Creating Your Personal Key Pair

GPG uses public key cryptography for encrypting and signing messages. Public key cryptography involves your *public key* which is distributed to the public and is used to encrypt messages to be delivered you and to decrypt signatures you have created, and your *private key* which complements your public key by allowing you to decrypt messages you receive and to encrypt signatures. Together, these are referred to as a *key pair*.

1.1 About Key Security

When you create a key pair, both your public and private keys must be stored on your computer. This creates a security risk, because anyone who can gain access to your private key can decrypt your messages and impersonate you. You can't commit your private key to memory and erase it from your computer--it is far too long (usually at least 1024 bits--that's like memorizing a 300-digit phone number!) Besides, even if you could memorize it, it would take too long to type it out each time you wanted to use it.

The practical solution to this problem is to go a step further and actually encrypt your private key using a reasonably short *passphrase* as a key for the key; this is what GPG does. Each time you perform an operation involving your private key, GPG reads the encrypted key from the disk, prompts you for your passphrase, decrypts the key in memory, and finally uses it. In order for this system to work, you must have a cryptographically strong passphrase--something that can't be guessed or brute-force attacked. For example, "orange" is so short that a dictionary attack will find it in seconds, and your spouse's full name is a poor choice because anyone who knows you might guess that you used it as your passphrase. Your passphrase should be a combination of letters and numbers and upper and lower case.

1.2 Using the GPG Gen-key Command

Now that you've got your passphrase, you're ready to generate your personal key pair. At the command prompt, type

```
gpg --gen-key
```

GPG responds with a menu asking what kind of key pair you want to generate. Choose

the default, "DSA and ElGamal."

Next, it asks you the size of the key. Again, choose the default of "1024." If your key is too small, it is easier to crack; if it is too large, then every operation that uses it may execute too slowly.

The next prompt asks you when (if) the key should expire. Letting the key expire after a certain amount of time adds a little bit of security, because documents encrypted after this time are not connected with the old key in any way. The tradeoff, of course, is that everyone you correspond with must fetch your new key when the old one expires. At the prompt, choose a reasonable time period for the lifetime of your key, or select "key does not expire."

Now you enter the *Real Name* and *Email Address* which will be used to identify the key in everyone's key collections, not just yours. If you don't feel comfortable attaching your full real name to your key, be sure to pick something unique so that your key won't be confused with anyone else's.

Finally, enter the passphrase you selected. GPG will **NOT** give you any feedback as you type your passphrase--it won't print asterisks or spaces as most password input functions do. This is yet another security measure. After you enter and then confirm your passphrase, GPG will start doing some number crunching to generate all the random bits it needs in your key pair. It uses all sorts of sources inside the computer to simulate randomness, including console input--so if you want to speed up the process, type some random characters on the keyboard.

1.3 Publishing Your Public Key

The easiest way to publish your public key is to simply post it on a web page or email it directly to people who need it. Of course, this is also one of the least secure ways of doing it--plaintext email and web pages can be (theoretically, at least) subject to man-in-the-middle attacks. More advanced methods of key exchange involve things such as webs of trust and key servers, which are beyond the scope of this guide. If you don't believe specifically that someone will try to attack you, you may be comfortable with the method described here.

You have been warned. The exchange of public keys without a trusted intermediary can be subject to a man-in-the-middle attack.

To transmit your public key over the Internet, the first step is to export it to ASCII format. Open a command prompt window and go to a folder where you want to place the

exported key. Type

```
gpg --armor --output "key.txt" --export "YOUR-NAME"
```

You may change *key.txt* to some other filename if you'd like. *YOUR-NAME* can be your *Real Name* or your *Email Address*; GPG will find it either way. The "`--armor`" option instructs GPG to format the output "armored" for plain-text transmission. This makes it easy to copy and paste the key to and from web pages and email messages. The "`--armor`" option applies to most GPG commands that produce any kind of output.

