passphrase: My_31337_Passphrase

gpg: encrypted with 2048-bit ELG-E key, ID AB53B492, created 2001-11-13
     "Bob Bone <bobbone@cowtownu.edu>"

D:\TEMP>
```

Once GPG has decrypted the file to my-file.txt, you can open my-file.txt and view the decrypted (*plaintext*) contents.

(For more information on decrypting messages and files, see the **GNU Privacy Handbook**.)

"Encrypt-to-Self"

When you encrypt a file or message with the **--encrypt** command, you are encrypting with someone else's public key. Strangely enough, even though you encrypted the file or message yourself, you won't be able to decrypt that encrypted file and access the *plaintext*. The only person who can decrypt the file is the owner of the secret key that is the partner of the public key used to encrypt the file. That's the nature of asymmetric, public key encryption: you encrypt with the public key and decrypt with the secret key (private key). If you don't keep a copy of the *plaintext* original file yourself (and you probably shouldn't for security reasons), then you face being locked out of the very files and messages that you have encrypted and sent to other people. Happily, there is a solution: the **--encrypt-to** option.

You can include the **--encrypt-to** option in your Options file and specify your own public key. This option is often called the "encrypt-to-self" option, because it tells GPG to encrypt the message with your own public key as well as your recipient's public key. With an -**-encrypt-to** key designated in the Options file, GPG automatically encrypts messages and files to the public keys of the recipients you specify with the --recipient option as well as your own public key. The result: both you and your recipients will be able to decrypt the files or messages.

To use the -**-encrypt-to** option in your Options file, drop the leading dashes (--) and specify your own key's Key ID. (You can get your own Key ID with the **--list-keys** command.) For example, Bob (whose Key ID is 0x3FAD9F1E) could include the following line in his Options file:

```
encrypt-to 0x3FAD9F1E
```

(Note that even though Bob's key includes an encryption subkey with a separate Key ID, he simply uses the Key ID for his master key.)

Now Bob can encrypt a file to his friend Phil, just as he normally would...

```
D:\TEMP>gpg --recipient Phil --encrypt my-file.txt

D:\TEMP>
```

...and still turn around and decrypt the file himself.

```
D:\TEMP>gpg --decrypt my-file.gpg

You need a passphrase to unlock the secret key for
user: "Bob Bone <bobbone@cowtownu.edu>"
2048-bit ELG-E key, ID AB53B492, created 2001-11-13 (main key ID 3FAD9F1E)

Enter passphrase: My_31337_Passphrase

gpg: encrypted with 2048-bit ELG-E key, ID 42F0A0A0, created 1997-04-07
      "Philip R. Zimmermann <prz@pgp.com>"
gpg: encrypted with 2048-bit ELG-E key, ID AB53B492, created 2001-11-13
      "Bob Bone <bobbone@cowtownu.edu>"

This is my file.

I have many such files.

But this is the file I'm working with now.

D:\TEMP>
```

Notice GPG reports that the file (my-file.gpg) was encrypted with both Phil's key and Bob's key. GPG automatically recognizes that Bob has the secret key for one of the public keys used to encrypt the file and uses that secret key to decrypt.

It would probably be a good idea to use the "encrypt-to-self" option (**--encrypt-to**) in your Options file, as it can save you a lot of frustration down the line.

Signing / Verifying

In This Section

Signing (gpg [--options] --sign *file*)

You can sign files with the **--sign** command. GPG will prompt you for the passphrase for your secret key (private key).

```
D:\TEMP>gpg --armor --sign my-file.txt

You need a passphrase to unlock the secret key for
user: "Alice Wong <a-wong@big-corp.com>"
```

```
4096-bit RSA key, ID 49B58839, created 2002-03-27

Enter passphrase: My_31337_Passphrase

D:\TEMP>
```

Note: if you have multiple secret keys that you can use to sign, then you'll have to indicate which of those secret keys you want to use to produce a signature. To designate the secret key, you can either use the **--local-user** option with the **--sign** command, or you can use the **default-key** option in your Options file.

In the example above, we used the **--armor** option to produce ASCII Armored output (though we could have dropped the **--armor** option and produced a binary .GPG file). When we open the encrypted, ASCII Armored file, the *ciphertext* contents look just like a file that we encrypted.

```
D:\TEMP>type my-file.asc

-----BEGIN PGP MESSAGE-----
Version: GnuPG v1.0.7 (MingW32)

owGbwMvMwMSYbMOW4rG1w4JxTU4Sd26lblpmTqpeSUWJzYNN3CEZmcUKQJRbqQAW
5uXi5fJUyEgsS1XITcyrVCguTc4AyxSDpZxKSxRKoFpKMlLBMgqe6rkK5flF2Zl5
6QrlmSUZCnn55UDVnUyizKwMIEvgDmAqqxZg2GmxYbE416mL34594XlSUZ0mJXPo
l9Xpmfdn3bUTPvS5O+ZWcOvcazHGnJZ/L4tOa5k383zFAS8PqyjOV5f4ss6xB9YZ
XXcMvmly8qzbHoavXfwuXrGaVR6q9/R/7rx8YWMC52+/BZ0XlGffTK46n1GTm10p
rBBy+Mwx7czsqle6/EbF2ZtbgvdHS1y5c7qowpD98asnRrzz7HOPfl4bW/ZObHVC
3axWs4iTqdxJ/izrV7bvzbm18Fplmx77u9rqdn756q0nGhPkj9ms/LCy6G6WKyPj
zbzdsceds6MUbyYeYYrzfrUyYVZc9nW52oi9tpN+GWnG7Mr/qXlMzHZVz9bfBby/
L2YllEw1kXxotFrv4DGLpSxsu6TOFXl2S29a/73rwJPwgrjrphbXl0oYZz7IrJxm
m18sqc322UvzxTWputLZcRGHl5Tsd/1w9MvDR3HdJd6GR0/HGDFUGS30uTjV8WqH
8PN3/7Tunt/8W8Bhz4WmZW47NS7Me9fP+7Ji87QWxvuPFnLpZWWuUJf6a+hvOJlX
8yULv1zP+4OeB3we1a6QExWOSuiK6nYVr3d8MxnBbG/frP6NVnO/Sywvhye4X7x
egbX/es9FbcV09O85Q9s+Hng73XzLFPzqffmPTT6t2by0wsqDxc8XzbDwHNL89sP
z3j15EtCrhinXcnL3Ooms1PPQkH3wGXufRvbA9lsRWof22+6lf0LAA==
=erCh
-----END PGP MESSAGE-----

D:\TEMP>
```

In fact, we *have* encrypted the original file, but we encrypted it with our own secret key (as opposed to someone else's public key). In fact, signing is sometimes known as "encrypting to the private key." The recipient will decrypt the with our public key and verify the signature. The problem here, of course, is that we may want to sign the file, but leave the contents in *plaintext* form so that the contents are still readable. To do this, we'll clearsign the file with the **--clearsign** command instead of signing it with the **--sign** command.

(For more information on signing messages and files, see the **GNU Privacy Handbook**.)

Clearsigning (gpg [--options] --clearsign *file*)

To sign a message or file but leave the actual text or contents unencrypted (in *plaintext*), you can clearsign the file or message with the **--clearsign** command.

```
D:\TEMP>gpg --clearsign my-file.txt

You need a passphrase to unlock the secret key for
user: "Alice Wong <a-wong@big-corp.com>"
4096-bit RSA key, ID 49B58839, created 2002-03-27

Enter passphrase: My_31337_Passphrase

D:\TEMP>
```

When you open the clearsigned output file, you'll see that GPG has left the original contents in *plaintext* and appended a signature for the contents at the bottom.

```
D:\TEMP>type my-file.asc

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

This is my file.

I have many such files.
```

```
But this is the file I'm working with now.
-----BEGIN PGP SIGNATURE-----
Version: GnuPG v1.0.7 (MingW32)

iQIVAwUBPOCy1mM8BmRItYg4AQKWrA//fR5LKFyt+78CMtfpzkHgCVFyEe2ImsBy
FJ2HvzRIP4Bvor1iEOZ9A0fux8gBNXrvEtaDXSiGyXH+Ru4F3g1+K119fgBPRBgo
oOTbSLZSlRYWp8mRALsiWXKEHWgpy4zIHVTY6tPJdxFBZYJXnQj/4S6MRP+eJdam
rU8ufExxaqQPw+KCNEVCSk1yHZ886k6MTSa1oDqUOLiM1cBDCtD8Jv+BE0gLHPb9
1h7lEka8QGNe+P7iiUzvsuD7HCL6dGb6T70/KBBHIP6lDwOgUX3eTd8e+I3jczs9
RyEmd6G4swM3IzCD1km+SN5/k5QsMjd6Lw5fB95Mroi47QNpya8ifYbMgCg0+BVm
c7QOwr79+9cJiKhEICbMf5pKQWzP/AznaYlM0IOGGCvxa5loLl7BbtvktVMocitF
zWM9SB0kmSu3OlMxjXYcBsyHCHN4dTpCD9d1jfbgth9YV06sWpONLohdaWx+n9kO
CxsSDGI+aW8sGKHWonw0Uy4UAvUzY3tiZTzTF+FzoJzhy13KK1j4Y0MMx1jZ68f9
R9wSKVdiyXwuMXkWWK0uxSZuBz4mTofZ7YmFm7UdxOH4bMnO+rWNCSPR7md+X0j1
nQSwtxEnIu7Tucb/ZG3t9kR+KTByPTu7tHINr4HFd8m2Cu7Wi10TP/EBtXbtYA/1
SBaUXcbgCD8=
=Hn6O
-----END PGP SIGNATURE-----

D:\TEMP>
```

When clearsigning files, it is not necessary to use the **--armor** option. GPG automatically uses ASCII Armor for the clear signature it appends to the bottom of the encrypted contents (*ciphertext*). Of course, it only makes sense to clearsign simple text files. If you clearsign binary files, GPG *will* produce an ASCII Armored signature, but the original contents will still be binary gobbledygook.

(For more information on clearsigning messages and files, see the **GNU Privacy Handbook**.)

Detached Signatures (gpg [--options] --detach-sign *file*)

You can also produce a signature as a detached signature file. When creating detached signatures, GPG leaves the original file "as is" and creates a separate file that contains only the digital signature. To sign a file and produce a detached signature, use the **--detach-sign** command..

```
D:\TEMP>gpg --detach-sign my-file.zip
```

```
You need a passphrase to unlock the secret key for
user: "Alice Wong <a-wong@big-corp.com>"
4096-bit RSA key, ID 49B58839, created 2002-03-27

Enter passphrase: My_31337_Passphrase

D:\TEMP>
```

Once you enter your passphrase, GPG creates a detached signature file (my-file.sig) that is named similar to the file being signed (my-file.txt).

.SIG files are binary files like .GPG files. If you prefer GPG to produce detached signature files in ASCII Armor format, use the **--armor** option.

```
D:\TEMP>gpg --armor --detach-sign my-file.zip

You need a passphrase to unlock the secret key for
user: "Alice Wong <a-wong@big-corp.com>"
4096-bit RSA key, ID 49B58839, created 2002-03-27

Enter passphrase: My_31337_Passphrase

D:\TEMP>
```

As you might expect, you can open the ASCII Armored detached signature file (which has the .ASC extension) and view the contents.

```
D:\TEMP>type my-file.asc

-----BEGIN PGP SIGNATURE-----
Version: GnuPG v1.0.7 (MingW32)

iQIVAwUAPOC1eGM8BmRItYg4AQJaXxAAjS3DDRXllMhL4p2z/5tK1/wGielnpofg
VYCtx0Gga+ca3h1pM0xBpw3n5gXehSlFr3Arhh0ZE7rAfwqwHDUvIPZN05LPsTh8
exCyq6yWRqT4KVBqRwV47bGgoRV7bz5hT4UUBQcoevGtywUmon+g4sB/OmFF8QjU
```

```
I2nf8OvUZyZ4SZMwNuQEwo84kfL4kYFgla062ruCNVSaWSFaDHHbzUKOBdKqM6Zp
wi8tTNKm8giqqaUGoCHUnkszf0+evIj6efILLxk1EylQmTLD8/cBS5GsCpiFdiH0
hRV1MdqgR11hrR9YtomVPFTT86eE8QI4Wk05TMr5r1VV8nKdh/+2IkGfkycBQte5
YPqk+nYRgC6yRIW1ylg7WJTGVhtsxekMC4MAs3ZdhqQMCwD99cCJ4zHd56LEuvp3
z8eC9VKB9r141D0QZxBFUy1I1abWQRWM/+zoMMoV6jP3b9iMwWvNRZAEqEa4MI7u
jT45OjUwFd/ZlpVV8ZgmbrFCjwJK3JbERFLY24nn/hY6REAgplVQHvWspgzG/pBH
WgbkKm+0W6gMEUQ+w1rvKOZOnkF3tju8GqP6NZ79D+T+oJi8Avd689Mz7e8ykzap
rTTymknv17XOOFNTzZK77slx4nKjHXo9yqtpWB/Ek3y42LdPKALaaeBIOc+E7tky
7IFQXTZXJgw=
=sws/
-----END PGP SIGNATURE-----

D:\TEMP>
```

Once you have produced a detached signature, you should send both the original file that you signed as well as the detached signature file to your recipients. It does your recipients no good to send just the detached signature file; the detached signature file contains only the signature, not the actual contents of the file that you signed.

(For more information on creating detached signatures, see the **GNU Privacy Handbook**.)

Verifying Signed & Clearsigned Files **(gpg [--options] --decrypt *file*)**

To verify the signatures on files that you have received from others, use the **--decrypt** command.

```
D:\TEMP>gpg --decrypt my-file.gpg

This is my file.

I have many such files.

But this is the file I'm working with now.

gpg: Signature made 05/14/02 02:06:03  using RSA key ID 49B58839
gpg: Good signature from "Alice Wong <a-wong@big-corp.com>"

D:\TEMP>
```

GPG shows you the contents of the file and verifies the signature. The **--decrypt** command should be used with files signed with the **--sign** command as well as with files clearsigned with the **--clearsign** command.

We can specify an output file with the **--output** option.

```
D:\TEMP>gpg --output your-file.txt --decrypt my-file.gpg

gpg: Signature made 05/14/02 02:06:03  using RSA key ID 49B58839
gpg: Good signature from "Alice Wong <a-wong@big-corp.com>"

D:\TEMP>
```

The decrypted file (your-file.txt) contains only the *plaintext* contents. GPG still reports the results of its signature verification inline.

Keep in mind that to verify signature from someone, you must have that person's public key on your keyring. If you don't have that person's public key, you won't be able to verify the signature.

(For more information on verifying signatures, see the **GNU Privacy Handbook**.)

Verifying Detached Signatures (gpg [--options] --verify *sigfile signed_files*)

To verify files with detached signatures, use the **--verify** command and specify the detached signature file as well as the files that were signed..

```
D:\TEMP>gpg --verify my-file.sig my-file.txt

gpg: Signature made 05/14/02 02:13:29  using RSA key ID 49B58839
gpg: Good signature from "Alice Wong <a-wong@big-corp.com>"

D:\TEMP>
```

GPG doesn't show you the contents of the original file that was signed. The original file remains in *plaintext* form, so you can view it as you normally would without any special command.

We can use the **--verify** command to verify files signed with the **--sign** or **--clearsign** commands...

```
D:\TEMP>gpg --verify my-file.gpg

gpg: Signature made 05/14/02 02:06:03  using RSA key ID 49B58839
gpg: Good signature from "Alice Wong <a-wong@big-corp.com>"

D:\TEMP>
```

...but GPG doesn't show us the *plaintext* contents of the signed files. It only verifies the signature on the file. That's why we use the **--decrypt** command to verify files that have been signed with the **--sign** or **--clearsign** commands: we want to view the contents as well as verify the signature.

Keep in mind that to verify signature from someone, you must have that person's public key on your keyring. If you don't have that person's public key, you won't be able to verify the signature.

(For more information on verifying detached signatures, see the **GNU Privacy Handbook**.)

Understanding Signatures & Trust

In all of the signature verification examples that we looked at above, GPG reported the following when verifying a signature:

```
gpg: Signature made 05/14/02 02:13:29  using RSA key ID 49B58839
gpg: Good signature from "Alice Wong <a-wong@big-corp.com>"
```

This signature verification is what we want and expect: a "good signature." But GPG may not always give you the same unambiguous report when verifying signatures. In some situations, GPG's signature verification report may include something else: a "warning," such as the following:

```
gpg: Signature made 05/14/02 02:13:29  using RSA key ID 49B58839
gpg: Good signature from "Alice Wong <a-wong@big-corp.com>"
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
Fingerprint: AE15 DB8D F29B 00F8 D213 1C18 633C 0664 49B5 8839
```

This warning is similar to one you might receive when encrypting a message or file to someone else with the **--encrypt** command:

```
gpg: checking the trustdb
gpg: checking at depth 0 signed=0 ot(-/q/n/m/f/u)=0/0/0/0/0/4
gpg: AB53B492: There is no indication that this key really belongs to the owner
2048g/AB53B492 2001-11-13 "Bob Bone <bobbone@cowtownu.edu>"
             Fingerprint: C8C5 2C0A B2A4 8174 01E8  12C8 F3CC 3233 3FAD 9F1E

It is NOT certain that the key belongs to its owner.
```

In the case of the signature verification warning, you're probably wondering why GPG reports "Good signature from..." on one line and then issues a "Warning" about the key not being "certified with a trusted signature" on the very next line. It might seem that there's a contradiction here -- i.e., how can a signature be both "good" and "untrusted" at the same time? -- but there really isn't a contradiction at all. Let's look in detail at what GPG is telling us.

The first line tells us what key was used to make the signature and when the signature was made.

