An Introduction to GNU Privacy Guard

By David D. Scribner
August 23, 2002

"It's personal. It's private. And it's no one's business but yours." – Philip Zimmermann

GNU Privacy Guard, or GnuPG (http://www.gnupg.org/), is the open-source equivalent of Philip Zimmermann's PGP (Pretty Good Privacy) encryption/authentication software released under GPL. Philip Zimmermann and others developed PGP in 1990 using the Rivest-Shamir-Adleman (RSA) public-key cryptosystem to answer the need for private and secure communications between individuals over digital medium. After its release to the public in 1991, it quickly grew to become the de facto standard worldwide for secure public-key encryption.

Even though the concept of public-key cryptography for encryption purposes was introduced close to three decades ago, and PGP has been around for over a third of that, you'll likely find that for some reason only a small number of PC users take full advantage of public-key security. I myself am guilty of this. When I first started using PGP 2.6 for DOS in the mid-nineties, it was used for little more than encrypting local documents and files for my own use. Few outside of technical circles had heard of it, and the majority of the people I talked to felt it was too cumbersome to use on a regular basis. That was then.

Today, I use GnuPG for a variety of tasks. Whether it's to sign and encrypt documents and contracts submitted to businesses, encrypt local files or merely sign email and files to ensure others that no modifications have occurred to its content, I have found GnuPG to be a "must have" utility kept close at hand when using my PCs.

Why did I switch from PGP to GnuPG? A lot of personal reasons, really, but it all basically came down to trust, and the fact that I believe in GPL. GnuPG is also fully OpenPGP compliant, and was built from scratch from the ground up. It does not natively use any patented algorithms, supports a very wide array of current cipher technologies, is built to easily integrate future cipher technologies, and decrypts and verifies PGP versions 5.x, 6.x and 7.x messages. I still keep my early RSA keys around in case the need arises to decrypt a file from one of my old archives, but eventually even those files will be wiped, or re-encrypted with one of the newer, and perhaps stronger cipher algorithms available with GnuPG.

GnuPG is available for various flavors of Linux/UNIX, as well as Windows and Macintosh operating systems. As a GNU/Linux advocate, the focus of this article will be on using GnuPG with Linux, though the commands and procedures listed would apply across any of the supported platforms. As there is already very fine documentation for GnuPG, including the man page, The GNU Privacy Handbook, the GnuPG mini-HOWTO and the GnuPG FAQs, I won't go into great detail on all the many command-line operations or configuration settings. Instead, I'll try to explain in simple terms what GnuPG can do, the very basics in using it, and why it can be so important in becoming a valuable utility in your toolbox, both personally and professionally.

Privacy, do you need it?

When most people talk about privacy and strong encryption in the same sentence, they often think the only people needing such things must be doing something wrong, illegal, or involved with government espionage. Spies, smugglers, mobsters, and terrorists come to mind, so you, not fitting

into any of these categories and being an honorable and upstanding citizen certainly don't need to be involved with this kind of stuff, right?

You may also think, with Linux's advanced security features such as a protected filesystem structure, memory and strong policies in place, along with a good firewall and IDS, your private data (that list of important account numbers for example) is safe. Perhaps... at least until your fortress gets cracked. By then however, your private data may have been compromised, leaving you wondering what additional steps could have been taken to protect it.

Perhaps you conduct a lot of your business via email, sending quotes, contracts, or financial reports over the Internet to your corporation, boss, or business partners. If you've used the Internet for any length of time, surely you know that the email containing your documents takes multiple paths, and makes multiple "pit stops" at various servers before it reaches its destination. Are you sure that when your document passes through these servers that your files and communications are not scanned, maybe even forwarded on to a paying competitor?

Maybe your data isn't that confidential, but you would still like to ensure that the document or communication you're submitting reaches its final destination exactly as it was sent... no additions, no subtractions, no modifications at all. Can you be sure of this when sending it via standard email procedures?