Open the output file, *key.txt*. You should see something resembling this:

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v1.2.1 (MingW32)

mQGiBD53m34RBAC6GXvDFWD3a+GOkQKubz5Koq9lks9d+gel29/sA5kqSfQnoaeR
qdTKLlB+oNsVjDX/Szfi3fsrK5zmKKZVHv3JO4DkxtABf4HgfaGkpav2PvXevYoB
YWSGTclHOHN5D3xsbIX6wvunkNhllcqrFlC3braG2tQnt2+PMk1gMA2jhwCg7qam
RtC27n0RY25jMfM/fESTImcD/1OtFwRtchqjPvl2IHSCBlltJyksuStevOfAFnc1
p3H+JqdUiKVf8oAF4NP4KarXL34xPCJXLKlBwHC3SH8powy2HX0mhsCjvVQQeLOP
fU3Q5DJxGM16hfmqlD1k4a7NUjnXwlCrce4rVToFbCnLrJTVoMDyNhowrXPbPCVM
N3FNA/0Uciz19raTBGkwo6kpsicpZG7Mk4eGFK5ssWxPxlLYoQ7yzgekWd9h6zBT
IQpdkatzdNf7xDEaUEBhO4vD7Il02OLigro95N5savsH9StTNCsJgmaiwtX5hxuF
WpW974xgfCmbGOPnbs1QTuyT85VllLxHI5fsGOYKCd/qo8H4nLQHQksgVGVzdIhZ
BBMRAgAZBQI+d5t+BAsHAwIDFQIDAxYCAQIeAQIXgAAKCRDCVd5vH6A96m7fAKDs
YVhdSqNn+u/rkj1pU6kFldY0JwCcCCXth72RJ9tAIz5gq9M3m6y2+7q5AQ0EPneb
gRAEAIb/sxLIAKaahBfBpGxpn3ZKhvug1z6yP7jLWFNLFugaakYjm5LXsI5Hpj06
mAE2fJPUNc1of0ZaK4La3XA8l/nVaadDP6FDqnxuPv3ne5JAxcK76ecT+m0ljQXZ
oVnUkUqeNnwtcIs0fmmXnaeD68OHdidYsIuuEDhrFvPAT6cfAAMFA/923B1Bm4NR
riLy8QxYNuTxImtxoVg4NtfnYuHWyoxP1Ic3C1nZD2+fxf2685KNKx+3ZwaE81zN
ZNl0kNdFhB24Vmr6HM5C+eSlHj8C+LOUdP1A/9Un5utceg/qjNYkRXJx5mjyCizh
Gg/+1mLB5e+OA9Tl5R+96PKPFov+UjNPu4hGBBgRAgAGBQI+d5uBAAoJEMJV3m8f
oD3qewYAoNFRCBVfbX+LKxmWOZoqyQhB7jfnAKDDOCzQhZLZmrf0Uqdk6yj+HDm0
rA==
=+DfK
-----END PGP PUBLIC KEY BLOCK-----
```

Copy the entire contents of that file, including the lines beginning with "`-----`," and paste them on a web page or in an email message to someone else. That's all it takes to export your public key.

Note: it is possible to export your private key in the same manner, with the "`--export-secret-keys`" instead of "`--export`." This is useful for backup or for transferring it to another computer, as long as you know what you're doing and you can be sure the new copy will be secure. Make sure you don't ever accidentally publish your *private* key! Before you publish a key in ASCII form, check that the first line has the word "`PUBLIC`," not "`PRIVATE`."

1.4 Backing Up Your Keys

Keeping your GPG key files safe is just as important as remembering your passphrase--your passphrase does you no good if you don't have a copy of your full private key on your computer. If your key file is destroyed, there is absolutely no way to reconstruct it, short of executing a cryptanalysis on your own data. To back up your GPG keys, including all your private keys, locate the files *pubring.gpg*, *secring.gpg*, and *trustdb.gpg* in the folder where you have installed GPG, and copy them to a safe location, such as a CD-R disc. Store this disc in a safe place where potential attackers won't likely be able to find it. (Of course, your keys are safe long as your attacker doesn't know your passphrase.)

Now, you are finally ready to actually encrypt something.

2. Encrypting and Decrypting Files

The basic encryption and decryption procedure in GPG is this: The sender determines the *recipient* of a file, acquires that recipient's public key if he hasn't already done so, and then runs the plaintext through GPG along with this key to obtain the cyphertext.[4.1] When the recipient wants to decrypt the file, he applies his private key to the cyphertext to obtain the plaintext.

In fact, the sender and recipient aren't always different people. One important use of GPG is to encrypt your own data, storing the cyphertext and destroying the plaintext. This is an excellent defense against physical attacks on your computer or your local file server.

2.1 Using the GPG Encrypt Command

Choose a file you want to encrypt. For example, let's assume you have diary, where each month is a new file, and you're done with February 2003, which is called *diary 2003-02.txt*. Suppose you want to encrypt this file and then put it away in an archive folder or a CD-R disc. At the command prompt, type (all on one line)