```
gpg: Signature made 05/14/02 02:13:29  using RSA key ID 49B58839
```

The next line confirms that the signature on the file was in fact made using this particular key (with Key ID 49B58839). It also tells us that the User ID on that key is for Alice Wong (a-wong@big-corp.com) and that the signature is valid or "good."

```
gpg: Good signature from "Alice Wong <a-wong@big-corp.com>"
```

What GPG has done is verify that the signature on the file does indeed match the signature it would expect from this key for this file. The fact that the signature matches tells us that the file has not been altered or tampered with in transit. If the file had been altered in transit, GPG would have reported a "bad signature" instead:

```
gpg: Signature made 05/14/02 02:13:29 CDT using RSA key ID 49B58839
gpg: BAD signature from "Alice Wong <a-wong@big-corp.com>"
```

In other words, the signature on the file didn't match what GPG expected. Perhaps the contents of the message in the file were altered somehow. Another potential cause for this "bad signature" is that the signature itself was altered or doesn't

even belong to the contents of the file (it might be a signature that was produced for some other file). Whatever the cause, it's a "bad signature."

The example we're looking at, though, has a "good signature." The problem with this signature is that it was produced with a key that is not "trusted."

```
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
Fingerprint: AE15 DB8D F29B 00F8 D213 1C18 633C 0664 49B5 8839
```

As GPG insists, we don't have any reason to suppose that the key used to produce the signature does in fact belong to the person identified in the key's User ID. In other words, while we know that this key was indeed used to make the signature on the file and that the file hasn't been altered, we do NOT know that this key really belongs to Alice. (The same holds true for the warning GPG issues with the **--encrypt** command: you don't know for certain that the person specified in the User ID of the key to which you're encrypting is actually the owner of the key.) It's always possible that someone other than Alice generated the key, used Alice's name and email address in the User ID, and is now masquerading as Alice. In short, the key is "untrusted."

We can make this key "trusted" by "certifying" the key. To certify the key, we need to sign the key. Once we sign and certify the key, the trust level associated with the key will change. In this example, GPG has warned us that we are using an "untrusted key" because we have not yet certified the key that was used to verify the signature. Put another way, GPG is telling us that we have have not "certified" the key used to make the signature by signing that key with our own secret key in order to change the trust level associated with the key. ("*This key is not certified with a trusted signature!*") Once we sign the key to certify it, the key will become "trusted."

The trust level on a key is a measure of our confidence in the identity of the owner of the key. If we are confident that this key does actually belong to Alice (who is listed in the key's User ID), we can change the trust level on Alice's key by signing it with our own secret key. Once we sign Alice's key and change the trust level associated with the key, GPG will no longer warn us that we are using an "untrusted" key when we verify signatures from Alice. Instead, GPG will simply report:

```
gpg: Signature made 05/14/02 02:13:29  using RSA key ID 49B58839
gpg: Good signature from "Alice Wong <a-wong@big-corp.com>"
```

And when encrypting files or messages with the **--encrypt** command, GPG will no longer warn you about the "untrusted" nature of the key -- it will simply encrypt the file without complaint.

To learn how to sign keys and change the trust level on a key, see the **Signing Keys** section below. You can also suppress GPG's trust warnings by adding the **--always-trust** option to your Options file. (For more general information on signing keys and using the Web of Trust, see the **GNU Privacy Handbook**.)

Combining Commands

In This Section

Most commands used by GPG cannot be combined with other commands. Although you can use multiple options at the same time (as we did in many of our examples earlier), commands must be used one at a time. In some situations, though, you can combine commands instead of issuing them separately.

Encrypt & Sign (gpg [--options] --encrypt -- sign *file*)

You can encrypt and sign a file at the same time by using the **--encrypt** and **--sign** commands simultaneously.

```
D:\TEMP>gpg --armor --recipient Bob --encrypt --sign my-file.txt

You need a passphrase to unlock the secret key for
user: "Alice Wong <a-wong@big-corp.com>"
4096-bit RSA key, ID 49B58839, created 2002-03-27

Enter passphrase: My_31337_Passphrase

D:\TEMP>
```

With the combined **--encrypt** and **--sign** commands, GPG produces a signed and encrypted file.

```
D:\TEMP>type my-file.asc

-----BEGIN PGP MESSAGE-----
Version: GnuPG v1.0.7 (MingW32)

owGbwMvMwMSYbMOW4rG1w4JxTU4Sd26lblpmTqpeSUWJzYNN3CEZmcUKQJRbqQAW
5uXi5fJUyEgsS1XITcyrVCguTc4AyxSDpZxKSxRKoFpKMlLBMgqe6rkK5flF2Zl5
6QrlmSUZCnn55UDVnUyizKwMIEvgDmAqqxZg2GmxYbE416mL34594XlSUZ0mJXPo
l9Xpmfdn3bUTPvS5O+ZWcOvcazHGnJZ/L4tOa5k383zFAS8PqyjOV5f4ss6xB9YZ
XXcMvmly8qzbHoavXfwuXrGaVR6q9/R/7rx8YWMC52+/BZ0XlGffTK46n1GTm10p
rBBy+Mwx7czsqle6/EbF2ZtbgvdHS1y5c7qowpD98asnRrzz7HOPfl4bW/ZObHVC
3axWs4iTqdxJ/izrV7bvzbm18Fplmx77u9rqdn756q0nGhPkj9ms/LCy6G6WKyPj
zbzdsceds6MUbyYeYYrzfrUyYVZc9nW52oi9tpN+GWnG7Mr/qXlMzHZVz9bfBby/
L2YllEw1kXxotFrv4DGLpSxsu6TOFXl2S29a/73rwJPwgrjrphbXl0oYZz7IrJxm
m18sqc322UvzxTWputLZcRGHl5Tsd/1w9MvDR3HdJd6GR0/HGDFUGS30uTjV8WqH
8PN3/7Tunt/8W8Bhz4WmZW47NS7Me9fP+7Ji87QWxvuPFnLpZWWuUJf6a+hvOJlX
8yULv1zP+4OeB3we1a6QEexWOSuiK6nYVr3d8MxnBbG/frP6NVnO/Sywvhye4X7x
egbX/es9FbcV09O85Q9s+Hng73XzLFPzqffmPTT6t2by0wsqDxc8XzbDwHNL89sP
z3j15EtCrhinXcnL3Ooms1PPQkH3wGXufRvbA9lsRWof22+6lf0LAA==
=erCh
-----END PGP MESSAGE-----

D:\TEMP>
```

The **--encrypt** command cannot be combined with the **--clearsign** or **--detach-sign** commands. Indeed, it wouldn't make sense to do so, because we want to encrypt the original file, not leave it in *plaintext* form, as both the **--clearsign** and **--detach-sign** commands do.

Decrypt & Verify **(gpg [--options] --decrypt *file*)**

To decrypt and verify a file that has been both signed and encrypted, use the **--decrypt** command. GPG will decrypt the contents and verify the signature automatically..

```
D:\TEMP>gpg --decrypt my-file.asc

You need a passphrase to unlock the secret key for
```

```
user: "Bob Bone <bobbone@cowtownu.edu>"
2048-bit ELG-E key, ID AB53B492, created 2001-11-13 (main key ID 3FAD9F1E)

Enter passphrase: My_31337_Passphrase

gpg: encrypted with 2048-bit ELG-E key, ID AB53B492, created 2001-11-13
     "Bob Bone <bobbone@cowtownu.edu>"

This is my file.

I have many such files.

But this is the file I'm working with now.

gpg: Signature made 05/14/02 02:38:06  using RSA key ID 49B58839
gpg: Good signature from "Alice Wong <a-wong@big-corp.com>"

D:\TEMP>
```

Of course, you can always specify an output file for the decrypted contents.

```
D:\TEMP>gpg --output your-file.txt --decrypt my-file.asc

You need a passphrase to unlock the secret key for
user: "Bob Bone <bobbone@cowtownu.edu>"
2048-bit ELG-E key, ID AB53B492, created 2001-11-13 (main key ID 3FAD9F1E)

Enter passphrase: My_31337_Passphrase

gpg: encrypted with 2048-bit ELG-E key, ID AB53B492, created 2001-11-13
     "Bob Bone <bobbone@cowtownu.edu>"
gpg: Signature made 05/14/02 02:38:06  using RSA key ID 49B58839
gpg: Good signature from "Alice Wong <a-wong@big-corp.com>"

D:\TEMP>
```

The output file will contain only the decrypted (*plaintext*) contents of the encrypted file. GPG still reports the results of its signature verification inline.

Key Management

In This Section

Key Generation (gpg [--options] --gen-key)

Before you can receive encrypted messages and files from others or digitally sign files and messages to send to others, you must generate a keypair for yourself. A keypair consists of a public key -- which others use to encrypt messages to you and to verify signatures that you make -- and a secret key (often called a private key) -- which you use to decrypt messages sent to you by others and to sign files and messages that you send to others. (For more information on encryption, ciphers, and keys, see the **GNU Privacy Handbook**.)

The key generation process in GPG involves several steps and requires you to make a several important decisions along the way. We start the key generation process with the **--gen-key** command.

```
D:\Programs\gnupg>gpg --gen-key

gpg (GnuPG) 1.0.7; Copyright (C) 2002 Free Software Foundation, Inc.
This program comes with ABSOLUTELY NO WARRANTY.
This is free software, and you are welcome to redistribute it
under certain conditions. See the file COPYING for details.
```

```
Please select what kind of key you want:
   (1) DSA and ElGamal (default)
   (2) DSA (sign only)
   (4) ElGamal (sign and encrypt)
   (5) RSA (sign only)
Your selection?
```

The first choice we must make is the type of key to generate. As this is our first keypair, we should generate a keypair that can be used to both sign and encrypt, which means we'll choose option (1) or (4). We'll choose (1) DSA/ElGamal (default). DSA/ElGamal keypairs include a DSA master signing key and an ElGamal encryption subkey. Both the master signing key and the encryption subkey will have public and secret keys. (For a discussion of DSA/ElGamal keypairs, see the **Note on Key Types & Subkeys** section below.)

Next we choose a keysize. Note that we are choosing the size for the ElGamal encryption subkey -- the size of the master DSA signing key is fixed at 1024 bits.

```
Your selection? 1
DSA keypair will have 1024 bits.
About to generate a new ELG-E keypair.
              minimum keysize is  768 bits
              default keysize is 1024 bits
    highest suggested keysize is 2048 bits
What keysize do you want? (1024) 2048
Requested keysize is 2048 bits
```

The larger the keysize, the stronger the key and the more robust the encryption used for messages and files. The minimum keysize you ought to consider using is 2048 bits. The largest keysize you can choose is 4096 bits.(An intermediate step between the two would be 3072 bits.) (For more information on key sizes, see the **GNU Privacy Handbook**.)

GPG now asks us to specify an expiration. We can always **revoke our key** in the future should we decide to, so we'll choose no expiration. (For more information on expiration dates, see the **GNU Privacy Handbook**.)

```
Please specify how long the key should be valid.
         0 = key does not expire
      <n>  = key expires in n days
      <n>w = key expires in n weeks
```

```
      <n>m = key expires in n months
      <n>y = key expires in n years
Key is valid for? (0) 0
Key does not expire at all
Is this correct (y/n)? Y
```

Next, we must create a User ID for our keypair. The User ID is a kind of name tag for our keypair. It lets those who get our public key know who that public key belongs to. The User ID, in other words, identifies us as the owner of the keypair. GPG asks us for a name and email address to create the User ID. We can **change** both at a later time should we need to.

```
You need a User-ID to identify your key; the software constructs the user id
from Real Name, Comment and Email Address in this form:
    "Heinrich Heine (Der Dichter) <heinrichh@duesseldorf.de>"

Real name:  George P. Gumbel
Email address:  gpgumbel@cowtownu.edu
Comment:
You selected this USER-ID:
    "George P. Gumbel <gpgumbel@cowtownu.edu>"

Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit? O
```

Finally, we must specify a passphrase for our secret key (private key). This passphrase is critical, as GPG uses it to protect and control access to our secret key. If your passphrase is compromised or broken, anyone who gets a hold of your secret key will be able to use it to decrypt messages sent to you and to sign files sent to others just as if they were you.

A strong passphrase should consist of a mix of upper and lowercase letters, numbers, and non-standard keyboard characters. Your passphrase should not use familiar names and numbers (e.g., names of friends and family, names of pets, social security numbers, etc.). It should also avoid repeated characters as much as possible. Finally, your passphrase should be long (a standard 8 character password will not suffice) and yet easy to memorize.

```
You need a Passphrase to protect your secret key.

Enter passphrase: My_31337_Passphrase
Repeat passphrase: My_31337_Passphrase
```

GPG asks you to enter your passphrase twice. As you type, GPG will not "echo" what you type on screen. If what you type differs even slightly from the first time to the second time, GPG will ask you to type your passphrase from scratch.

Once you successfully enter a passphrase, don't forget it. If you forget your passphrase, you'll lose access to your own secret key, and you won't be able to regain access to it. Also, don't write it down anywhere. If other people learn your passphrase, your secret key will no longer be secret, and those persons will be able to decrypt and read all of your encrypted messages and files. They'll also be able to sign messages and files just as if they were you.

After confirming your passphrase, GPG generates a keypair for you. While it is generating a keypair, GPG asks you to move the mouse around and type randomly on the keyboard in order to generate "seed" data to randomize the key generation process (thus making your keypair stronger and harder to break).

```
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
++++++++++.+++++++++++++++..++++++++++.+++++++++++++++.++++++++++++++++
++++++++++..++++++++++++++++++++++++++++++++.++++++++++>++++++++++>++
.....+++++
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
++++++++++++++++++++.++++++++++++++++++++...++++++++++++++++++++..++++++
+.+++++...++++++++++++++++++++.++++++++++++++++++++++++++.++++++++++.++>.
+++++++++.............................................................
.....................................>...+++++.........................
.....................................................................+++++^^^^
^^^^^
public and secret key created and signed.
key marked as ultimately trusted.

pub  1024D/0A484ECB 2002-05-26 George P. Gumbel <gpgumbel@cowtownu.edu>
     Key fingerprint = 572B 8AA3 075D 1E6E 5B80  D9CB CE18 FB44 0A48 4ECB
sub  2048g/C31174A2 2002-05-26
```

```
D:\Programs\gnupg>
```

Once GPG has finished generating a keypair, it adds the keypair (both the public and secret keys) to our keyring.. Notice that GPG has signed our public key with our secret key (an act known as self-signing) and has marked our public key as "ultimately trusted." (See the **GNU Privacy Handbook** for more information on the Web of Trust.)

Now that you've generated a keypair, you ought to consider **creating a revocation certificate** as well.

(For more information on generating keypairs, see the **GNU Privacy Handbook**.)

A Note on Key Types & Subkeys

The DSA/ElGamal keypair that we generated above consists of a public key and a secret key (or private key). It also has a special type of key known as a subkey (which has its own public and secret keys). Subkeys are often used to encrypt, but not sign. DSA/ElGamal keypairs are a common combination of master signing key and encryption subkey. In some cases, as was the case with our DSA/ElGamal keypair, GPG will create the necessary subkey for you when you generate a keypair. In other cases you will have to create a subkey yourself, depending on the type of keypair you choose to generate.

DSA & ElGamal Keypairs

When you select choice (1) (DSA and ElGamal) from the key type menu, GPG automatically creates a keypair consisting of a DSA master signing key and an ElGamal encryption subkey. (ElGamal keys are a variant of the Diffie-Hellman keys familiar to PGP users.) Each key on your keypair will be used for a particular task (signing or encryption). Moreover, each will probably be different in size: the DSA master signing key is limited to 1024 bits (GPG sets this size automatically); the ElGamal encryption subkey can be up to 4096 bits (GPG allows you to set this size yourself).

GPG lists the master signing key and encryption subkey separately when providing basic information about your keypair (such as with the **--edit-key** command).