Suppose you don't have any secrets, and there's not any real concern about someone modifying your email. Do you still want your ISP's service technician with nothing better to do snooping through your email on the server while it's sitting there waiting to be retrieved to know all about your life? If you're in a small or rural town, your ISP may be just a core crew that runs the show. Do you trust them fully with your account? If you sometimes email your family traveling itineraries, information about doctor visits or what stocks you've bought recently, do you really want your ISP to possibly know this information, too?

Even if you're using a large corporate ISP with established privacy policies, can you be sure that their policy is enforced all the way down, and at every level? Believe me, there's been more than one technician discovered snooping through user's mailboxes without their knowing. Some of these "technicians" consider such a practice as entertaining as others might consider soap operas! As Philip Zimmermann stated in the original PGP User's Guide, "It's personal. It's private. And it's no one's business but yours." If you send plain-text email over the Internet, it may be personal, and it may be private, but it can easily become anyone else's business, too!

When you start thinking about it, you will no doubt find many areas in your life needing increased privacy, encryption and digital signatures, which may include private files, documents and email. If the need arises to share those files, documents and email with others, you may want to seriously consider incorporating public-key security into your routines.

Public-Key Security

The concept behind public-key security is that there is no need to disclose what should be kept most private, your secret key. With conventional encryption techniques, you encrypt a file for example with a password or passphrase. However, should you need to deliver that file to someone else, you will also need to disclose that password or passphrase to him or her so they can decrypt the file.

How can this be accomplished, and securely? Well, you could personally hand-deliver the key to the other party, along with the file, however this may not always be possible due to distance or other restrictions. You could email the file, then snail-mail, phone or fax the key to them, but between the time it leaves your hand, mouth or fax machine and is received by the other party, you still can't be 100% sure that it wasn't intercepted.

You could also email the file, and in a separate email, disclose the key. However even this is not secure, and less so than one of the above methods at that! Short of hiring trusted "key couriers" to deliver your password or passphrase such as government agencies might do, there is one alternative and that is to use two keys. One key, kept private, is used to decrypt or sign information, and the other key is made "public" and is used to actually encrypt the data or verify the signature.

That is the basis behind GnuPG and PGP techniques. Using various cipher algorithms you create "key rings" to hold your secret (private) and public keys. Your secret keys are protected by passphrases known only to you and should be kept secure. But your public key, which can be dispersed freely, instructs the GnuPG/PGP application used by others on how to encrypt the data, after which only one key can then decrypt it... your secret key.

Before you create your first key pair, you should come up with a good passphrase to protect it. The word "passphrase" is used, as it should really be more than just a pass "word." There are utilities available to help you choose a passphrase if needed (Diceware is one such utility), but the general idea is to come up with a string of words or characters using mixed case, and one that includes numbers, punctuation marks and special characters that will provide strong resistance to cracking. The trick is to come up with a passphrase you're not likely to ever forget!

You *could* use a single password, but if you're protecting your data with strong encryption ciphers, why make the weakest link (the passphrase) even weaker with something that might make it easier for someone to crack? You could also use a sentence perhaps, just be careful. Most all of the common and well known "phrases" and quotes are already included in a cracker's toolkit, right along with all the dictionary words, and in multiple languages at that.

You can always change your passphrase after the secret key has been generated as well. Based on your use of GnuPG, with regards to privacy and surroundings, the passphrase *should* be changed regularly. Some suggest monthly, but if you ever find yourself doubting the security of your passphrase because someone was too near your shoulder when it was last entered, it should be changed the first chance you get. This feature is also handy if you find yourself wanting to experiment and work with GnuPG for a short session of learning... change your passphrase to something that can be quickly entered for the exercise, then change it back when your session is over.

More information on creating a strong passphrase can be found in the documentation or on the Internet, but the same common sense used in creating a strong password should be used in creating a stronger passphrase. Don't write it down. Don't tell it to anyone. And remember, if you forget your passphrase, you're sunk. There's truly no way to retrieve it.