```
gpg --recipient "YOUR-NAME" --output "diary 2003-02.txt.gpg"
 --encrypt "diary 2003-02.txt"
```

Don't forget to fill in *YOUR-NAME* with the actual name *you* attached to your key. Always remember the "-output" option when you use an encryption command in GPG; if

you omit this option, the output will be dumped to the command prompt window instead of to a file. Finally, notice that the command (usually an action verb) always goes in the last position on the GPG command line, after any options. Now *diary 2003-02.txt.gpg* will contain a seemingly random string of bytes. You can look at it with Notepad if you'd like.

There is a similar command, "-encrypt-files," which will automatically choose and name an output file for you. But the filename it chooses will be missing the extension of the plaintext filename (*.txt*, *.jpg*, *.zip*, etc.) so I don't use it, myself.

2.2 Using the GPG Decrypt-Files Command

Now, suppose a year from now you're feeling nostalgic and you want to read February 2003's diary. You would copy the cyphertext back to your workspace on your computer, and type the following at the command prompt:

```
gpg --decrypt-files "diary 2003-02.txt.gpg"
```

GPG will look up your private key and prompt you for the passphrase. Provided your private key is still installed on your computer, and you still remember your passphrase (you didn't write it on a Post-It and stick it on your monitor, did you?) you will get back the original plaintext exactly as it was before you encrypted it. If you want to decrypt a short file and display it immediately in the console, you can use the "`--decrypt`" command instead of the "`--decrypt-files`" command.

2.3 Sending an Encrypted File by Email

Encrypting your own files is useful, but a more common use of GPG is to send encrypted data to someone else. Before you can use GPG to encrypt a file for someone else, you need to get their public key.

2.3.1 Importing the key

As I said before, two convenient ways of getting someone's public key are email and personal web pages. As an example, you can download *my* public key from my web server; try it right now. Go to www.glump.net/signature . If you scroll down a page or so,

you will see a familiar-looking block of text representing my public key. Save the entire page to a text file. (In Internet Explorer, choose the type "Text File" in the "Save As" dialog box.) Suppose you named the saved file *brendan.txt*. At the command prompt, type

```
gpg --import "brendan.txt"
```

GPG should say

```
gpg: key A3CA0378: public key "Brendan Kidwell <brendan@glump.net>"
 imported
gpg: Total number processed: 1
gpg:               imported: 1
```

Notice that GPG wasn't distracted by all the extra text on the page. It looks for the telltale "`BEGIN PGP PUBLIC KEY BLOCK`" line and ignores everything outside that block of text.

One more step you need to perform after you've imported a key from an external source is set the trust level on it. GPG is paranoid, and if you use the key right now as it is, you will get a warning message saying that you haven't established the authenticity of the key. To make this warning message go away, use the GPG "`--edit-key`" command to set the trust level:

```
gpg --edit-key "Brendan Kidwell"
```

GPG will enter the interactive key editing mode. Enter the command "`trust`" and select level "**5**) I trust ultimately." Then enter "`quit`" to save your change.

2.3.2 Encrypting the message

Now you're ready to encrypt the file. Let's assume you have a file you want to send to me called *message to brendan.txt*. At the command prompt, type

```
gpg --armor --recipient "Brendan Kidwell"
 --output "message to brendan.txt.asc"
 --encrypt "message to brendan.txt"
```

GPG will produce a file called *message to brendan.txt.asc*, whose content you can copy and paste into an email.

Alternatively, if you need to send a particularly large file, you should use the encrypt command without the "`--armor`" option:

```
gpg --recipient "RECIPIENT" --output "FILE.gpg" --encrypt "FILE"
```

and instead of pasting *FILE.gpg* into the body of the email, include it as an attachment. Make sure the *name* of the file doesn't reveal anything that should be secret.

If you'd like, you can go ahead and try to send an encrypted message to me. I'll let you know if I receive it correctly. If you want an encrypted reply, make sure you include your public key or tell me where I can get it.

2.4 Decrypting Files Sent by Email

How you deal with an encrypted email message which you have received depends on how it was sent to you. When you receive an encrypted message, its body might contain

```
-----BEGIN PGP MESSAGE-----
```

followed by a string of random-looking characters. Or the message might simply have an attached file whose name ends with "*.gpg*" or "*.pgp*."

2.4.1 Encrypted data is in the message body

If the encrypted data is in the message body, save the entire message to a file, and end the file name with *.asc*. If you know that the encrypted data is some binary format, include the file extension before the *.asc*. For example, if you know the message contains an encrypted Microsoft Word file, you would name the file *message.doc.asc*. At the command prompt, type

```
gpg --decrypt-files "FILE.asc"
```

where *FILE.asc* is the filename you used to save the message. GPG will tell you who the file was encrypted for and prompt you for the passphrase. If the file wasn't encrypted using *your* public key, GPG give up and tell you that it doesn't have the private key needed to decrypt this file. If the decryption succeeded, you should get the original file back, with the name you gave it, minus the *.asc* extension.

Remember, if you know the encrypted data is just a short text message, you can display it on the console instead of storing it in a file with the "`--decrypt`" command:

```
gpg --decrypt "FILE.asc"
```

2.4.2 Encrypted data is in an attached file

If the encrypted data is in an attached file, save that file to your computer. At the

command prompt, type

```
gpg --decrypt-files "FILE.gpg"
```

where *FILE.gpg* is the name of the file you saved. (If the message was created using PGP,[4.2] the name of the attached file will probably end with "*.pgp*" instead.) Again, GPG will only work if you have the private key needed to decrypt the file.

2.5 Encrypting for Multiple Recipients

Sometimes, you'll want to send an encrypted file to more than one person. This could create a problem, though, because no one should be sharing a private key with anyone else. You could always make a separate encrypted file for each recipient, but this can get tiring if you need to send a file to as many as five people.

There is a better way: GPG allows you to specify a list of people who may be able to decrypt a file. GPG will then use all of those individuals' public keys to encrypt the data in such a way that any one of their private keys (and no one else's) can decrypt the data.

The syntax is straightforward. Just add more "`--recipient`" options to the command line. Suppose you wanted to encrypt the same message as in Subsection 4.3.2 above, but wanted to send the message so that both you and I could decrypt it later. You would type the following at the command prompt:

```
gpg --armor --recipient "Brendan Kidwell"
 --recipient "YOUR-NAME"
 --output "message to brendan.txt.asc"
 --encrypt "message to brendan.txt"
```

and then copy the output file into an email message as before.

Sending encrypted email this way can make it easier to manage your saved correspondence. Normally when you send plaintext email, a copy of the sent message is saved somewhere in your email software (unless you specified that you don't want to save copies.) You can always go back and review your sent email to recall what was said. If you specify your target *and* yourself as recipients when you prepare an encrypted message, then you can go back and review it in your sent email collection whenever you need to, with only the added step that you need to decrypt it before you view it. You needn't save a separate plaintext copy of the message, nor do you need to make another copy encrypted for yourself.

Specifying several recipients does not adversely affect the size of the encrypted data. I tried encrypting a large (~8MB) compressed binary file for one and then two recipients.

The difference in the size of the output was only a few hundred bytes.

3. Signing Files

Often it is desirable to verify the origin of data, whether it is encrypted or not. GPG's signature functions provide a means of verifying authenticity.

The theory is simple. Public and private GPG keys work either way. Once you have encrypted data with one of the keys in a pair, it can only be decrypted with its complement in a the same key pair. Normally GPG operates by encrypting with the public key so that only the recipient can decrypt the data using his private key.

Digital signatures work the other way around; data is encrypted using the signer's private key. If someone receives the file and succeeds in decrypting the data with the signer's public key, then presumably, the data *must* have been encrypted by that signer. Therefore, the signer must have created the data himself, or at least approve of its contents in some way (depending on the nature of the actual data.) A digital signature is just as useful as a physical one made with a pen, and arguably, it is more secure.

When GPG creates a digital signature, it doesn't encrypt the entire file with the signer's private key. Instead, it computes a *hash value*,[5.1] encrypts that, and *appends* it to the original data as the signature. This makes it possible to create signed files that are readable without any encryption software, and aren't significantly larger; GPG is needed only to *verify* the authenticity of the file.

To verify a signature, GPG reads the data that was signed and computes its hash value. Then it decrypts the signature, using the signer's public key, to obtain the true hash value. If the two hash values match, the signature is valid and the data you have is exactly the data the signer had when he created the signature.

3.1 Using the GPG Clearsign Command

Suppose you want to send a message to someone in such a way that they can prove it was you who authored the message. First, compose the message in a text editor and save it as *message.txt* in a convenient folder. Then, at the command prompt, type