```
D:\Programs\gnupg>gpg --edit-key george

gpg (GnuPG) 1.0.7; Copyright (C) 2002 Free Software Foundation, Inc.
This program comes with ABSOLUTELY NO WARRANTY.
This is free software, and you are welcome to redistribute it
under certain conditions. See the file COPYING for details.
```

```
Secret key is available.

gpg: checking the trustdb
gpg: checking at depth 0 signed=1 ot(-/q/n/m/f/u)=0/0/0/0/0/7
gpg: checking at depth 1 signed=0 ot(-/q/n/m/f/u)=1/0/0/0/0/0
pub  1024D/0A484ECB  created: 2002-05-26 expires: never      trust: u/u
sub  2048g/C31174A2  created: 2002-05-26 expires: never
(1). George P. Gumbel <gpgumbel@cowtownu.edu>

Command>
```

Notice that the main public key (pub) is a DSA (D) signing key of 1024 bits; the ElGamal (g) encryption subkey (sub) is 2048 bits and is listed separately. (You might compare these two keys with the single key from the RSAv4 key generation example below.)

When using your DSA/ElGamal keypair to encrypt or sign, GPG (and PGP) automatically selects the proper key (master key or subkey) to use.

RSAv4 Keypairs (Sign & Encrypt)

Keypairs do not have to include an encryption subkey. It is possible to create an RSAv4 keypair, for example, that consists of but one key which is used both to sign and encrypt. RSAv3 keys (discussed **below**) also use a single signing and encryption key.

By default, though, GPG will not let you create RSAv4 keypairs with a single signing and encryption key. Since we're using the **"Nullify" build** of GPG 1.0.7, we can get a wider range of choices for key types by using the **--expert** option in conjunction with the **--gen-key** command.

```
D:\Programs\gnupg>gpg --expert --gen-key

gpg (GnuPG) 1.0.7; Copyright (C) 2002 Free Software Foundation, Inc.
This program comes with ABSOLUTELY NO WARRANTY.
This is free software, and you are welcome to redistribute it
under certain conditions. See the file COPYING for details.

Please select what kind of key you want:
```

```
   (1) DSA and ElGamal (default)
   (2) DSA (sign only)
   (4) ElGamal (sign and encrypt)
   (5) RSA (sign only)
   (7) RSA (sign and encrypt, not recommended)
Your selection? 7
```

Notice the new choice (7): an RSAv4 keypair that can encrypt and sign. With the **--expert** option, we now have three choices for keypairs that can encrypt and sign.

If we choose (7) RSA (sign and encrypt), GPG will create a keypair with a single signing and encryption key. Once we're finished creating the keypair, we can edit it (**--edit-key**) and view basic information about the key.

```
D:\Programs\gnupg>gpg --edit-key gpgumbel@cowtownu.edu

gpg (GnuPG) 1.0.7; Copyright (C) 2002 Free Software Foundation, Inc.
This program comes with ABSOLUTELY NO WARRANTY.
This is free software, and you are welcome to redistribute it
under certain conditions. See the file COPYING for details.

Secret key is available.

pub  2048R/32E3B3DF  created: 2002-05-14 expires: never      trust: u/u
(1). George P. Gumbel <gpgumbel@cowtownu.edu>

Command>
```

Note that there is no subkey -- only a single key used to sign and encrypt.

RSAv4 keypairs with single signing and encryption keys are not recommended, however. Re-using an RSA key for encryption and signing exposes the key to potential cryptoanalytic attacks. If you're interested in generating an RSAv4 keypair, you'd be better off generating an RSAv4 keypair with a master signing key and an encryption subkey. For a discussion of this type of RSAv4 keypair, see the next section.

RSAv4 Keypairs (w/ Encryption Subkey)

Although we generated an RSAv4 keypair in the previous section, we generated a special kind of RSAv4 keypair that uses a single key for encryption and signing. Here is the RSAv4 key from our example above:

```
public and secret key created and signed.
key marked as ultimately trusted.

pub  2048R/32E3B3DF 2002-05-14 George P. Gumbel <gpgumbel@cowtownu.edu>
     Key fingerprint = 448E D686 3AFC 8148 07E9  1DD3 329E D4B4 32E3 B3DF


D:\Programs\gnupg>
```

A single RSA key for encryption and signing is not recommended because of its vulnerability to certain types of attacks. In fact, the only way we were able to generate such a key was with the **--expert** option.

Normally, to generate an RSAv4 keypair, we would select choice (5) from the default menu of key types. Without the **--expert** option, choice (5) is RSA (sign only). After creating the master signing key, we can generate an encryption subkey.

```
D:\Programs\gnupg>gpg --gen-key

gpg (GnuPG) 1.0.7; Copyright (C) 2002 Free Software Foundation, Inc.
This program comes with ABSOLUTELY NO WARRANTY.
This is free software, and you are welcome to redistribute it
under certain conditions. See the file COPYING for details.

Please select what kind of key you want:
   (1) DSA and ElGamal (default)
   (2) DSA (sign only)
   (4) ElGamal (sign and encrypt)
   (5) RSA (sign only)
Your selection?
```

When it has finished generating the RSA master signing key, GPG warns us that the key can be used for digital signatures only.

```
public and secret key created and signed.
key marked as ultimately trusted.

pub  2048R/01B4D4E6 2002-05-20 Bob Bone <bobbone@cowtownu.edu>
     Key fingerprint = FD87 06D4 9537 DBA8 DB34  7C94 2A4D 50AC 01B4 D4E6

Note that this key cannot be used for encryption. You may want to use
the command "--edit-key" to generate a secondary key for this purpose.

D:\Programs\gnupg>
```

As GPG recommends, we can create an RSA encryption subkey. To create a subkey, use the **--edit-key** command and issue the **addkey** command from the **--edit-key** command line. GPG will ask for our passphrase before it allows us to make changes to our keypair.

```
D:\Programs\gnupg>gpg --edit-key bob

gpg (GnuPG) 1.0.7; Copyright (C) 2002 Free Software Foundation, Inc.
This program comes with ABSOLUTELY NO WARRANTY.
This is free software, and you are welcome to redistribute it
under certain conditions. See the file COPYING for details.

Secret key is available.

gpg: checking the trustdb
gpg: checking at depth 0 signed=1 ot(-/q/n/m/f/u)=0/0/0/0/0/5
gpg: checking at depth 1 signed=0 ot(-/q/n/m/f/u)=1/0/0/0/0/0
pub  2048R/01B4D4E6  created: 2002-05-20 expires: never      trust: u/u
(1). Bob Bone <bobbone@cowtownu.edu>

Command> addkey

Key is protected.

You need a passphrase to unlock the secret key for
user: "Bob Bone <bobbone@cowtownu.edu>"
2048-bit RSA key, ID 01B4D4E6, created 2002-05-20
```

```
Enter passphrase: My_31337_Passphrase
```

GPG then asks what kind of subkey type we want to generate.

```
Please select what kind of key you want:
   (2) DSA (sign only)
   (3) ElGamal (encrypt only)
   (4) ElGamal (sign and encrypt)
   (5) RSA (sign only)
   (6) RSA (encrypt only)
Your selection? 6
```

We'll choose (6) RSA (encrypt only), a choice that did not appear when we originally generated our master signing key above. Next we specify a keysize and expiration for the subkey.

```
What keysize do you want? (1024) 2048
Requested keysize is 2048 bits
Please specify how long the key should be valid.
         0 = key does not expire
      <n>  = key expires in n days
      <n>w = key expires in n weeks
      <n>m = key expires in n months
      <n>y = key expires in n years
Key is valid for? (0) 0
Key does not expire at all
Is this correct (y/n)? Y
```

Finally, GPG confirms our choice and generates the subkey. You probably noticed that GPG did not ask us for information to create a User ID -- that's because the master signing key already has a User ID.

```
Really create? Y
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
```

```
generator a better chance to gain enough entropy.
.+++++
....+++++

pub  2048R/01B4D4E6  created: 2002-05-20 expires: never      trust: u/u
sub  2048R/89EFD32C  created: 2002-05-20 expires: never
(1). Bob Bone <bobbone@cowtownu.edu>

Command> save

D:\Programs\gnupg>
```

Now our RSA keypair has an encryption subkey (sub). Don't forget to **save** your changes.

If we had neglected to generate an encryption subkey and simply distributed our public key "as is," other people wouldn't have been able to use it to encrypt messages to us (they could have used it only to verify digital signatures from us). If they attempted to use our key to encrypt, they would receive an error message from GPG.

```
D:\TEMP>gpg --recipient bob --encrypt my-file.txt

gpg: bob: skipped: unusable public key
gpg: my-file.txt: encryption failed: unusable public key

D:\TEMP>
```

Whether you choose to generate an RSAv4 keypair that uses the same key for encryption and signing or an RSAv4 keypair with a master signing key and an encryption subkey is up to you. Remember, though, that most crypto experts recommend not re-using the same RSA key for encryption and signing.

RSAv3 Keypairs

The **"Nullify" build** of GPG 1.0.7 (which was compiled with the **RSAv3 key patch**) allows you to generate RSAv3 keys, RSAv3 keys (known to PGP 7.x users as "RSA legacy keys") are PGP 2.6.x compatible. RSAv3 keys use a single key for encryption and signing -- there are no subkeys. RSAv3 keys are not recommended for precisely this reason -- re-using an RSA key for encryption and signing exposes the key to potential cryptoanalytic attacks. (This is the same reason that standard RSAv4 keys use a master signing key and an encryption subkey, as we discussed in the **previous section**.)

Since we're using the **"Nullify" build** of GPG 1.0.7 (which was compiled with the RSAv3 key patch), we can use the **--expert** and **--pgp2** options in order to generate a PGP 2.6 compatible RSAv3 keypair.

```
D:\Programs\gnupg>gpg --expert --pgp2 --gen-key

gpg (GnuPG) 1.0.7; Copyright (C) 2002 Free Software Foundation, Inc.
This program comes with ABSOLUTELY NO WARRANTY.
This is free software, and you are welcome to redistribute it
under certain conditions. See the file COPYING for details.

Please select what kind of key you want:
   (1) DSA and ElGamal (default)
   (2) DSA (sign only)
   (4) ElGamal (sign and encrypt)
   (5) RSAv3 (sign and encrypt, PGP 2.6 compatible, not recommended)
Your selection? 5
```

Notice choice (5), which is now slightly different from the "original" choice (5) RSA (sign only). Instead of RSAv4 keypair that can sign only, choice (5) is now an RSAv3 keypair that can sign and encrypt.

Once GPG finishes generating an RSAv3 keypair, it will erroneously warn you that the key cannot be used for encryption.

```
public and secret key created and signed.
key marked as ultimately trusted.

pub  2048R/ADBD19AD 2002-05-20 Alice Wong <a-wong@big-corp.com>
     Key fingerprint = 19 BC 2D 4B 43 20 38 9D  9F 79 B8 AF 1A AA FF CC

Note that this key cannot be used for encryption.  You may want to use
the command "--edit-key" to generate a secondary key for this purpose.

D:\Programs\gnupg>
```

Ignore this warning. Not only can this single RSAv3 key be used to encrypt, but if you try to generate a subkey for it with the **--edit-key** and **addkey** commands, GPG will refuse to generate a subkey.

```
D:\Programs\gnupg>gpg --edit-key alice

gpg (GnuPG) 1.0.7; Copyright (C) 2002 Free Software Foundation, Inc.
This program comes with ABSOLUTELY NO WARRANTY.
This is free software, and you are welcome to redistribute it
under certain conditions. See the file COPYING for details.

Secret key is available.

gpg: checking the trustdb
gpg: checking at depth 0 signed=1 ot(-/q/n/m/f/u)=0/0/0/0/0/6
gpg: checking at depth 1 signed=0 ot(-/q/n/m/f/u)=1/0/0/0/0/0
pub  2048R/ADBD19AD  created: 2002-05-20 expires: never      trust: u/u
(1). Alice Wong <a-wong@big-corp.com>

Command> addkey

gpg: NOTE: creating subkeys for v3 keys is not OpenPGP compliant

Command>
```

RSAv3 keys can be useful if you're corresponding or working with PGP 2.6x users. If you don't need the compatibility of an RSAv3 key, though, you would be better off generating either an RSAv4 keypair with an encryption subkey (as we did just **above**) or a DSA/ElGamal keypair (also discussed **earlier**).

Working with Subkeys

If you've generated a keypair with an encryption subkey, you can manage the subkey *somewhat* independently of the master signing key. You can revoke (**revkey**) or remove (**delkey**) the encryption subkey from the master signing key and then add (**addkey**) a new encryption subkey. You can also set the expiration (**expire**) on a subkey. To work with encryption subkeys, use the **--edit-key** command, which allows you to use the **addkey**, **revkey**, **delkey**, and **expire** sub-commands.

In case you were wondering, it is possible to create multiple subkeys for a single master signing key (the second subkey will be a signing subkey). In fact, you can even mix key types, as DSA/ElGamal keypairs do. You can mix RSAv4, DSA, and ElGamal keys in a variety of different ways (e.g., a DSA master signing key with an ElGamal encryption subkey and an RSAv4 signing key; or an RSAv4 master signing key with an RSAv4 encryption key and a DSA signing key, et al). In such

cases, only one encryption key and one signing key will be used (with the last key added taking precedence). Moreover, keyservers may have problems handling such hybrid keys and end up mangling them. Unless you have a good reason to mix and match multiple subkeys in non-standard combinations, it's probably best to stick to the standard keypairs we've discussed here.

(For more information on working with keypairs and subkeys, see the **GNU Privacy Handbook's** discussion.of **adding and deleting subkeys**, **revoking subkeys**, and setting **expirations for subkeys**.)

Importing Keys (gpg [--options] --import *files*)

Before you can encrypt files and messages to send to other people, you must have their public keys on your keyring. You might receive copies of other people's public keys as .ASC or .GPG files in email or in person. You can also get people's public keys by downloading them from the Internet or a keyserver. Whatever the case, once you get someone's public key, you must import it onto your keyring with the **--import** command.

In this example, we've received a copy of someone's public key as an ASCII Armored file (prz.asc), which we can import onto our keyring

```
D:\Programs\gnupg>gpg --import d:\temp\prz.asc

gpg: key FAEBD5FC: public key imported
gpg: Total number processed: 1
gpg:               imported: 1

D:\Programs\gnupg>
```

If we want to confirm that this person's public key has been imported onto our keyring, we can use the **--list-keys** command, which is discussed **below**.

After you've imported someone's public key onto your keyring, you should consider **signing the key** and **setting the owner trust level**. Once a recipient's public key is on your keyring, you can use it to encrypt files and messages to that person. You can also use it to verify signatures on messages and files from that person.

To give your own public key to other people (so that they can encrypt files and messages to you and verify signatures from you), you'll need to export your public key, which is discussed in the **next section**.

GPG's **--import** option can also be used to import secret keys onto your keyring. For example, you might also want to import keys from your old **PGP keyrings** to GPG's keyrings. Some of those keys will undoubtedly be your own secret keys, and GPG will import them without a problem.