So, with your freshly installed copy of GnuPG you can create a pair of these keys easily with:

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

By following the prompts, you will indicate what type of key you want to create (signing, encryption, or both) decide on a key size, expiration date, and enter your name, email address and an optional comment. You will also enter the good passphrase you decided to protect the pair with and the key generation will begin. After you've generated your first key pair, you can see that the keys are now in your keyring with the command:

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

This, as the command indicates, lists all keys currently in your public keyring. If you're just starting out with GnuPG the only key listed will be the one just created. In this example, the output's format will resemble (with your own identity of course):

```
pub  1024D/F357CB52 2002-07-09 GnuPG User <user@some-email.com>
sub  2048g/A9CB69B3 2002-07-09
```

In the example output above, you can see that the key listed has a key ID of F357CB52, and contains the name and email address of the key's owner. If your public keyring contains several keys for other users, you can specify which key you want to view by simply adding the "key specifier" of the key you want to view after the '`--list-keys`' directive. The "key specifier" could be the owner's name, the key's ID, or the user's email address.

Depending on how many keys are on your keyring, an owner's name could be as simple as their first name, or their last name. If there's only one Kathy in your keyring, you could use Kathy. If there are two keys belonging to Kathy, but the email addresses are different, the email address of the chosen key could be used. If both keys have the same name and email addresses, the key's ID can be used to differentiate them.

When using an email address to identify a key, it's not necessary to use the entire email address, only the part that's unique to that user's key. Whatever means you use to identify that key, all that's necessary is for it to be unique enough to identify the key you want. One note to make is that when using a key ID as a specifier, since the key ID is a hexadecimal number you can also prefix the ID with "`0x`". If you were specifying the key ID F357CB52, it could be entered as `0xF357CB52`. This is not required for GnuPG, but as you start working more with key servers, you will find that when using a key ID to search their database, they need to be prefixed with "`0x`".

I highly suggest that the first thing you do after creating a key pair is generate a revocation certificate for that pair, then move it onto a diskette you can keep secured someplace safe (along with a printed hard copy of the certificate in case the medium becomes damaged). To create a revocation certificate for the key pair we just created (using the key ID as exampled above), issue the command:

```
$gpg --output revokedkey.asc --gen-revoke 0xF357CB52
```

This will generate a revocation certificate for the key ID F357CB52. The resulting file, "`revokedkey.asc`" is the revocation certificate.

What is a revocation certificate and why do you need one? In the beginning, if you're just experimenting with GnuPG and haven't dispersed your public key, there might not be a need for one. But, once you've started using GnuPG to encrypt files and your public key is out there for others to use, a revocation certificate adds another layer of protection to your key pair.

Should your secret key become compromised or lost, or if you forget your passphrase, a revocation certificate posted to a keyserver or sent to your contacts to update their keyring will not only inform them of your key's demise, but also prevent them from encrypting new files to that public key (and likewise prevent you from using that key to sign or encrypt files yourself). If your key has been compromised, you can still use the secret key to decrypt files previously encrypted to or by you, and others can still verify your signatures created before the revocation, but it's an added safeguard you should take now.

If you've forgotten your passphrase, there's nothing you can do short of a brute-force attack of trying every passphrase combination you can think of that may have been used. And, without that passphrase (and access to the secret key), generating a revocation certificate is not possible. Hence the importance of generating one before it's too late!

Encryption & Digital Signatures

Since GnuPG uses the dual-key concept for increased security, how can this be used in your personal or business life? Let's take a look at the various areas where strong encryption and authentication could apply.

Asymmetric Encryption – Asymmetric encryption is the standard with dual-key concepts. As explained above, anyone can use your public key to encrypt information, but only you can decrypt it. You might be working on a project meant for "limited eyes only." Protecting the data generated in that project via encryption could keep competitors from uncovering, and possibly beating you to market, with your project's outcome. Say it's a programming project for a new "killer application" that will revolutionize the personal computer industry, a new product-packaging concept that will reduce costs for your company, or even a new and creative marketing scheme that will rake in big bucks. It would be a devastating blow to your project should your competitor catch wind of it before its time, so encrypting the files and communications relating to that project would definitely help keep them out of the loop.