```
gpg --local-user "YOUR-NAME" --clearsign "message.txt"
```

Since this operation involves your private key, GPG will prompt you for your passphrase. After that, GPG will compute a signature and write a new file called *message.asc*

containing the plaintext and the signature. The contents of this file can be copied into an email and sent to the intended recipient.

As an example, here is message that I have signed:

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

This is a test message signed by Brendan Kidwell.
-----BEGIN PGP SIGNATURE-----
Version: GnuPG v1.2.1 (MingW32)

iD8DBQE+fnwc4lxlBKPKA3gRAq13AJ4557Md6xF15OoEDyIIB+UvDQKwmwCfcrCY
na12Ng9W4K5mP1ZWEueNjCo=
=73hB
-----END PGP SIGNATURE-----
```

3.2 Verifying a Clearsigned Message

Suppose you receive a message like the one produced in the previous section. Or you might find such a message posted on a public web site or electronic message board. Before you can verify its signature, you need to obtain the signer's public key and install it on your computer. This procedure is described in the previous chapter, under the heading "Importing the key."

If the message to be verified is contained in an email, export it to a text file. If the message is displayed on a web page or some other online medium, save it as a text file (named, for example, *message.txt*.) Then type the following at the command prompt:

```
gpg --verify "message.txt.asc"
```

GPG will locate the signer's key if you have it, and use it to check the signature and report whether or not it is valid.

If you're reading the online version of this document and you've already installed my key, you can try copying the test message displayed in the previous section into a text file and verifying the signature with this procedure.

3.3 Signing and Verifying Binary Files

Text messages can have signatures appended to them without disrupting the contents of the message too much, but binary files such as Microsoft Word documents and Zip archives can't have arbitrary data attached to them. To sign binary files, it is costumary to

have GPG create a separate signature file. Suppose you have a Zip archive you want to sign, called *monthly report.zip*. Type the following at the command prompt:

```
gpg --local-user "YOUR-NAME" --output "monthly report.zip.sig"
 --detach-sign "monthly report.zip"
```

Again, GPG will prompt you for your passphrase and then it will generate a signature in *monthly report.zip.sig*. If you were going to email this to someone, you would attach both files to the email message.

Now suppose you're on the other end and you receive a file with a signature like this via email. Save both files to the same folder and type the following at the command prompt:

```
gpg --verify "monthly report.zip.sig"
```

GPG will verify the signature of the file using the signer's public key and report whether or not it is valid. Again, the person doing the verifying must have a the signer's public key installed.

Software distributed over the Internet is often signed in this manner--especially software that relates to security. A user can download a large installation package quickly from a local site, which need not be trusted. After the download is complete, he can go back to the creator's web site and fetch a public key and the signature for the installation package and use them to verify the package's authenticity.

3.4 Encrypting and Signing at the Same Time

It is possible to encrypt and sign a file at the same time. Use this command to encrypt and sign a file:

```
gpg --local-user "YOUR-NAME" --recipient "RECIPIENT" --armor
 --sign --output "FILENAME.asc" --encrypt "FILENAME"
```

This produces an output file named *FILENAME.asc*.

To decrypt such a file, simply run

```
gpg --decrypt-files "FILENAME.asc"
```

GPG will see that the file has been signed and it will automatically verify it if it has the signer's public key.

And, as always if you prefer simple binary output, omit the "`--armor`" option.