```
D:\Programs\gnupg>gpg --import d:\temp\my-sec.gpg

gpg: key 32E3B3DF: secret key imported
gpg: Total number processed: 1
gpg:       secret keys read: 1
gpg:   secret keys imported: 1

D:\Programs\gnupg>
```

In previous versions of GPG, it was necessary to use the **--allow-secret-key-import** option with the **--import** command when importing secret keys. That is no longer the case with GPG 1.0.7 (making the **--allow-secret-key-import** option largely obsolete). There are still a few issues with importing secret keys from PGP 2.6.x, however. See the next section for more information on **migrating your PGP keys** to GPG.

If you've imported your own public and secret keys onto your GPG keyrings, you'll probably want to set the owner trust and calculated trust levels to "ultimately trusted" (u/u) for those keys. Use the **--edit-key | trust** command and select "5 = I trust ultimately" when asked how much you trust the user "to correctly verify other users' keys." Setting the owner trust level to "ultimately trusted" will also set the calculated trust level to "ultimately trusted." See the **Setting Owner Trust** section below for a discussion of the **--edit-key | trust** command.

(For more information on importing keys, see the **GNU Privacy Handbook**.)

Migrating Keys from PGP

If you're moving to GPG from Pretty Good Privacy (PGP), then you'll undoubtedly have public and secret keys that you want to migrate from your PGP keyrings to your new GPG keyrings. Moving your keys from PGP to GPG is a simple, straightforward process.

- **Export** your keys from your PGP keyrings to key export files.
- **Import** those key export files onto your GPG keyrings with GPG's **--import** command.

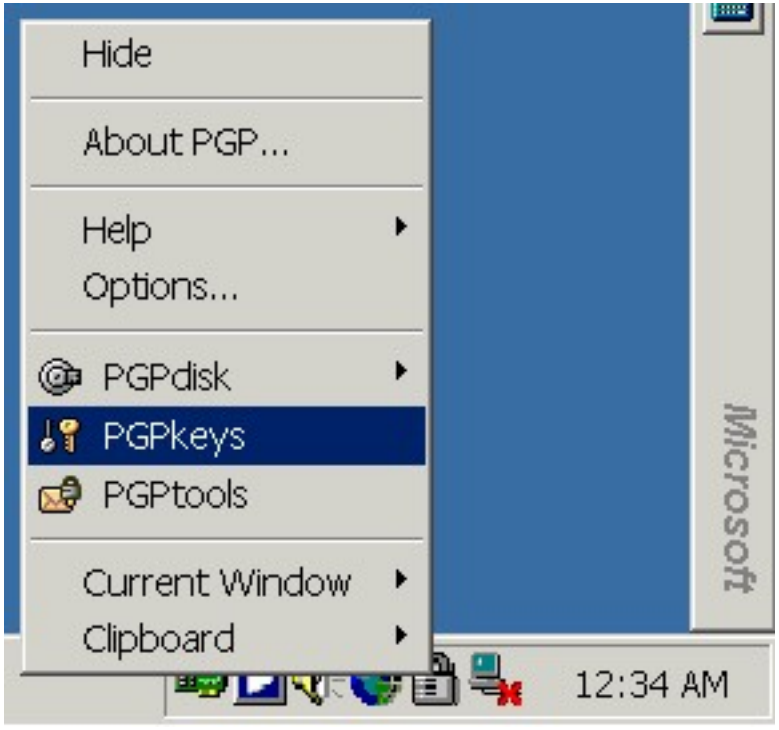
What follows are instructions for exporting (or extracting) keys from PGP's keyrings using PGP 6.x/7.x or PGP 2.6x.

Migrating Keys from PGP 6.x and 7.x

To export your keys from PGP 6.x or 7.x:

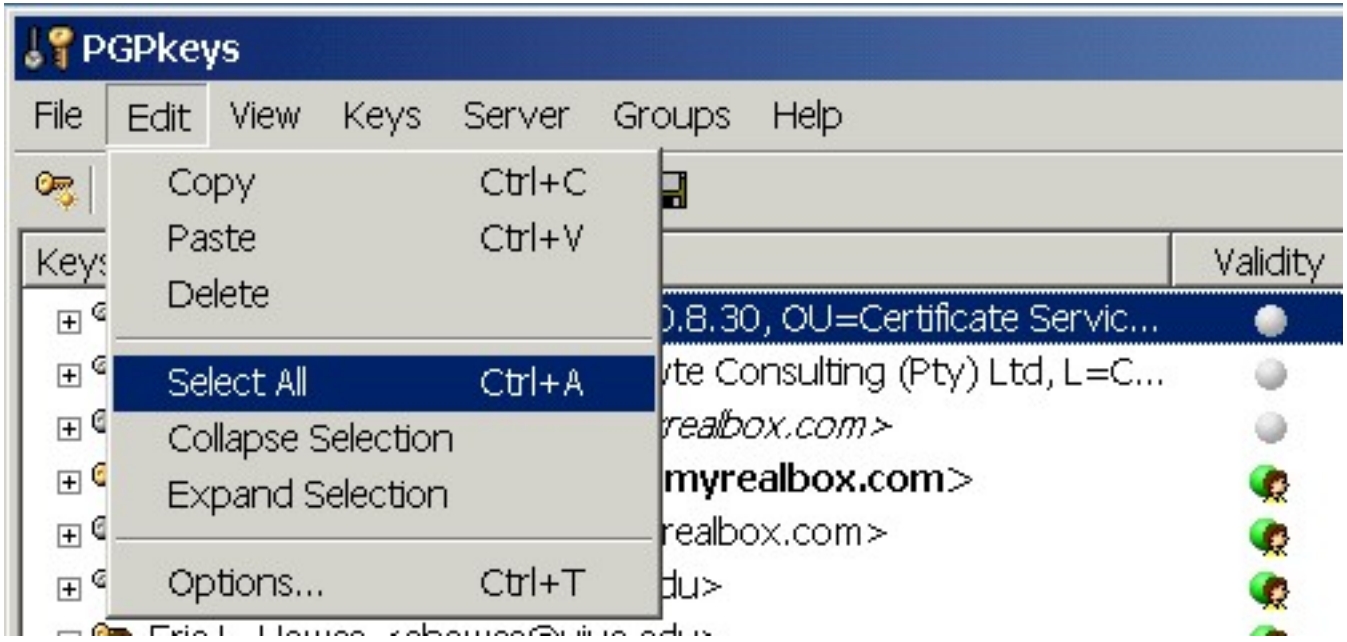
1. **Open PGPkeys**

   Open PGPkeys from the Start menu or the PGPtray tray icon.



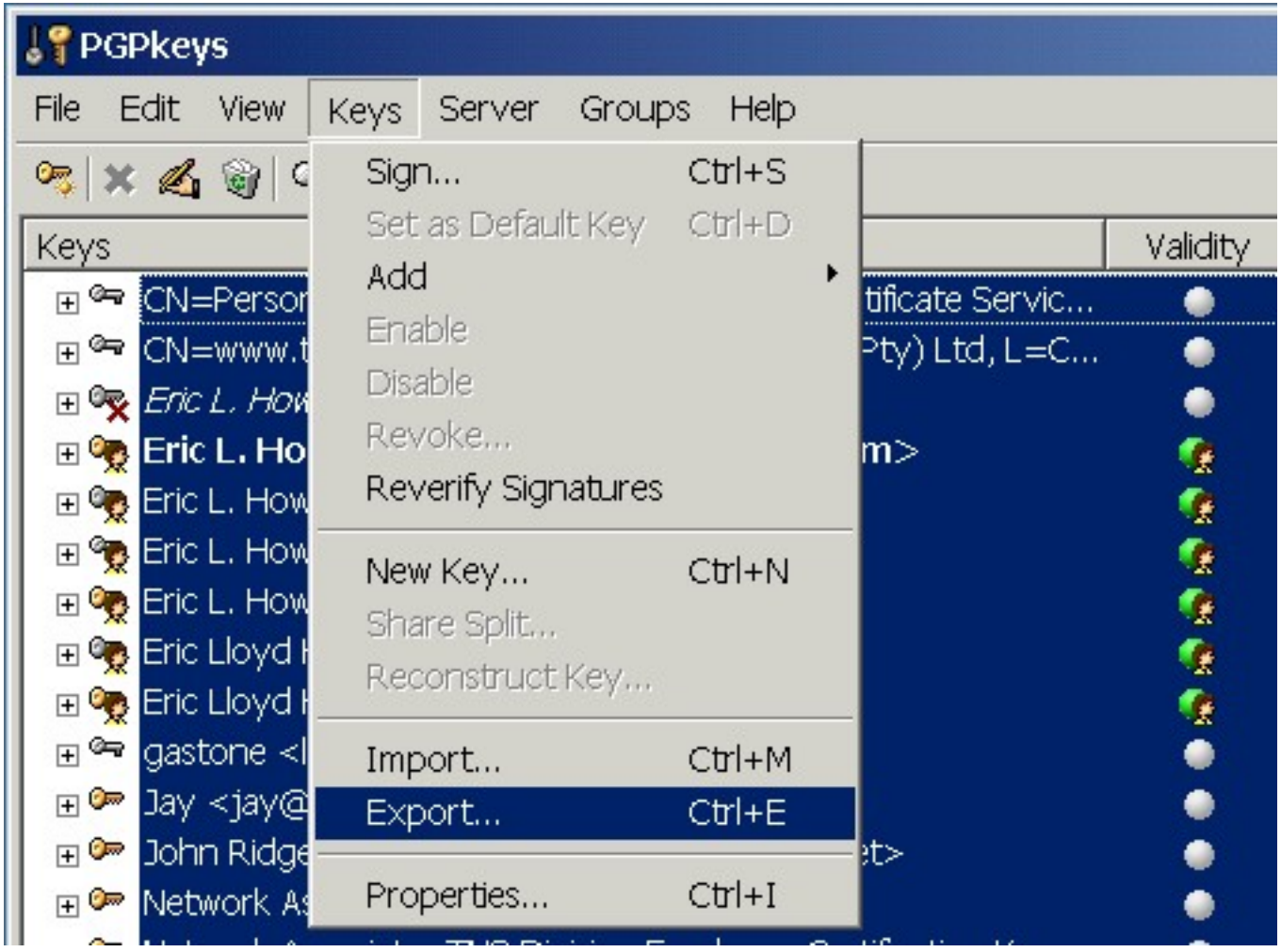
2. **Select All PGP Keys**

   From the PGPkeys menu bar, hit **Edit** >> **Select All**. All the keys on your keyring should be selected (highlighted).

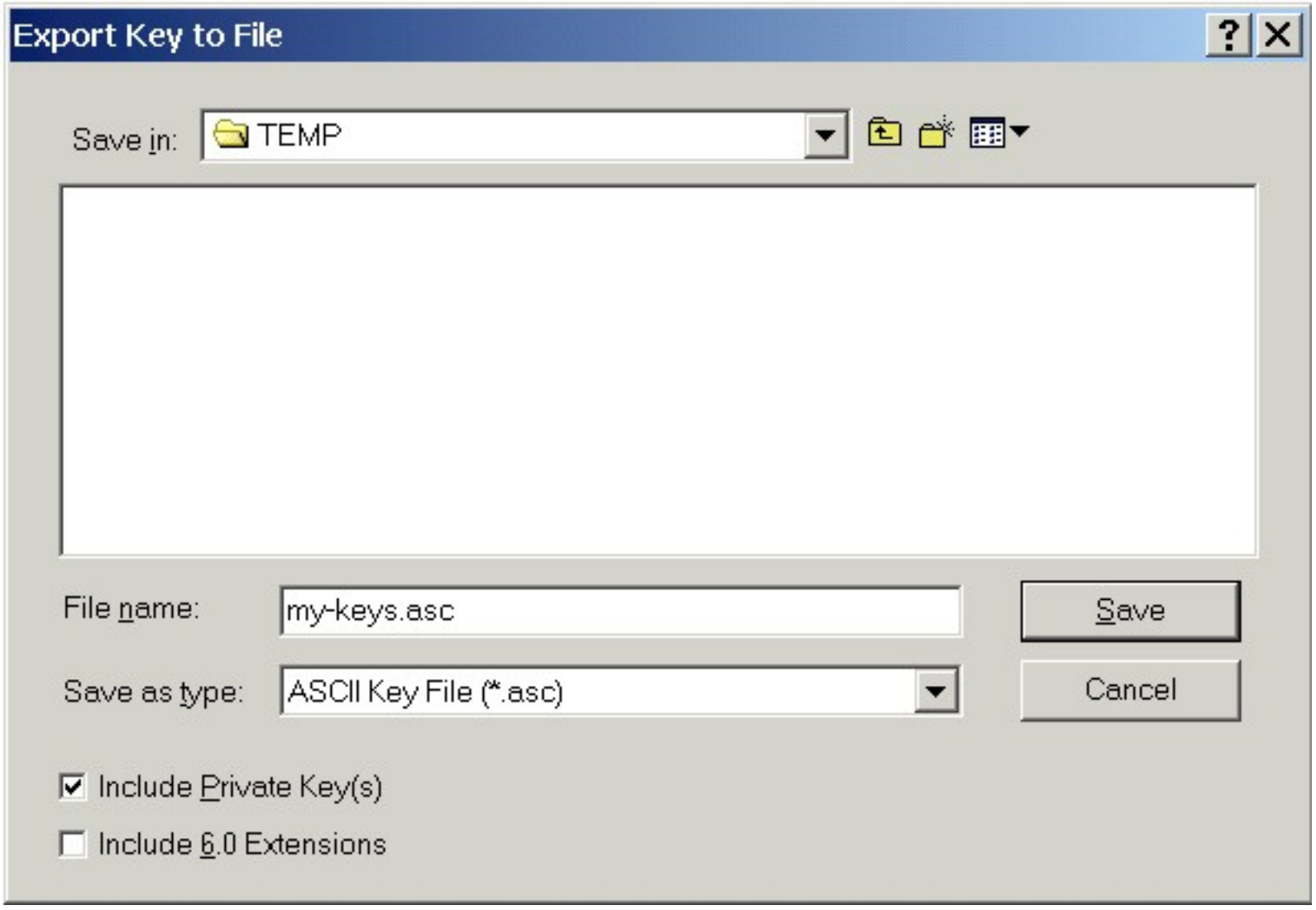


3. **Export PGP Keys**

From the PGPkeys menu bar, hit **Keys** >> **Export...**



PGPkeys will pop up a dialog box asking you where you want to save the exported keys.



Specify a directory and file name. Also, make sure you check the "**Include Private Key(s)**" box -- if you don't, your secret keys won't be exported from PGP (thus leaving you without the ability to decrypt files or make signatures with your keys in GPG). Then hit "**Save**" to export your keys. Note that PGP exports all your public and secret keys to a single file.

Once you've exported your PGP keys to a key export file, you can import that file with GPG's **--import** command. GPG will recognize and properly import all the separate keys from that common key export file.

If you've imported your own public and secret keys onto your GPG keyrings, you'll probably want to set the owner trust and calculated trust levels to "ultimately trusted" (u/u) for those keys. Use the **--edit-key | trust** command and select "5 = I trust ultimately" when asked how much you trust the user "to correctly verify other users' keys." Setting the owner trust level to "ultimately trusted" will also set the calculated trust level to "ultimately trusted." See the **Setting Owner Trust** section below for a discussion of the **--edit-key | trust** command.

Migrating Keys from PGP 2.6.x

To export keys from PGP 2.6.x (or other command line versions of PGP):

1. **Get a List of Public Keys to Export**

   Get a list of the public keys you want to export with PGP's **-kv** option:

```
D:\Programs\pgp2>pgp -kv

Pretty Good Privacy(tm) 2.6.3ia-multi06 - Public-key encryption for the masses
(c) 1990-96 Philip Zimmermann, Phil's Pretty Good Software. 2002-04-22
International version - for use everywhere (including USA).
Current time: 2002/05/22 04:51 GMT

PGP is now using IDEA with MD5.

Key ring: 'pubring.pgp'
Type Bits/KeyID    Date       User ID
pub  2048/32E3B3DF 2002/05/20 George P. Gumbel <gpgumbel@cowtownu.edu>
pub  2048/3FAD9F1E 2001/11/13 Bob Bone <bobbone@cowtownu.edu>
pub  4096/49B58839 2002/03/27 Alice Wong <a-wong@big-corp.com>
pub  1024/FAEBD5FC 1997/04/07 Philip R. Zimmermann <prz@pgp.com>
4 matching keys found.

D:\Programs\pgp2>
```

   Note the Key ID's or User ID's of the public keys you want to export. If you plan to use Key ID's to specify keys, remember that you'll have to add the 0x onto the front of the Key ID you see with PGP **-kv**. For example, Phil's Key ID is **0xFAEBD5FC**.

2. Self-Sign Your Own Keys

PGP 2.6.x keys are not self-signed by default, though most experts **strongly recommend** that users sign their own keys, and newer versions of PGP do automatically self-sign keys when they're created. (GPG also self-signs newly created keys automatically, as we saw in the **key generation section** above.) More to the point, however, GPG will not allow you to import keys that are not self-signed unless you use the **--allow-non-selfsigned-uid** option when importing secret keys onto GPG's keyrings.

To self-sign your own keys in PGP 2.6.x, use the **-ks** option and specify your own Key ID or User ID as both the key to be signed and the signing key to be used. For example, George could self-sign his own key with the following:

```
D:\Programs\pgp2>pgp -ks 0x32E3B3DF -u george

Pretty Good Privacy(tm) 2.6.3ia-multi06 - Public-key encryption for the masses
(c) 1990-96 Philip Zimmermann, Phil's Pretty Good Software. 2002-04-22
International version - for use everywhere (including USA).
Current time: 2002/05/26 05:05 GMT

PGP is now using IDEA with MD5.

Looking for key for user '32E3B3DF':

Key for user ID: George P. Gumbel <gpgumbel@cowtownu.edu>
2048-bit key, key ID 32E3B3DF, created 2002/03/27
            Key fingerprint = 448E D686 3AFC 8148 07E9 1DD3 329E D4B4 32E3 B3DF

READ CAREFULLY:  Based on your own direct first-hand knowledge, are
you absolutely certain that you are prepared to solemnly certify that
the above public key actually belongs to the user specified by the
above user ID (y/N)? Y

You need a pass phrase to unlock your RSA secret key.
Key for user ID: George P. Gumbel <gpgumbel@cowtownu.edu>
2048-bit key, key ID 32E3B3DF, created 2002/03/27

Enter pass phrase: My_31337_Passphrase
```

```
Pass phrase is good.  Just a moment....
Key signature certificate added.