Aside from other business data such as financial reports, projections, or statistics that might serve well being encrypted before being sent off over the Internet, as also mentioned it doesn't have to be "secret" data that should be kept from prying eyes. It could be common data that you or I wouldn't even consider the need to secure, but with today's problems with identity theft, burglaries and crime on the rise, perhaps we should take a second look at some of this "common" data we so easily send off without a second thought.

Phil Zimmermann likened sending plain-text email to a postcard. Actually, I think it's worse. I say this in consideration that at least those working for the US Postal Service are government-regulated employees. Surely those that consider government "big brother" might disagree, but, having a career in the computer industry and not having known too many postal workers personally, I have met far more network and ISP technicians that I wouldn't trust with a ten-foot pole. I would dread the thought of having them know I was taking my family on a vacation on a particular date and wouldn't be back for a week... I might arrive home and find it empty! If it's none of their business, encrypt it!

The general command to encrypt a file using another's public key is:

```
$gpg --recipient username --encrypt filename
```

As an added measure of security explained later in the article, when encrypting a file to be sent to another GnuPG/PGP user, most prefer to also sign the file. The command for this would be:

```
$gpg --recipient username --sign --encrypt filename
```

The result of either of these two commands will encrypt a file ("`filename`") to a particular user's ("`username`") public key, producing a binary encrypted file ("`filename.gpg`"). If the '`--sign`' command was given, the encrypted file would incorporate a signature produced with your private key as well.

Symmetric (conventional) Encryption – Symmetric, or conventional, encryption is where encrypted data needs only one key to decrypt it, the same key it was encrypted with. The cipher used to encrypt the document or file is generated at the time of encryption, and during the process you will be asked for a passphrase. The passphrase should be different from the one you normally use with your secret key, particularly if the file will be passed on to another. You will still be faced with finding a secure way of providing the passphrase to the recipient, but if that person does not yet utilize a public-key system such as that provided by GnuPG or PGP, all that's necessary to decrypt this file with either of those utilities is the password or phrase used to encrypt it with.

Another use for symmetric encryption might be sensitive documents or files kept on your own system. Although it's advised that any truly sensitive data should not even be kept on a system accessible to the Internet, no doubt there's still data on your system that would best be protected if encrypted.

Take for example all those Internet sites and web mail accounts you have passwords for. Some users opt for simple passwords; some just use one password for all. Neither of these is secure, so short of keeping a list handy, trying to remember all these passwords can be a daunting task. Many browsers nowadays have the ability to store passwords, but if you change browsers, wipe your cookies clean or suffer a hard drive crash without having a current backup, restoring more than a few of these passwords would be difficult for most anyone.

That's where having a document listing these passwords and account information can come to the rescue. You can still use strong passwords without the worry of forgetting them, yet at the same time you don't want this document sitting on your hard drive unprotected, or worse, printed and sitting next to your keyboard. Encrypt it! When you run into a situation where you need a password that you can't remember, it only takes a second to decrypt the file to the terminal screen, keeping it safe from prying eyes when not needed.

No doubt you will think of plenty of other data you might find on your hard drive that would be better left protected with strong encryption, and especially if that hard drive is on a laptop that travels... it's bad enough if the laptop gets stolen, but in many instances the loss of sensitive data kept on the laptop can be worth more than the hardware by far!

To encrypt a file named "`filename`" using symmetric encryption, issue the GnuPG command:

```
$gpg --symmetric filename
```

A copy of the original file in encrypted form, "`filename.gpg`", will be generated.

Digital Signatures – Although digital signatures accompanying data don't yet carry the weight they deserve in most states, they can be of extreme value in so many instances that I'm actually surprised their use hasn't caught on more than they have already.

Using your secret key, you can have documents, files and messages "signed" and time-stamped to authenticate that they actually did come from you. If the signature verifies, the receiver can be assured that the data was not modified in any way, and has arrived just as it was meant to be. This can be valuable for authors submitting stories or articles to publishers, programmers submitting files to the public, or businesses sending contracts, purchase orders or invoices to their partners. Literally just about any time you would normally send such a document signed with your own signature, a digitally signed document offers the same validity and authenticity in many cases, and prevents tampering to boot.

GnuPG's flexibility in signing permits signatures to be embedded in text messages as "clear-signed". Since the signature appears as ASCII text, this is useful for emails. The recipient can verify the signature to guarantee that the email has not been altered between the time it was sent or posted and the time it was received if they wish, but the message remains readable by all. To clear-sign a file named "`message`", issue the GnuPG command:

```
$gpg --clearsign message
```

The resulting file, "`message.asc`" can then be easily emailed as an attachment, or the contents included in the body of an email or posted to a newsgroup forum.

For an example of clear-signed messages you can visit one of the GnuPG newsgroup forums, such as gnupg-users, where you'll find most every message signed by the GnuPG user submitting the post. Clear-signed signatures are often referred to as ASCII-armored signatures. I'll be covering ASCII-armoring in the next section.

A digital signature can also be included along with the document as a single file; the file is compressed along with the included signature. Although the file's not actually encrypted, by using the GnuPG decrypt command and the sender's public key the file can be uncompressed, restored to its original format and the signature verified. The signature can also be verified without decompression in case you wish to keep the binary file intact as is.

Signatures can also be included as a detached file separate from the original file. This accommodates others that might not be using GnuPG or PGP utilities, allowing the original document or file to remain "untouched." Yet, because the signature hash was created from that file, it can still be verified as being unmodified and the time stamp checked. This is popular with many binary files found on the Internet (though in the Linux world MD5 hash signatures are sometimes more common for this), or with documents where an embedded signature is not desired. In order to verify a signature, all anyone needs is that person's public key.

To create a detached binary signature ("`filename.sig`") for the file "`filename`", the GnuPG command would be:

```
$gpg --detach-sign filename
```

Because signatures carry weight and can verify that the data hasn't changed, as previously mentioned it is strongly suggested that anytime you encrypt a file with another's public key, the file is also signed using your secret key prior to encryption. This adds protection to the file in that

not only can that particular recipient be the only one to decrypt it, but during the process they are also assured that it came from you, and only you. The added authenticity helps thwart attempts by someone else trying to intercept and modify the file for nefarious means, as without your secret key there would not be a way to reapply the signature. To decrypt the file and make changes they would not only need the secret key and passphrase of the recipient, but to reapply the signature they would also need yours... very unlikely.

To verify a signature or a signed file, the GnuPG '`--verify`' command is used. If the file includes an embedded signature, it could be as simple as:

```
$gpg --verify signedfile
```

Or, if a detached signature accompanies a file, the command would include both the signature file, and the file signed, such as:

```
$gpg --verify sigfile.asc signedfile
```

ASCII Armor

Although many applications support MIME/PGP standards and work very well with public-key encrypted or signed files in binary format, there may be some instances where ASCII format files are needed. Many email clients such as *Mutt*, *Sylpheed* and *KMail* for example fully support MIME/PGP standards and work very well with GnuPG, but for those MUAs lacking full support, even though there's very likely a plug-in available, ASCII armor comes to the rescue.

A good example of situations where ASCII format files might be needed is email or newsgroup postings. You want your messages signed, permitting those using public-key utilities to authenticate your signature, yet still allow those that don't to easily view the message's content. Or, perhaps your company's policies prohibit binary attachments passing through their email servers to or from outside networks, requiring any binary files to first be converted to ASCII, similar to uuencoded files.

GnuPG takes into account these needs by allowing the user to specify that an ASCII armored file be produced for its output. The option for this is '`--armor`', which can be added to most any command where ASCII format is needed. Another command-line option you will often use in conjunction with '`--armor`' is '`--output`', which takes a filename as an argument.

You may have also seen public keys displayed as "blocks" of garbled ASCII text on web sites. This makes it convenient to display the user's public key so visitors to the site can easily import the key into their own keyring. As you would expect, just about any output GnuPG generates, whether it be keys, encrypted files or signatures, can be output as ASCII-armored text. A few examples of producing ASCII-armored output with GnuPG are included below.