Make a determination in your own mind whether this key actually
belongs to the person whom you think it belongs to, based on available
evidence.  If you think it does, then based on your estimate of
that person's integrity and competence in key management, answer
the following question:

Would you trust "George P. Gumbel <gpgumbel@cowtownu.edu>"
to act as an introducer and certify other people's public keys to you?
(1=I don't know. 2=No. 3=Usually. 4=Yes, always.) ? 4

D:\Programs\pgp2>
```

If you prefer not to self-sign your PGP 2.6.x keys, don't forget to use the **--allow-non-selfsigned-uid** option when importing your keys onto GPG's keyrings.

3. Export Public Keys

Export each public key (including your own) with PGP's **-kx** option -- be sure to specify the key you wish to export as well as a key export file :

```
D:\Programs\pgp2>pgp -kx george my-pub.pgp

Pretty Good Privacy(tm) 2.6.3ia-multi06 - Public-key encryption for the masses
(c) 1990-96 Philip Zimmermann, Phil's Pretty Good Software. 2002-04-22
International version - for use everywhere (including USA).
Current time: 2002/05/22 04:47 GMT

PGP is now using IDEA with MD5.

Extracting from key ring: 'pubring.pgp', userid "george".
```

```
Key for user ID: George P. Gumbel <gpgumbel@cowtownu.edu>
2048-bit key, key ID 32E3B3DF, created 2002/05/20

Key extracted to file 'my-pub.pgp'.

D:\Programs\pgp2>
```

You don't have to specify a new export file for each key. If you specify the same export file for more than one key, PGP will simply add new exported keys to the keys that are already in that common export file.

Note: do not armor the key export files with PGP 2.6.x's **-kxa** or **-a** options. The ASCII Armor format used by PGP2.6.x is incompatible with the newer OpenPGP format that GPG uses. If you armor the key export files, GPG won't be able to import them properly.

4. **Get a List of Secret Keys to Export**

Get a list of the secret keys you wish to export with the **-kv** option -- make sure to specify your PGP secret keyring :

```
D:\Programs\pgp2>pgp -kv secring.pgp

Pretty Good Privacy(tm) 2.6.3ia-multi06 - Public-key encryption for the masses
(c) 1990-96 Philip Zimmermann, Phil's Pretty Good Software. 2002-04-22
International version - for use everywhere (including USA).
Current time: 2002/05/22 04:54 GMT

PGP is now using IDEA with MD5.

Key ring: 'secring.pgp'
Type Bits/KeyID    Date       User ID
sec  2048/32E3B3DF 2002/05/20 George P. Gumbel <gpgumbel@cowtownu.edu>
1 matching key found.

D:\Programs\pgp2>
```

If you have multiple secret keys, note the Key ID's or User ID's of all the secret keys you want to export.

5. **Export Secret Keys**

Export your secret keys with the **-kx** option -- be sure to specify the key you wish to export, an export file, and your PGP secret keyring:

```
D:\Programs\pgp2>pgp -kx george my-sec.pgp secring.pgp

Pretty Good Privacy(tm) 2.6.3ia-multi06 - Public-key encryption for the masses
(c) 1990-96 Philip Zimmermann, Phil's Pretty Good Software. 2002-04-22
International version - for use everywhere (including USA).
Current time: 2002/05/22 04:47 GMT

PGP is now using IDEA with MD5.

Extracting from key ring: 'secring.pgp', userid "george".

Key for user ID: George P. Gumbel <gpgumbel@cowtownu.edu>
2048-bit key, key ID 32E3B3DF, created 2002/05/20

Key extracted to file 'my-sec.pgp'.

D:\Programs\pgp2>
```

Note that if you have multiple secret keys, you can export them to a common file, but that file must be *different* file than the one to which you exported your public keys.

Once you have exported all of your public and secret keys from PGP to key export files, import those files with GPG's **--import** command. If you have exported multiple keys to a common file, GPG will recognize and properly import all the separate keys in that common key export file. If you are importing PGP 2.6.x keys that are not self-signed, don't forget to use the **--allow-non-selfsigned-uid** option when importing those keys.

If you've imported your own public and secret keys onto your GPG keyrings, you'll probably want to set the owner trust and calculated trust levels to "ultimately trusted" (u/u) for those keys. Use the **--edit-key | trust** command and select "5 = I trust ultimately" when asked how much you trust the user "to correctly verify other users' keys." Setting the owner trust level to "ultimately trusted" will also set the calculated trust level to "ultimately trusted." See the **Setting Owner Trust** section

below for a discussion of the **--edit-key | trust** command.

Exporting Public Keys (gpg [--options] --export *names*)

Before other people can encrypt messages to you or verify signatures from you, they must have a copy of your public key. In order to distribute your public key -- either to a keyserver or to other people.directly -- you need to export your public key. Use the **--export** command to export your public key. Although you can export your public key as binary data to a .GPG file, you'll find it more useful to use the **--armor** option and export your public key as ASCII Armor.

```
D:\Programs\gnupg>gpg --armor --export gpgumbel@cowtownu.edu

-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v1.0.7 (MingW32)

mQELBDzhaTcBCACkJHW1vu3VeUlGEnaViY6WhFBhtmbkR1fqmsckkawrmNiRn5SW
+VfDgsY3dIDhqDkTDkflLRaafJf6rUkDOsDL6fetwgfRW61hNjOpiiqGYLTAM0pm
RfKkeNMMUEC5+RFSLRWxhTPFdsdKid88p2SpPUEe1g9o6HVJhUzNx1k9qErwldL3
SaSORQyX3uxLZD7x3nWsEAAChv4Ddm1nrFKofna9u+klmOBGedfzovPasbwTePpx
Xx3irJytkDv0mvjq8ptV/X8Q8MR6LS9brlH+R3ArPLKGpAeUf51tWipqT3Vx5DpR
WxBYmSTlyPY7EgQ6COfwhxLhcOeFQXbBrCAtAAYptBtFTCBIb3dlcyA8ZWhvd2Vz
QHlhaG9vLmNvbT6JATYEEwECACACGw8FCwcDAgEDFQIDAxYCAQIeAQIXgAIZAQUC
POFpOAAKCRAyntS0MeOz3pnpB/4nlZEwGR65iXeQixZoPsHzqZNdGWZgLbEzmbeQ
XxE5SbP5ZRRo39+JZmlOUTV0G9MF0/mXqEVDP2EPlQiZuUPd/iAqCzkZmoC9KjUX
bl6gGHLmj5nFTdw3zZprlbUhtcuBiVGnVP1sBPQkVqB5mjgEHu9rqeOWOjxqULmC
31+b52hCaosHCdwoGe6LRXyoZUueYoGZ6q7lkZqc1XfWrytjknll3RZQX4r6Ucim
tld9BXsezYTHjRggGQhGc62njW3+Z4TvNRWMB1UOEqBKGAbQ9nnhpyg0d72/9WN5
Go/oWRQ+wZ8JCQrMpfywLHA7t9p4zSXMDPYI60jBfByRXM8z
=p41P
-----END PGP PUBLIC KEY BLOCK-----

D:\Programs\gnupg>
```

You can also use the **--output** option to specify an output file so that you have an easily transportable file.

```
D:\Programs\gnupg>gpg --armor --output gumbel-pub.asc --export george
```

```
D:\Programs\gnupg>
```

If you open up that output file (gumbel-pub.asc), you'll see that the ASCII Armored public key block is easily usable in text email messages and other contexts that require plain text (say, posting on a web page).

```
D:\Programs\gnupg>type gumbel-pub.asc

-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v1.0.7 (MingW32)

mQELBDzhaTcBCACkJHW1vu3VeUlGEnaViY6WhFBhtmbkR1fqmsckkawrmNiRn5SW
+VfDgsY3dIDhqDkTDkflLRaafJf6rUkDOsDL6fetwgfRW61hNjOpiiqGYLTAM0pm
RfKkeNMMUEC5+RFSLRWxhTPFdsdKid88p2SpPUEe1g9o6HVJhUzNx1k9qErwldL3
SaSORQyX3uxLZD7x3nWsEAAChv4Ddm1nrFKofna9u+klmOBGedfzovPasbwTePpx
Xx3irJytkDv0mvjq8ptV/X8Q8MR6LS9brlH+R3ArPLKGpAeUf51tWipqT3Vx5DpR
WxBYmSTlyPY7EgQ6COfwhxLhcOeFQXbBrCAtAAYptBtFTCBIb3dlcyA8ZWhvd2Vz
QHlhaG9vLmNvbT6JATYEEwECACACGw8FCwcDAgEDFQIDAxYCAQIeAQIXgAIZAQUC
POFpOAAKCRAyntS0MeOz3pnpB/4nlZEwGR65iXeQixZoPsHzqZNdGWZgLbEzmbeQ
XxE5SbP5ZRRo39+JZmlOUTV0G9MF0/mXqEVDP2EPlQiZuUPd/iAqCzkZmoC9KjUX
bl6gGHLmj5nFTdw3zZprlbUhtcuBiVGnVP1sBPQkVqB5mjgEHu9rqeOWOjxqULmC
31+b52hCaosHCdwoGe6LRXyoZUueYoGZ6q7lkZqc1XfWrytjknll3RZQX4r6Ucim
tld9BXsezYTHjRggGQhGc62njW3+Z4TvNRWMB1UOEqBKGAbQ9nnhpyg0d72/9WN5
Go/oWRQ+wZ8JCQrMpfywLHA7t9p4zSXMDPYI60jBfByRXM8z
=p41P
-----END PGP PUBLIC KEY BLOCK-----

D:\Programs\gnupg>
```

This ASCII Armored file containing your public key can be distributed to others so that they can import your public key onto their keyrings. Once your public key is on their keyrings, they can use it to send you encrypted email and verify signatures that you make with your secret key.

(For more information on exporting public keys, see the **GNU Privacy Handbook**.)

Exporting Secret Keys (gpg [--options] --export-secret-keys *names*)

You can export secret keys as well as public keys. You should exercise great caution in exporting your secret keys, though. Once exported, secret keys should be guarded as zealously as your keyrings. Use the **--export-secret-keys** command combined with the **--armor** and **--output** options to export secret keys to ASCII Armored files.

```
D:\Programs\gnupg>gpg --armor --output gumbel-sec.asc --export-secret-keys george

D:\Programs\gnupg>
```

The output file (gumbel-sec.asc) consists of a private key block (private key being another name for secret key).

```
D:\Programs\gnupg>type gumbel-sec.asc

-----BEGIN PGP PRIVATE KEY BLOCK-----
Version: GnuPG v1.0.7 (MingW32)

lQO8BDzhaTcBCACkJHW1vu3VeUlGEnaViY6WhFBhtmbkR1fqmsckkawrmNiRn5SW
+VfDgsY3dIDhqDkTDkflLRaafJf6rUkDOsDL6fetwgfRW61hNjOpiiqGYLTAM0pm
RfKkeNMMUEC5+RFSLRWxhTPFdsdKid88p2SpPUEe1g9o6HVJhUzNx1k9qErwldL3
SaSORQyX3uxLZD7x3nWsEAAChv4Ddm1nrFKofna9u+klmOBGedfzovPasbwTePpx
Xx3irJytkDv0mvjq8ptV/X8Q8MR6LS9brlH+R3ArPLKGpAeUf51tWipqT3Vx5DpR
WxBYmSTlyPY7EgQ6COfwhxLhcOeFQXbBrCAtAAYp/gMDAk396O6ttFVdYDwrMZlP
9G94x4SuFWeqC1aSEV/0V02ONViTRWbFQyITApkCL2m/PhjJpzUFbyGu4EL0L80k
1CfupordgIp4u0exD/OjRMEPLO2LS6HxAscrkadWl1DtzMhNJGrCI2ID+FYymQNE
qu1odPkRUVPHc0jltc8gIBE007OrIjRtgc/1wRIybfNUeihGEn2eADoHntrY/Sts
UvL4Y6IA5MebShqM7ipN9wmQ4z8DC92pGc6Ipn8FQD9ERZOxwrTw9SEab4Nly+Qw
c61iNl53A3QQDgyYdDYGAR3ITrcrEX8ZhAUDqbsUl7kaXydpY00dMwQktauPHCwO
pJ3TJkKMAisvM1XR58Ej70mwDRFsCdH7kJTmIr85RVzx97OM94VBbCZyt4tyh6Jx
syO0Vo8KV4uAxgnV61/lspBWcNqedUkM1VE7DF2J08kb9mY1KjkuLsLw4B0eFP2w
q802N82qcc0EzgWskoFrkdMilRlXjHFkmZlF5OS4CeYKAYA+hfOeuIRnxcBT8y4S
U3+86byCKqv4B5UlZtnPNyGEXArnqPLl2Z5pRdQ9/KMn3XW5GkULB/z+z/KhFnVQ
4/85NHM5Dyc8wa/0NysLv2AvhdALRVKmyaL8hUxJa/yDgyXlMsSslP5ICFASouLw
g/iq5XWHkk/byQCkxE4kHBuyPl4XbFfMgykTmhqHVT+T8SRwhA1o1tkCRkmeuXe5
Psin/haB4Y9/4PsNN9VsVfL+0qIn5VZBNA2IKqqpG0YzlPrYWIMJpgV9BUKLbdGW
uI1h2fKLve/dEJAY8S0VB21fi+HdHkx62hJez6NdU9F0ULosizmlXMhKDJpuo9EH
R6hwkxOl6xsNuDkjCO9UNAsXAC0vxkKmvC7yQ+TovySKSQui9Dtnth1fKmGw1aC0
G0VMIEhvd2VzIDxlaG93ZXNAeWFob28uY29tPokBMwQTAQIAHQUCPOFpNwIbDwUL
```

```
BwMCAQMVAgMDFgIBAh4BAheAAAoJEDKe1LQx47PeMlcH/3Pl8gBSiykWFhcZuLHw
peaSY40+3USuN911Y90Cx6/O4kLrRJyLAA7hebUb9hZESjaAVln48/qJhRvVry8m
1yTtIDgwPo7Nw/94EydRhuewlkDm7n/Z9FHuHG1o8j567LfoBZO0hojTgEhsARr0
kUgGOx3+Pvr/y39yEFBRIM6yemlJB5dT7nChuY0w3rd8D0l4nmXTYRHw17bC5zI2
9Ah0B/djkZYyXZcVpy2Kz92B0bpQYqZKSEJR27+uxjNpkJsM7B4k9SNATc6K8UiF
Wvbe3Ei2SrrzIoaXK7zJ13bhGES5QYqCuRszk/f1BgdZ3LK5LwTPYc0hO1w+CR6R
X4I=
=78n5
-----END PGP PRIVATE KEY BLOCK-----

D:\Programs\gnupg>
```

Think hard about exporting your secret keys and have a plan for guarding them before you do export them.

Exporting Secret Keys for Other Platforms

One reason you might want to export your secret key is so that you can import it and use it in PGP or a different installation of GPG. If you're running GPG 1.0.7 and plan to export one of your secret keys to import and use in PGP or an *earlier* version of GPG, you'll first need to convert your secret keys to a format that those older programs recognize. To perform that conversion, use the **--simple-sk-checksum** option in conjunction with the **--edit-key | passwd** command. You should convert your secret keys to the older format *before* exporting them.

GPG 1.0.7 protects the integrity of its secret keys with a 20-byte SHA1 hash, a format which PGP (including the last version from NAI, PGP Corporate Desktop 7.1.1) and older GPG versions don't recognize (PGP and older versions of GPG use a simple 16-bit checksum).***** If you simply export one of your secret keys from GPG 1.0.7 and then import it into PGP, for example, the key will be unusable. When you attempt to decrypt files and messages with the secret key that you imported, PGP will not recognize the passphrase (or let you change the passphrase) and thus will refuse to decrypt with the secret key.

To avoid this problem, change the passphrase on your secret keys by using the **--simple-sk-checksum** option in conjunction with the **--edit-key | passwd** command *before* you export your secret keys. This converts the key integrity check from the newer 20-byte SHA1 hash format (which only GPG 1.0.7 recognizes) to the older 16-bit checksum format recognized by PGP and earlier versions of GPG.

Here's how to do it. First edit the key (**--edit-key**) with the **--simple-sk-checksum** option.

```
D:\Programs\gnupg>gpg --simple-sk-checksum --edit-key george

gpg (GnuPG) 1.0.7; Copyright (C) 2002 Free Software Foundation, Inc.
This program comes with ABSOLUTELY NO WARRANTY.
This is free software, and you are welcome to redistribute it
under certain conditions. See the file COPYING for details.

Secret key is available.

pub  1024D/0A484ECB  created: 2002-05-26 expires: never      trust: u/u
sub  2048g/C31174A2  created: 2002-05-26 expires: never
(1). George P. Gumbel <gpgumbel@cowtownu.edu>

Command>
```

Next, change the passphrase on your secret key with the **passwd** command. You'll have to enter the current passphrase before you can change it.

```
Command> passwd

Key is protected.

You need a passphrase to unlock the secret key for
user: "George P. Gumbel <gpgumbel@cowtownu.edu>"
1024-bit DSA key, ID 0A484ECB, created 2002-05-26

Enter passphrase: My_31337_Passphrase

Enter the new passphrase for this secret key.

Enter passphrase: My_31337_Passphrase
Repeat passphrase: My_31337_Passphrase

gpg: generating the deprecated 16-bit checksum for secret key protection
gpg: generating the deprecated 16-bit checksum for secret key protection

Command> save
```

```
D:\Programs\gnupg>
```

Note that you don't actually have to choose a new passphrase -- you can simply use the same passphrase as before. Once you enter a "new" passphrase, GPG generates the simple 16-bit checksum that PGP and older versions of GPG recognize. Make sure to **save** your changes. Now you can export your secret key with the **--export-secret-keys** command and import that secret key onto your keyring for PGP or an older version of GPG.

To change your secret key's integrity check back to the newer SHA1 hash format (which provides protection against certain kinds of attacks), simply re-edit your key (**--edit-key**) *without* the **--simple-sk-checksum** option and change the passphrase with the **passwd** command. Once you **save** your changes, GPG will restore the format of the secret key integrity check to the newer SHA1 hash format.

It is possible to use the **--simple-sk-checksum** option in the Options file. With **--simple-sk-checksum** in the Options file, any new keys generated with GPG will use the older key integrity format. Keys that were generated and placed on your keyring before **--simple-sk-checksum** was added to the Options file must still be converted with the **--edit-key | passwd** command, as described above (though you won't need to use the **--simple-sk-checksum** option with the **--edit-key** command since that option is already in the Options file).

**** Note:*** **PGP 6.5.8ckt build 09 betas 1-3** do recognize the newer SHA1 format. Also, they will convert from the old format to the new, but not convert from the new to the old format like GPG 1.0.7. CKT builds of PGP are enhanced versions of PGP 6.5.8 that include support for very large keys and a wider range of encryption and hash algorithms. See **Imad's PGP Page** for more details. And for more information on the new 20 byte SHA1 secret key hash and how it affects PGP and GnuPG usage, see **THIS** page.

Listing Public Keys **(gpg [--options] --list-keys)**

You can see the public keys that are on your keyring with the **--list-keys** command.

```
D:\Programs\gnupg>gpg --list-keys

D:/Programs/GnuPG/pubring.gpg
------------------------------
pub  1024D/0A484ECB 2002-05-26 George P. Gumbel <gpgumbel@cowtownu.edu>
sub  2048g/C31174A2 2002-05-26
```

```
pub  2048R/ADBD19AD 2002-05-20 Alice Wong <a-wong@big-corp.com>

pub  1024D/FAEBD5FC 1997-04-07 Philip R. Zimmermann <prz@pgp.com>
sub  2048g/42F0A0A0 1997-04-07


D:\Programs\gnupg>
```

GPG tells us that both keys are public keys (pub). For each key GPG also tells us the key length (2048 or 1024), the key type (R for RSA, D for DSA, g for ElGamal), the Key IDs (0A484ECB, ADBD19AD, and FAEBD5FC), as well as the creation dates and the User IDs. Finally, GPG lets us know that George and Phil's keys have subkeys (sub), both of which are 2048 bit ElGamal (g) keys. (For a discussion of subkeys, see the **Note on Subkeys** in the **Key Generation** section above.)

Remember that the **--list-keys** command displays only the public keys on your keyring, not any secret keys for your own keypairs. To view a list of secret keys on your keyring, use the **--list-secret-keys** command, which is discussed in the **next section**.

(For more information on listing keys, see the **GNU Privacy Handbook**.)

Listing Secret Keys (gpg [--options] --list-secret-keys)

You can see the secret keys on your keyring with the **--list-secret-keys** command..

```
D:\Programs\gnupg>gpg --list-secret-keys

D:/Programs/GnuPG/secring.gpg
------------------------------
sec  1024D/0A484ECB 2002-05-26 George P. Gumbel <gpgumbel@cowtownu.edu>
ssb  2048g/C31174A2 2002-05-26

D:\Programs\gnupg>
```

GPG provides the same type of information for your secret keys as for the public keys on your keyring.

Editing Keys (Basic) (gpg [--options] --edit-key *name*)

While GPG can use a wide range of commands to work with the keys on your keyring, one of the more versatile and powerful commands is the **--edit-key** command..

```
D:\Programs\gnupg>gpg --edit-key phil

gpg (GnuPG) 1.0.7; Copyright (C) 2002 Free Software Foundation, Inc.
This program comes with ABSOLUTELY NO WARRANTY.
This is free software, and you are welcome to redistribute it
under certain conditions. See the file COPYING for details.

gpg: checking the trustdb
gpg: checking at depth 0 signed=0 ot(-/q/n/m/f/u)=0/0/0/0/0/1
pub  1024D/FAEBD5FC  created: 1997-04-07 expires: never      trust: -/-
sub  2048g/42F0A0A0  created: 1997-04-07 expires: never
(1). Philip R. Zimmermann <prz@pgp.com>

Command>
```

When you use the **--edit-key** command on a particular key, GPG first displays basic information about that key (which is a public key with a **subkey** in this example) and then gives you a special command line. From this command line you can use particular sub-commands from the **edit-key menu**. We'll look at a few of them here, though you should consult the command reference for a **full list** of the sub-commands that can be used with the **--edit-key** command. In fact, many of the commands covered on this page of examples use the **--edit-key** command.

You'll notice, in the example above, that GPG gives us much the same information about a key as we received with the **--list-keys** command.

The **--edit-key** command also gives us information about signatures on the key (none at this point) as well as the trust levels assigned or associated with this key: "Trust: -/-" means that the user has not assigned a trust level to the owner of this key and that the overall trust level associated with this key is none. To change the trust levels and make this key trusted, we'll have to **sign the key** and set an **owner trust level**. (See the **command reference** for the **--editkey** command and the **GNU Privacy Handbook** for a more complete explanation of GPG and the Web of Trust.)

You can get more information about the key owner's preferences specified on the key with the **showpref** command..

```
Command> showpref
```

```
pub  1024D/FAEBD5FC  created: 1997-04-07 expires: never      trust: -/-
(1). Philip R. Zimmermann <prz@pgp.com>
     Cipher: CAST5, IDEA, 3DES
     Hash:
     Compression:

Command>
```

Note that we didn't use the leading dashes (--) with this command since we're working at the **--edit-key** command line. There is also a **pref** command, but the **showpref** command provides information in a slightly friendlier format. (For more information on key preferences and using the **pref** and **showpref** commands, see the section **Setting Preferences** below.)

If the key you edit is part of a key pair (meaning that you have the secret key), GPG lets you know that a secret key is available.

```
D:\Programs\gnupg>gpg --edit-key gpgumbel@cowtownu.edu

gpg (GnuPG) 1.0.7; Copyright (C) 2002 Free Software Foundation, Inc.
This program comes with ABSOLUTELY NO WARRANTY.
This is free software, and you are welcome to redistribute it
under certain conditions. See the file COPYING for details.

Secret key is available.

pub  1024D/0A484ECB  created: 2002-05-26 expires: never      trust: u/u
sub  2048g/C31174A2  created: 2002-05-26 expires: never
(1). George P. Gumbel <gpgumbel@cowtownu.edu>

Command> showpref

pub  1024D/0A484ECB  created: 2002-05-26 expires: never      trust: u/u
(1). George P. Gumbel <gpgumbel@cowtownu.edu>
     Cipher: AES, CAST5, 3DES, IDEA
     Hash: SHA1, RIPEMD160
     Compression: ZLIB, ZIP
```

```
Command>
```

Right now, though, we're looking at the public key (GPG displays the public key first with the **--edit-key** command).

To view the secret key (sec), use the **toggle** command..

```
Command> toggle

sec  1024D/0A484ECB  created: 2002-05-26 expires: never
ssb  2048g/C31174A2  created: 2002-05-26 expires: never
(1)  George P. Gumbel <gpgumbel@cowtownu.edu>

Command> quit

D:\Programs\gnupg>
```

You can revert to viewing the public key by using the **toggle** command again. To exit the **--edit-key** menu and return to a normal command line, use the **quit** command. If you've made changes to any of your keys, use the **save** command to save your changes and exit back to a normal command line. (If you make changes to your key but merely **quit**, GPG will prompt you to **save** your changes.)

(For more information on editing keys, see the **GNU Privacy Handbook**.)

Signing Keys (gpg [--options] --sign-key *name*)

One of the more important things you will do with keys is sign them. You'll recall that when you generated your own keypair, GPG used your secret key (or private key) to sign your public key (an act known as self-signing). You can use your secret key to sign other people's public keys as well. Signing other people's keys is important because it certifies those keys as trusted and integrates them into the Web of Trust used by GPG and PGP.

When you sign and certify someone else's public key, you are making a statement about your confidence that the public key you're signing actually belongs to the person specified in the User ID for that key. By signing someone's public key, you are building the Web of Trust for the keys on your own keyring and contributing to the Web of Trust for the larger community GPG and PGP users. Until you sign someone's public key and change the trust level associated with that key, GPG will warn you that the key is untrusted whenever you use that key to verify signatures or encrypt files and messages (see the note on ***Understanding Signatures & Trust*** above for a discussion of this warning). (For more information on signing keys

and using the Web of Trust, see the **GNU Privacy Handbook**.)

Before signing someone's key, you should validate it. Validating a key means that you have checked with the owner of the key (identified in the User ID) and verified that the key that you have is in fact that person's key. Ideally, you would contact the key owner and check the key fingerprint on the key you have against the key fingerprint the owner has. (A key's fingerprint is preferable to the Key ID, even the long version of the Key ID, as there is the remote chance the multiple keys can have the same Key ID.) You can get the key fingerprint either by using the **--fingerprint** command...

```
D:\Programs\gnupg>gpg --fingerprint phil

pub  1024D/FAEBD5FC 1997-04-07 Philip R. Zimmermann <prz@pgp.com>
     Key fingerprint = 17AF BAAF 2106 4E51 3F03  7E6E 63CB 691D FAEB D5FC
sub  2048g/42F0A0A0 1997-04-07

D:\Programs\gnupg>
```

...or by editing the key (**--edit-key**) and using the **fpr** command.

```
D:\Programs\gnupg>gpg --edit-key phil

gpg (GnuPG) 1.0.7; Copyright (C) 2002 Free Software Foundation, Inc.
This program comes with ABSOLUTELY NO WARRANTY.
This is free software, and you are welcome to redistribute it
under certain conditions. See the file COPYING for details.

pub  1024D/FAEBD5FC  created: 1997-04-07 expires: never      trust: -/-
sub  2048g/42F0A0A0  created: 1997-04-07 expires: never
(1). Philip R. Zimmermann <prz@pgp.com>

Command> fpr

pub  1024D/FAEBD5FC 1997-04-07 Philip R. Zimmermann <prz@pgp.com>
             Fingerprint: 17AF BAAF 2106 4E51 3F03  7E6E 63CB 691D FAEB D5FC

D:\Programs\gnupg>
```

Once you have validated the key, you can sign it. To sign someone's public key on your keyring, use the **--sign-key** command and specify the key you want to sign. GPG will ask you how carefully you have checked the identity of the person specified in the key's User ID.

```
D:\Programs\gnupg>gpg --sign-key phil

pub  1024D/FAEBD5FC  created: 1997-04-07 expires: never      trust: -/-
sub  2048g/42F0A0A0  created: 1997-04-07 expires: never
(1). Philip R. Zimmermann <prz@pgp.com>

pub  1024D/FAEBD5FC  created: 1997-04-07 expires: never      trust: -/-
     Fingerprint: 17AF BAAF 2106 4E51 3F03  7E6E 63CB 691D FAEB D5FC

Philip R. Zimmermann <prz@pgp.com>

How carefully have you verified the key you are about to sign actually belongs
to the person named above?  If you don't know what to answer, enter "0".

   (0) I will not answer. (default)
   (1) I have not checked at all.
   (2) I have done casual checking.
   (3) I have done very careful checking.

Your selection? 2
Are you really sure that you want to sign this key
with your key: "George P. Gumbel <gpgumbel@cowtownu.edu>"

I have checked this key casually.

Really sign? Y
```

If you have multiple secret keys of your own that you can use to sign others' keys, either use the **--local-user** option to indicate which secret key you wish to sign with, or specify the **default-key** option in your Options file.

As you are signing with your secret key, GPG will ask you for the passphrase for your secret key.

```
You need a passphrase to unlock the secret key for
user: "George P. Gumbel <gpgumbel@cowtownu.edu>"
1024-bit DSA key, ID 0A484ECB, created 2002-05-26

Enter passphrase: My_31337_Passphrase

D:\Programs\gnupg>
```

You can also sign keys by using the **--edit-key** command (introduced **earlier**) and the **sign** command from the **--edit-key** command line..

```
D:\Programs\gnupg>gpg --edit-key phil

gpg (GnuPG) 1.0.7; Copyright (C) 2002 Free Software Foundation, Inc.
This program comes with ABSOLUTELY NO WARRANTY.
This is free software, and you are welcome to redistribute it
under certain conditions. See the file COPYING for details.

pub  1024D/FAEBD5FC  created: 1997-04-07 expires: never      trust: -/-
sub  2048g/42F0A0A0  created: 1997-04-07 expires: never
(1). Philip R. Zimmermann <prz@pgp.com>

Command> sign

pub  1024D/FAEBD5FC  created: 1997-04-07 expires: never      trust: -/-
     Fingerprint: 17AF BAAF 2106 4E51 3F03  7E6E 63CB 691D FAEB D5FC

     Philip R. Zimmermann <prz@pgp.com>

How carefully have you verified the key you are about to sign actually belongs
to the person named above?  If you don't know what to answer, enter "0".

   (0) I will not answer. (default)
   (1) I have not checked at all.
   (2) I have done casual checking.
   (3) I have done very careful checking.
```

```
Your selection? 2

Are you really sure that you want to sign this key
with your key: "George P. Gumbel <gpgumbel@cowtownu.edu>"

I have checked this key casually.

Really sign? Y

You need a passphrase to unlock the secret key for
user: "George P. Gumbel <gpgumbel@cowtownu.edu>"
1024-bit DSA key, ID 0A484ECB, created 2002-05-26

Enter passphrase: My_31337_Passphrase

Command> quit
Save changes? Y

D:\Programs\gnupg>
```

If you chose option (2) or (3) when asked about how carefully you checked the identity of the person in the User ID, the calculated **level of trust** changes from none to something else. GPG prompts you to save your changes when you **quit**. (You could also **save** in order to exit back to the command line.)

You can check the level of trust by using the **--edit-key** command.

```
D:\Programs\gnupg>gpg --edit-key phil

gpg (GnuPG) 1.0.7; Copyright (C) 2002 Free Software Foundation, Inc.
This program comes with ABSOLUTELY NO WARRANTY.
This is free software, and you are welcome to redistribute it
under certain conditions. See the file COPYING for details.

pub  1024D/FAEBD5FC  created: 1997-04-07 expires: never      trust: -/f
sub  2048g/42F0A0A0  created: 1997-04-07 expires: never
(1). Philip R. Zimmermann <prz@pgp.com>
```

```
Command>
```

Notice that the calculated **level of trust** (the right-hand value) has changed from none (-) to full (f). To set the other trust level -- the owner trust level -- you'll need to use the **trust** command. See the **Setting Trust** section below for more details.

Keep in mind that the secret key you use to sign other people's keys should be a key that is itself "ultimately trusted" (u/u). (GPG automatically marks keys that you generate within GPG as "ultimately trusted.") If you sign someone else's key with a secret key that is not ultimately trusted, the trust level on the key you sign won't change. You can't build a Web of Trust around key that is not ultimately trusted.

(For more information on managing and building your Web of Trust, see the **GNU Privacy Handbook**.)

Listing Signatures (gpg [--options] --list-sigs *names*)

You can view a list of the signatures on public keys with the **--list-sigs** command.