To encrypt the file "`filename`" for recipient "`username`", creating an armored ASCII text file ("`filename.asc`"):

```
$gpg  --armor --recipient username --encrypt filename
```

To encrypt the file "`filename`" for recipient "`username`", creating a signed armored ASCII text file ("`filename.asc`"):

```
$gpg --armor --recipient username --sign --encrypt filename
```

To encrypt the file "filename" using a symmetric cipher for encryption (does not include public key information), and creating an armored ASCII text file ("filename.asc"):

```
$gpg --armor --symmetric filename
```

To digitally sign the file "message", creating an ASCII-armored version ("message.asc") of the file that contains the attached signature:

```
$gpg --armor --sign message
```

To create an ASCII-armored text file named "sigfile.asc" as output for the detached signature, and is for the file "filename", you could use the command:

```
$gpg --armor --output sigfile.asc --detach-sign filename
```

Keyrings & Keyservers

As you've seen, the concept behind GnuPG and public-key security deals with keys, and keyrings. When you created your first pair of keys, you also created your first keyrings. If you take a look in your home directory, you'll find a hidden directory created for the GnuPG files (~/.gnupg). Within this directory there will be two keyring files, pubring.gpg and secring.gpg. The pubring.gpg file is your public keyring, and secring.gpg is your secret, or private, keyring.

Your secret keyring, containing only the one or two secret keys you create will remain small, however the public keyring will grow as you collect and add more public keys to it. Your secret keyring should be stringently protected at all costs, but your public keys, and keyrings if desired, can be shared.

As you continue working with GnuPG, you will note that many users publish their public keys on web sites or key servers. In the past, keys were exchanged via email, BBS or newsgroups postings. In recent years as the popularity of public-key security has gained ground, formal key servers have evolved to handle the need of dispersing these keys in a more convenient and centralized manner. After you've created the keys you'll be happy with and have worked with GnuPG to get to know your way around it a bit (and thoroughly memorized your passphrase!), you'll eventually want to start exchanging your public key(s) with others.

You'll also find various public keyrings dispersed on the Internet. Many of these keyrings hold the public keys of GnuPG/PGP users belonging to special interest groups. As you experiment with your secret and public keyrings, you may consider importing several keys from one of these keyrings to see what they're comprised of. You will likely find keys that have been signed by several other key holders, include additional email addresses, or even identifying photos. Since you may not actually know any of these people, you will probably limit your experiments to signing and encrypting files to your own key sets, but importing a few "popular" keys from one of these keyrings will give you an idea of what yours can become like down the road! When you're done, they can be easily deleted from your keyring if you choose.

Say for example you find a user's key available for download on the Internet that you want to import into your own public keyring, and the key is named "user-key.asc". After downloading the file, to import this public key into your keyring you would use the command:

```
$gpg --import user-key.asc
```

Likewise, to make an ASCII-armored copy of your public key that can be emailed, included as a download link, or whose contents can be displayed on a web page so others can import it into their own keyring, use the command:

```
$gpg --armor --export username > keyname.asc
```

This will export your key in ASCII armor format as previously explained, for the username or key ID specified, redirecting the output to the file "`keyname.asc`".

Key servers, or "keyservers" as they are sometimes referred to, have simplified the dispersing of public keys to the masses as it provides a somewhat centralized repository to post and retrieve public keys from. Although there are several key servers out there, most all are connected and synchronize their data. If you post or update your key to one of these servers, you'll find that the key will be broadcast and updated to the other key servers, sometimes rather quickly.

A couple of the more popular key servers are MIT's PGP Key Server and the OpenPGP Key Server. For other key servers closer to your geographic location, you might take a look at the OpenPGP Server Lookup web site. There, you'll find listed other key servers located around the world.