```
D:\Programs\gnupg>gpg --list-sigs phil

pub  1024D/FAEBD5FC 1997-04-07 Philip R. Zimmermann <prz@pgp.com>
sig         FAEBD5FC 1997-04-07   Philip R. Zimmermann <prz@pgp.com>
sig         C7A966DD 1997-04-07   [User id not found]
sig         0525419B 1997-06-04   [User id not found]
sig         D027A0A0 1997-06-04   [User id not found]
sig         413E3D33 1997-06-01   [User id not found]
sig         E2CA6F8B 1997-05-30   [User id not found]
sig         CF73EC4C 1997-06-04   [User id not found]
sig         7C5AABF1 1997-06-01   [User id not found]
sig 3       0A484ECB 2002-05-26   George P. Gumbel <gpgumbel@cowtownu.edu>
sub  2048g/42F0A0A0 1997-04-07
sig         FAEBD5FC 1997-04-07   Philip R. Zimmermann <prz@pgp.com>


D:\Programs\gnupg>
```

This is the same key that we signed **just above**. There are many signatures from other PGP and GPG users on Phil's public key, but GPG can't give us the User IDs for many of the keys used to sign Phil's key ("*User id not found*") because

we don't have those users' keys on our keyring at this point. (We could use the Key IDs that are listed here and track down those keys on a keyserver, though.)

Contrast the signatures on Phil's public key with the signatures on our own public key.

```
D:\TEMP>gpg --list-sigs gpgumbel@cowtownu.edu

pub  1024D/0A484ECB 2002-05-26 George P. Gumbel <gpgumbel@cowtownu.edu>
sig 3       0A484ECB 2002-05-26   George P. Gumbel <gpgumbel@cowtownu.edu>
sub  2048g/C31174A2 2002-05-26
sig         0A484ECB 2002-05-26   George P. Gumbel <gpgumbel@cowtownu.edu>

D:\TEMP>
```

While Phil's public key has a wealth of signatures, our public key has only one signature (our own self-signature). Phil's key will be inherently more trusted in the larger PGP and GPG community -- at least until we can get others to sign our public key.

For *our own level of trust*, however, Phil's key is *just as trustworthy* as our own. Why? Because we don't have the other keys used to sign Phil's key on our keyring; thus we can't assess the trustworthiness of those other signatures. In effect, those other signatures are next to meaningless for our own level of trust. The only signature on Phil's key that we know and trust is the one we made ourselves. As noted earlier, though, we could track down the other keys used to sign Phil's key and start building a Web of Trust around Phil's key on our own keyring.

(For more information on signed keys and the Web of Trust, see the **GNU Privacy Handbook**.)

Checking Signatures (gpg [--options] --check-sigs *names*)

You can view a list of the signatures on public keys and verify those signatures with the **--check-sigs** command.

```
D:\Programs\gnupg>gpg --check-sigs phil

pub  1024D/FAEBD5FC 1997-04-07 Philip R. Zimmermann <prz@pgp.com>
sig!        FAEBD5FC 1997-04-07   Philip R. Zimmermann <prz@pgp.com>
sig!3       0A484ECB 2002-05-26   George P. Gumbel <gpgumbel@cowtownu.edu>
sub  2048g/42F0A0A0 1997-04-07
```

```
sig!        FAEBD5FC 1997-04-07   Philip R. Zimmermann <prz@pgp.com>

7 signatures not checked due to missing keys

D:\Programs\gnupg>
```

The exclamation point (!) next to each signature indicates that GPG has verified that signature against the key used to generate the signature. Notice that GPG could not verify seven signatures because we don't have the corresponding public keys on our keyring. (We could locate and download those missing keys from a keyserver, however, and re-check all of the signatures on Phil's key.)

You can also verify signatures by editing the key (**--edit-key**) and using the **check** command.

```
D:\TEMP>gpg --edit-key phil

gpg (GnuPG) 1.0.7; Copyright (C) 2002 Free Software Foundation, Inc.
This program comes with ABSOLUTELY NO WARRANTY.
This is free software, and you are welcome to redistribute it
under certain conditions. See the file COPYING for details.

pub  1024D/FAEBD5FC  created: 1997-04-07 expires: never      trust: -/f
sub  2048g/42F0A0A0  created: 1997-04-07 expires: never
(1). Philip R. Zimmermann <prz@pgp.com>

Command> check

uid  Philip R. Zimmermann <prz@pgp.com>
pub  1024D/FAEBD5FC 1997-04-07 Philip R. Zimmermann <prz@pgp.com>
sig!        FAEBD5FC 1997-04-07   [self-signature]
sig!3       0A484ECB 2002-05-26   George P. Gumbel <gpgumbel@cowtownu.edu>

7 signatures not checked due to missing keys

Command>
```

The results are the same as the **--check-sigs** command.

(For more information on signed keys and the Web of Trust, see the **GNU Privacy Handbook**.)

Setting Owner Trust (gpg [--options] --edit-key *name* | trust)

Signing a key gives it a certain calculated level of trust (depending on how carefully you said you checked the identity of the person specified in the key's User ID). The calculated level of trust is but one of two different **levels of trust** associated with a key. The other type of trust is the owner trust level. We can assign a level of trust to the owner of the key by editing the key (with the **--edit-key** command) and then using the **trust** command.

Let's look at the trust on one of the public keys on our keyring.

```
D:\Programs\gnupg>gpg --edit-key phil

gpg (GnuPG) 1.0.7; Copyright (C) 2002 Free Software Foundation, Inc.
This program comes with ABSOLUTELY NO WARRANTY.
This is free software, and you are welcome to redistribute it
under certain conditions. See the file COPYING for details.

pub  1024D/FAEBD5FC  created: 1997-04-07 expires: never      trust: -/f
sub  2048g/42F0A0A0  created: 1997-04-07 expires: never
(1). Philip R. Zimmermann <prz@pgp.com>

Command>
```

This is the same key that we signed earlier. The trust for this key (-/f) consists of two parts:

- The key's **calculated trust level** -- which is based on the **signature we made earlier** -- is full (f). This calculated level of trust is a measure of the amount of certainty we have that the person specified in the User ID is actually the owner of the key.

- The second trust level on this key, which is none (-), is the **owner trust level**. This owner trust level means something different than the calculated trust level. The owner trust level is a measure of how much we "trust this user to correctly verify other users' keys." In other words, if the owner of this key has signed yet another person's key, how much confidence should we have in the signature on that key? (Our own key signatures carry a very high level of trust

for us; the question now is how much do we trust the signatures that other key owners make?)

We can use the **trust** command to specify an owner trust level. When we use the **trust** command to set an owner trust level, GPG ask us specifically how much we trust the owner of this key to verify the identities of owners of other keys.

```
Command> trust

pub  1024D/FAEBD5FC  created: 1997-04-07 expires: never      trust: -/f
sub  2048g/42F0A0A0  created: 1997-04-07 expires: never
(1). Philip R. Zimmermann <prz@pgp.com>

Please decide how far you trust this user to correctly
verify other users' keys (by looking at passports,
checking fingerprints from different sources...)?

  1 = Don't know
  2 = I do NOT trust
  3 = I trust marginally
  4 = I trust fully
  5 = I trust ultimately
  i = please show me more information
  m = back to the main menu

Your decision? 4

pub  1024D/FAEBD5FC  created: 1997-04-07 expires: never      trust: f/f
sub  2048g/42F0A0A0  created: 1997-04-07 expires: never
(1). Philip R. Zimmermann <prz@pgp.com>
Please note that the shown key validity is not necessary correct
unless you restart the program.

Command> save

D:\Programs\gnupg>
```

Notice that the owner **trust level** has changed from none (-) to full (f). You should **save** your changes.

If you've imported public and secret keys of your own onto your GPG keyrings, you'll probably want to set the owner trust and calculated trust levels to "ultimately trusted" (u/u) for those keys. Use the **--edit-key | trust** command and select "5 = I trust ultimately" when asked how much you trust the user "to correctly verify other users' keys." Setting the owner trust level to "ultimately trusted" will also set the calculated trust level to "ultimately trusted." (For more information on importing keys onto your keyrings, see the **Importing Keys** section above.)

(For more information on trust in a key's owner, see the **GNU Privacy Handbook**.)

Removing Public Keys (gpg [--options] --delete-key *name*)

There may come a time when you want to remove someone's public key from your keyring (perhaps you no longer correspond with that person). To remove people's public keys from your keyring, use the **--delete-key** command.

```
D:\Programs\gnupg>gpg --delete-key phil

gpg (GnuPG) 1.0.7; Copyright (C) 2002 Free Software Foundation, Inc.
This program comes with ABSOLUTELY NO WARRANTY.
This is free software, and you are welcome to redistribute it
under certain conditions. See the file COPYING for details.

pub  1024D/FAEBD5FC 1997-04-07   Philip R. Zimmermann <prz@pgp.com>

Delete this key from the keyring? Y

D:\TEMP>
```

Remember that the **--delete-key** command removes only public keys. If you want to remove one of your own keypairs (which contains a secret key as well as a public key), you'll have to use another command, discussed in the **next section**..

Removing Secret & Public Keys (gpg [--options] --delete-secret-and-public-key *name*)

Although you can use the **--delete-key** and **--delete-secret-key** commands in succession to remove one of your own keypairs from your keyring, you can remove both keys at once with the **--delete-secret-and-public-key** command..

```
D:\TEMP>gpg --delete-secret-and-public-key gpgumbel@cowtownu.edu
```

```
gpg (GnuPG) 1.0.7; Copyright (C) 2002 Free Software Foundation, Inc.
This program comes with ABSOLUTELY NO WARRANTY.
This is free software, and you are welcome to redistribute it
under certain conditions. See the file COPYING for details.

sec  1024D/0A484ECB 2002-05-26   George P. Gumbel <gpgumbel@cowtownu.edu>

Delete this key from the keyring? Y
This is a secret key! - really delete? Y

pub  1024D/0A484ECB 2002-05-26   George P. Gumbel <gpgumbel@cowtownu.edu>

Delete this key from the keyring? Y

D:\TEMP>
```

Keep in mind that once you remove a secret key from your keyring, you won't be able to use it to decrypt messages and files from other people or use it to sign messages and files to send to others.

Revoking Signatures (gpg [--options] --edit key *name* | revsig)

In some circumstances you may wish to revoke a signature you made to someone's public key (perhaps you no longer trust the identity of the person specified in that key's User ID). In that case, edit the key with the **--edit-key** command and use the **revsig** command.

```
D:\Programs\gnupg>gpg --edit-key phil

gpg (GnuPG) 1.0.7; Copyright (C) 2002 Free Software Foundation, Inc.
This program comes with ABSOLUTELY NO WARRANTY.
This is free software, and you are welcome to redistribute it
under certain conditions. See the file COPYING for details.

pub  1024D/FAEBD5FC  created: 1997-04-07 expires: never      trust: -/f
sub  2048g/42F0A0A0  created: 1997-04-07 expires: never
(1). Philip R. Zimmermann <prz@pgp.com>
```

```
Command>  revsig

You have signed these user IDs:
     Philip R. Zimmermann <prz@pgp.com>
     signed by 0A484ECB at 2002-05-26
user ID: "Philip R. Zimmermann <prz@pgp.com>"
signed with your key 0A484ECB at 2002-05-26
Create a revocation certificate for this signature? (y/N) Y
You are about to revoke these signatures:
     Philip R. Zimmermann <prz@pgp.com>
     signed by 0A484ECB at 2002-05-26
Really create the revocation certificates? (y/N) Y
```

If you have signed someone's public key with more than one of your own secret keys, GPG will ask you about the signature made with each of those secret keys.

GPG also asks you to specify a reason for the decision to revoke your signature.

```
Please select the reason for the revocation:
   0 = No reason specified
   4 = User ID is no longer valid
   Q = Cancel
Your decision? 0
Enter an optional description; end it with an empty line:

Reason for revocation: No reason specified
(No description given)
Is this okay? Y
```

As is always the case when working with your secret key, you will be prompted for your passphrase.

```
You need a passphrase to unlock the secret key for
user: "George P. Gumbel <gpgumbel@cowtownu.edu>"
1024-bit DSA key, ID 0A484ECB, created 2002-05-26
```

```
Enter passphrase: My_31337_Passphrase

pub  1024D/FAEBD5FC  created: 1997-04-07 expires: never      trust: -/f
sub  2048g/42F0A0A0  created: 1997-04-07 expires: never
(1). Philip R. Zimmermann <prz@pgp.com>

Command> quit
Save changes? Y

D:\Programs\gnupg>
```

GPG prompts you to save your changes when you **quit**. (You could also **save** to exit back to the command line.)

Once you have revoked your signature from someone's public key, you can use the **--list-sigs** command to view the status of all of the signatures on that key.

```
D:\Programs\gnupg>gpg --list-sigs phil

pub  1024D/FAEBD5FC 1997-04-07 Philip R. Zimmermann <prz@pgp.com>
rev         0A484ECB 2002-05-26   George P. Gumbel <gpgumbel@cowtownu.edu>
sig         FAEBD5FC 1997-04-07   Philip R. Zimmermann <prz@pgp.com>
sig         C7A966DD 1997-04-07   [User id not found]
sig         0525419B 1997-06-04   [User id not found]
sig         D027A0A0 1997-06-04   [User id not found]
sig         413E3D33 1997-06-01   [User id not found]
sig         E2CA6F8B 1997-05-30   [User id not found]
sig         CF73EC4C 1997-06-04   [User id not found]
sig         7C5AABF1 1997-06-01   [User id not found]
sig 3       0A484ECB 2002-05-26   George P. Gumbel <gpgumbel@cowtownu.edu>
sub  2048g/42F0A0A0 1997-04-07
sig         FAEBD5FC 1997-04-07   Philip R. Zimmermann <prz@pgp.com>


D:\Programs\gnupg>
```

Notice that there is now a revocation (rev) that was generated with our secret key. This revocation effectively cancels the

original signature we made (and which is still listed). And once our signature on Phils's key is revoked, the calculated trust level on his key drops to none (-/-).

```
D:\Programs\gnupg>gpg --edit-key phil

gpg (GnuPG) 1.0.7; Copyright (C) 2002 Free Software Foundation, Inc.
This program comes with ABSOLUTELY NO WARRANTY.
This is free software, and you are welcome to redistribute it
under certain conditions. See the file COPYING for details.

pub  1024D/FAEBD5FC  created: 1997-04-07 expires: never      trust: -/-
sub  2048g/42F0A0A0  created: 1997-04-07 expires: never
(1). Philip R. Zimmermann <prz@pgp.com>

Command>
```

(For more information on revoking signatures and other key components, see the **GNU Privacy Handbook**.)

Adding User IDs (gpg [--options] --edit-key *name* | adduid)

When you generated your keypair, GPG asked you for a name and email address to identify you in the User ID for your public key. At some later time, though, you may wish to add another User ID to your key (for example, to include a new email address that you use). You can add User IDs to your key by editing the key (**--edit-key**) and using the **adduid** command.

```
D:\Programs\gnupg>gpg --edit-key gpgumbel@cowtownu.edu

gpg (GnuPG) 1.0.7; Copyright (C) 2002 Free Software Foundation, Inc.
This program comes with ABSOLUTELY NO WARRANTY.
This is free software, and you are welcome to redistribute it
under certain conditions. See the file COPYING for details.

Secret key is available.

pub  1024D/0A484ECB  created: 2002-05-26 expires: never      trust: u/u
sub  2048g/C31174A2  created: 2002-05-26 expires: never
```

```
(1). George P. Gumbel <gpgumbel@cowtownu.edu>

Command> adduid

Real name: George Gumbel
Email address: gumbel@small-net.net
Comment:
You selected this USER-ID:
    "George Gumbel <gumbel@small-net.net>"

Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit? O
```

GPG asks for your passphrase because you are making changes to your key.

```
You need a passphrase to unlock the secret key for
user: "George P. Gumbel <gpgumbel@cowtownu.edu>"
1024-bit DSA key, ID 0A484ECB, created 2002-05-26

Enter passphrase: My_31337_Passphrase

pub  1024D/0A484ECB  created: 2002-05-26 expires: never      trust: u/u
sub  2048g/C31174A2  created: 2002-05-26 expires: never
(1).  George P. Gumbel <gpgumbel@cowtownu.edu>
(2)   George Gumbel <gumbel@small-net.net>

Command> quit
Save changes? Y

D:\Programs\gnupg>
```

The dot (.) next to your original User ID tells you that it is the primary User ID on this key. As you have made changes to your key, GPG prompts you to save those changes when you **quit**. (You could also **save** to exit back to the command line.)

(For more information on adding User IDs, see the **GNU Privacy Handbook**.)

Setting a Primary User ID (gpg [--options] --edit key *name* | primary)

If you have multiple User IDs on your key, you may wish to change the User ID that is designated as the primary User ID. The primary User ID is the User ID that GPG displays by default. To change the primary User ID, edit the key (**--edit-key**) and select the User ID that you want to be the primary User ID.

```
D:\Programs\gnupg>gpg --edit-key gumbel@small-net.net

gpg (GnuPG) 1.0.7; Copyright (C) 2002 Free Software Foundation, Inc.
This program comes with ABSOLUTELY NO WARRANTY.
This is free software, and you are welcome to redistribute it
under certain conditions. See the file COPYING for details.

Secret key is available.

pub  1024D/0A484ECB  created: 2002-05-26 expires: never      trust: u/u
sub  2048g/C31174A2  created: 2002-05-26 expires: never
(1). George P. Gumbel <gpgumbel@cowtownu.edu>
(2)  George Gumbel <gumbel@small-net.net>

Command> 1

pub  1024D/0A484ECB  created: 2002-05-26 expires: never      trust: u/u
sub  2048g/C31174A2  created: 2002-05-26 expires: never
(1). George P. Gumbel <gpgumbel@cowtownu.edu>
(2)* George Gumbel <gumbel@small-net.net>
```

GPG uses an asterisk (*) to indicate your selection. Now issue the **primary** command to set theUser ID you've selected as the primary User ID. You'll need to enter your passphrase because you're making changes to your key.

```
Command> primary

You need a passphrase to unlock the secret key for
user: "George Gumbel <gumbel@small-net.net>"
1024-bit DSA key, ID 0A484ECB, created 2002-05-26

Enter passphrase: My_31337_Passphrase
```

```
pub  1024D/0A484ECB  created: 2002-05-26 expires: never      trust: u/u
sub  2048g/C31174A2  created: 2002-05-26 expires: never
(1)  George P. Gumbel <gpgumbel@cowtownu.edu>
(2)* George Gumbel <gumbel@small-net.net>

Command> quit
Save changes? Y

D:\Programs\gnupg>
```

After you **quit** the **--edit-key** menu (GPG will prompt you to **save** your changes) and then re-enter it, you'll see that GPG has in fact changed your primary User ID.

```
D:\Programs\gnupg>gpg --edit-key gumbel@small-net.net

gpg (GnuPG) 1.0.7; Copyright (C) 2002 Free Software Foundation, Inc.
This program comes with ABSOLUTELY NO WARRANTY.
This is free software, and you are welcome to redistribute it
under certain conditions. See the file COPYING for details.

Secret key is available.

pub  1024D/0A484ECB  created: 2002-05-26 expires: never      trust: u/u
sub  2048g/C31174A2  created: 2002-05-26 expires: never
(1)  George P. Gumbel <gpgumbel@cowtownu.edu>
(2). George Gumbel <gumbel@small-net.net>

Command>
```

You should re-distribute your public key now that you have changed the primary User ID on it.

Removing User IDs (gpg [--options] --edit key *name* | deluid)

Removing User IDs is similar to changing the primary User ID. Edit the key (**--edit-key**), select the User ID you wish to remove...

```
D:\Programs\gnupg>gpg --edit-key gumbel@small-net.net

gpg (GnuPG) 1.0.7; Copyright (C) 2002 Free Software Foundation, Inc.
This program comes with ABSOLUTELY NO WARRANTY.
This is free software, and you are welcome to redistribute it
under certain conditions. See the file COPYING for details.

Secret key is available.

pub  1024D/0A484ECB  created: 2002-05-26 expires: never      trust: u/u
sub  2048g/C31174A2  created: 2002-05-26 expires: never
(1). George P. Gumbel <gpgumbel@cowtownu.edu>
(2)  George Gumbel <gumbel@small-net.net>

Command> 2

pub  1024D/0A484ECB  created: 2002-05-26 expires: never      trust: u/u
sub  2048g/C31174A2  created: 2002-05-26 expires: never
(1). George P. Gumbel <gpgumbel@cowtownu.edu>
(2)* George Gumbel <gumbel@small-net.net>
```

...and issue the **deluid** command.

```
Command> deluid

Really remove this user ID? Y

pub  1024D/0A484ECB  created: 2002-05-26 expires: never      trust: u/u
sub  2048g/C31174A2  created: 2002-05-26 expires: never
(1). George P. Gumbel <gpgumbel@cowtownu.edu>

Command> save

D:\Programs\gnupg>
```

GPG does not require a passphrase to remove User IDs (though you must keep at least one User ID on the key). Don't forget to **save** your changes.

(For more information on removing User IDs, see the **GNU Privacy Handbook**.)

Setting Key Preferences (gpg [--options] --edit-key *name* | setpref)

When you created your keypair, GPG set a default list of preferences for the several different types of algorithms that can be used for encryption, message digests (hashes), and compression. These preferences tell GPG what algorithms are to be employed when your key is used. Since these preferences are stored with your key, they can even affect what algorithms are used by GPG (or PGP) when others use your public key to encrypt messages to you. (For more information on ciphers, keys, hashes, and algorithms, see the **GNU Privacy Handbook**.)

The default preferences that GPG sets on your key are:

Algorithm Type	Preferences (in order)
Symmetric cipher	AES (128), CAST5, 3DES, IDEA
Hash	SHA1, RIPEMD160
Compression	ZLIB, ZIP

You can get a list of the preferences on your key by editing your key (**--edit-key**) and using the **showpref** command.

```
D:\Programs\gnupg>gpg --edit-key george

gpg (GnuPG) 1.0.7; Copyright (C) 2002 Free Software Foundation, Inc.
This program comes with ABSOLUTELY NO WARRANTY.
This is free software, and you are welcome to redistribute it
under certain conditions. See the file COPYING for details.

Secret key is available.

pub  1024D/0A484ECB  created: 2002-05-26 expires: never      trust: u/u
sub  2048g/C31174A2  created: 2002-05-26 expires: never
```

```
(1). George P. Gumbel <gpgumbel@cowtownu.edu>

Command> showpref

pub  1024D/0A484ECB  created: 2002-05-26 expires: never      trust: u/u
(1). George P. Gumbel <gpgumbel@cowtownu.edu>
     Cipher: AES, CAST5, 3DES, IDEA
     Hash: SHA1, RIPEMD160
     Compression: ZLIB, ZIP

Command>
```

You can get another version of the same list with the **pref** command.

```
Command> pref

pub  1024D/0A484ECB  created: 2002-05-26 expires: never      trust: u/u
(1). George P. Gumbel <gpgumbel@cowtownu.edu>
     S7 S3 S2 S1 H2 H3 Z2 Z1 [mdc]

Command>
```

The **pref** command, however, presents the preferences on your key as a *string* using a special set of codes (a kind of notation). If you compare the list (or *string*) of preferences from the **pref** command with that of the **showpref** command, you'll find that they match exactly.

What follows are summary tables of the algorithms supported by GPG 1.0.7, the code (*n*) for each algorithm ("Pref Code" column), and the *name* of each algorithm. (The code *n* and the *name* for each algorithm can be used with many other GPG options pertaining to **algorithms**.) The tables also indicate which algorithms are supported by major versions of PGP as well as the previous version of GPG (1.0.6).

Symmetric Encryption Algorithms

Pref Code (n)	*Algorithm (name)*	*PGP 2*	*PGP 5*	*PGP 6*	*PGP 7*	*PGP 6.5.8ckt*	*GPG 1.0.6*

s1 *	IDEA	X	X	X	X	X	X *
s2	3DES	---	X	X	X	X	X
s3	CAST5	---	X	X	X	X	X
s4	Blowfish	---	---	---	--	X (03)	X
s7	AES (128)	---	---	---	X (7.0.1)	X (03)	X
s8	AES192	---	---	---	X (7.0.1)	X (03)	X
s9	AES256	---	---	---	X (7.0.1)	X (03)	X
s10	Twofish	---	---	---	X	X (03)	X

** only with **IDEA module***

Digest (Hash) Algorithms

Pref Code (n)	***Algorithm (name)***	***PGP 2***	***PGP 5***	***PGP 6***	***PGP 7***	***PGP 6.5.8ckt***	***GPG 1.0.6***
h1	MD5	X	X	X	X	X	X
h2	SHA1	---	X	X	X	X	X
h3	RIPEMD160	---	X	X	X	X	X
h6 +	TIGER192	---	---	---	---	X (08)	X +
h8 *	SHA256	---	---	---	---	X (07)	X *
h9 *	SHA384	---	---	---	---	X (07)	X *
h10 *	SHA512	---	---	---	---	X (07)	X *

** only with **SHA-2 module** & **SHA-2 patch***
*+ only with **TIGER module***

Compression Algorithms

Pref Code (n)	*Algorithm (name)*	*PGP 2*	*PGP 5*	*PGP 6*	*PGP 7*	*PGP 6.5.8ckt*	*GPG 1.0.6*
z0	*uncompressed*	X	X	X	X	X	X
z1	ZIP (RFC 1951)	X	X	X	X	X	X
z2	ZLIB (RFC 1950)	---	---	---	---	---	X

You can change the list of preferences on your key by using the **setpref** command and specifying a new list (or *string*) of preferences in a *string* just like the string that the **pref** command generated. After you change the preferences on your key with the **setpref** command, issue the **updpref** command to save your changes. Since you're making changes to your key, GPG will ask for your passphrase.

```
Command> setpref s2 s9 s4 s3 h3 h2 z1 z2

Command> updpref

Current preference list: S2 S9 S4 S3 H3 H2 Z1 Z2
Really update the preferences? Y

You need a passphrase to unlock the secret key for
user: "George P. Gumbel <gpgumbel@cowtownu.edu>"
1024-bit DSA key, ID 0A484ECB, created 2002-05-26

Enter passphrase: My_31337_Passphrase

pub  1024D/0A484ECB  created: 2002-05-26 expires: never      trust: u/u
(1). George P. Gumbel <gpgumbel@cowtownu.edu>

Command>
```

Once you've updated your preferences, you can check the list of preferences on your key with the **pref** and **showpref** commands -- you'll see that they've changed.

```
Command> pref
```

```
pub  1024D/0A484ECB  created: 2002-05-26 expires: never      trust: u/u
(1). George P. Gumbel <gpgumbel@cowtownu.edu>
     S2 S9 S4 S3 H3 H2 Z1 Z2 [mdc]

Command> showpref

pub  1024D/0A484ECB  created: 2002-05-26 expires: never      trust: u/u
(1). George P. Gumbel <gpgumbel@cowtownu.edu>
     Cipher: 3DES, AES256, BLOWFISH, CAST5
     Hash: RIPEMD160, SHA1
     Compression: ZIP, ZLIB

Command>
```

You can reset the preferences to their defaults by using the **setpref** command and *not* specifying a *string* of preferences.

GPG also lets you specify a default set of preferences in the Options file with the **--default-preference-list** option (*sans* the leading dashes, --). The **--default-preference-list** option uses the same kind of *string* of preferences as the **setpref** command. The *string* of preferences that you set with the **--default-preference-list** option will be the default set of preferences for new User IDs, not new keypairs that you generate, however -- meaning that after you create a new keypair, you'll have to use the **updpref** command to set the list of preferences on your key to the defaults specified in your Options file.

Keep in mind that not all versions of PGP and GPG support the whole range of algorithms that GPG 1.0.7 **"Nullify"** (the Win32 build used in our examples here) does. In particular, the SHA-2 hashes (SHA256, SHA384, SHA512) and the TIGER hash are not supported by most versions of PGP. (GPG can support them with the **SHA-2 extension** and **clear signature patch**, and the **TIGER extension.**) Moreover, no version of GPG supports the IDEA symmetric encryption algorithm by default, since it is still patented in many countries including the United States. GPG can support IDEA with the **IDEA extension** (or module), but IDEA can be used freely only for non-commercial purposes (for commercial uses, you'll have to buy a license from **MediaCrypt**).

Note on RSAv3 Keys: The **setpref** and **updpref** commands, as well as the **--default-preference-list** option, cannot be used with RSAv3 keys, as RSAv3 keys do not store preferences. These commands and options can be used only with DSA/ElGamal and RSAv4 keys, which do store preferences.

Revoking Keys (gpg [--options] --gen-revoke *name*)

At some point in the future you may wish to revoke your key so that no one can use it to encrypt email to you. Perhaps your secret key has been compromised; or perhaps you've generated another keypair and prefer to use that key instead. Whatever the case, you'll need a revocation certificate (which is a special kind of signature) for the key. You can then import that revocation certificate onto your keyring to revoke the key. You can also distribute the revocation certificate to others in order to revoke copies of your public key that you previously distributed.

You ought to consider generating a revocation certificate immediately after creating your keypair. You need not import that revocation certificate; you can simply store it for future use. Revocation certificates are especially useful if you've forgotten the passphrase to your secret key and you need some way to "disable" or revoke that key. Since you've forgotten the passphrase, the only way to revoke the key will be with a revocation certificate that you generated earlier (when you still remembered the passphrase). In a way, a revocation certificate is a kind of insurance plan that lets you keep ultimate control over your key, even if you lose or forget the passphrase.

You can generate a revocation certificate by using the **--gen-revoke** command and specifying the key for which you want a revocation certificate generated. Use the **--output** option to specify an output file for the revocation certificate.

```
D:\Programs\gnupg>gpg --output gumbel-rev.asc --gen-revoke george

sec  1024D/0A484ECB 2002-05-26   George P. Gumbel <gpgumbel@cowtownu.edu>

Create a revocation certificate for this key? Y
```

GPG asks you for a reason for the revocation.

```
Please select the reason for the revocation:
   0 = No reason specified
   1 = Key has been compromised
   2 = Key is superseded
   3 = Key is no longer used
   Q = Cancel
(Probably you want to select 1 here)
Your decision? 3
Enter an optional description; end it with an empty line:

Reason for revocation: Key is no longer used
(No description given)
```

```
Is this okay? Y
```

You will need to enter your passphrase -- that's why it's a good idea to generate a revocation certificate now, while you are confident that you know the passphrase. If you've forgotten your passphrase, you won't be able to generate a revocation certificate.

```
You need a passphrase to unlock the secret key for
user: "George P. Gumbel <gpgumbel@cowtownu.edu>"
1024-bit DSA key, ID 0A484ECB, created 2002-05-26

Enter passphrase: My_31337_Passphrase

ASCII armored output forced.
Revocation certificate created.

Please move it to a medium which you can hide away; if Mallory gets
access to this certificate he can use it to make your key unusable.
It is smart to print this certificate and store it away, just in case
your media become unreadable.  But have some caution:  The print system of
your machine might store the data and make it available to others!

D:\Programs\gnupg>
```

Once you've generated a revocation certificate, you can either store it safely for future use, or immediately import it onto your keyring. To import the revocation certificate, use the **--import** command.

```
D:\Programs\gnupg>gpg --import gumbel-rev.asc

gpg: key 0A484ECB: revocation certificate imported
gpg: Total number processed: 1
gpg:    new key revocations: 1

D:\Programs\gnupg>
```

After you've imported the revocation certificate, you can verify that it was imported properly to the revoked key with the **--list-sigs** command.

```
D:\Programs\gnupg>gpg --list-sigs gpgumbel@cowtownu.edu

pub  1024D/0A484ECB 2002-05-26 [revoked]
rev          0A484ECB 2002-05-26   George P. Gumbel <gpgumbel@cowtownu.edu>
uid                               George P. Gumbel <gpgumbel@cowtownu.edu>
sig 3        0A484ECB 2002-05-26   George P. Gumbel <gpgumbel@cowtownu.edu>


D:\Programs\gnupg>
```

Notice that the revocation certificate shows up as a signature. The **--edit-key** command shows the calculated trust as revoked (r).

```
D:\Programs\gnupg>gpg --edit-key gpgumbel@cowtownu.edu

gpg (GnuPG) 1.0.7; Copyright (C) 2002 Free Software Foundation, Inc.
This program comes with ABSOLUTELY NO WARRANTY.
This is free software, and you are welcome to redistribute it
under certain conditions. See the file COPYING for details.

Secret key is available.

pub  1024D/0A484ECB  created: 2002-05-26 expires: never      trust: u/r
sub  2048g/C31174A2  created: 2002-05-26 expires: never
(1). George P. Gumbel <gpgumbel@cowtownu.edu>

Command>
```

Other people can import your revocation certificate onto their keyrings to revoke their copies of your public key. If someone who has imported your revocation certificate attempts to encrypt messages or files to you using that revoked key, GPG will tell them that the key is unusable (because it's been revoked).

```
D:\TEMP>gpg --recipient gpgumbel@cowtownu.edu --encrypt my-file.txt
```

```
gpg: gpgumbel@cowtownu.edu: skipped: unusable public key
gpg: my-file.txt: encryption failed: unusable public key

D:\TEMP>
```

Of course, some PGP and GPG users may not receive your revocation certificate before they use your (revoked) key to encrypt messages or files. In other words, some people may not know that you have revoked your key and will encrypt messages to you using that key. If someone does encrypt a message or file with your revoked key public key, you will still be able to decrypt it with your secret key (private key), so keep the revoked key pair on your keyring.

(For more information on generating revocation certificates, see the **GNU Privacy Handbook**.)

Home [frames] **Home [no frames]**