It's at these key servers that you can search for a public key belonging to someone by their name, email address or key ID. Keyservers are also good to use when verifying a key's fingerprint. If you were given a key by someone, or download one off the Internet from a web site, checking that key's fingerprint against the one listed on a keyserver to make sure they match is a good precaution to take as it will minimize the risk of receiving a bogus key. That's not to say that all keys on a keyserver are legitimate, but due to their circulation bogus keys can often be exposed for what they are.

Importing public keys from a key server into your keyring is as simple as specifying the key to import, and the keyserver to import it from. For example, the command:

```
$gpg --recv-keys --keyserver wwwkeys.pgp.net user@some-email.com
```

will import from the keyserver, "`wwwkeys.pgp.net`", the key for "`user@some-email.com`" into your keyring. Exporting a key is just as easy. The command:

```
$gpg --send-keys --keyserver wwwkeys.pgp.net user@some-email.com
```

will export the key for "`user@some-email.com`" currently residing in your public keyring to the keyserver "`wwwkeys.pgp.net`".

However, not everyone wishes to have his or her public key uploaded to a key server. So, if another user gives you their key for signing, always check with them prior to submitting it to one of the key servers. They may prefer that you email it back to them in a signed and encrypted message instead.

Web of Trust

Now that we've covered creating a key pair for your encryption and authentication needs, and have experimented a little with importing and exporting keys for yourself to use as a learning

experience, you're probably eager to start exchanging keys with other users. One of the fundamental aspects of using public-key cryptology with GnuPG/PGP is the ability for users to "sign" other users keys, adding yet another layer of legitimacy to that key. This also allows you to detect any tampering with the key in the future. However, signing another's key(s) is something that should not be taken lightly as it will affect the level of trust others have established in your key.

By having others sign your key, they are helping you establish what has become known as a "web of trust." Having several signatures on your key helps assure others receiving your key of its authenticity, especially if the key receiver also knows one or more of the others that has signed your key.

Keep in mind that when someone signs your key, they are only vouching for you in that your key is authentic and that it indeed belongs to you, because they have personally verified the fact. They are not vouching for your personality, ethics or morals, only that you are who you say you are identity-wise, and the key they've signed is yours.

Before you go signing another's key, it's important that you first verify that the key you were emailed, downloaded from a key server or were given actually belongs to that person. If you were to do this over the phone, or even in person, checking each character of the person's ASCII-armored key block would not only be exhausting, but extremely error prone as well. As such, each key has a "fingerprint." These fingerprints are derived from a hash of the key block, and provide a relatively sure way of verifying the key to the owner without error.

To view a key's fingerprint, the GnuPG command '`gpg --fingerprint keyID`' is used. The output displayed can then be used to communicate the received key's fingerprint to the owner, who can then verify that the fingerprint is correct, and matches the fingerprint of their key. An example of a key's fingerprint, as output by GnuPG, might look similar to:

```
pub  1024D/F357CB52 2002-07-09 GnuPG User <user@some-email.com>
     Key fingerprint = 5172 708D CA58 C2D9 4082  BA0D 103D 1293 F357 CB52
sub  2048g/A9CB69B3 2002-07-09
```

Once you've confirmed that the fingerprints match, you can then sign that user's public key with your secret key to validate it. When you created your key pair and a public key was formed, it was automatically signed by your secret key when using GnuPG. To sign another's key however, you can use the command:

```
$gpg --sign-key keyID
```

Along with '`--sign-key`', one other command for signing keys is available that I want to bring to your attention, and that is '`--lsign-key`'. The '`--lsign-key`' command signs the public key for local use only (instead of flagging your signature as being exportable along with the key as the '`--sign-key`' command). As your signature is not exportable, you're not validating that user's key for others to rely on, but it will still be treated as a valid key on your local system.

Why would you sign a key for local use only? Perhaps one example would be that you imported a key used to verify the signature of a software package. You want to sign the key so that the risk of tampering with this key without your knowledge is minimized, and warnings issued by GnuPG about using a non-validated key won't be displayed. The command to locally sign a key is:

```
$gpg --lsign-key keyID
```

You will be prompted for your passphrase, after which your signature will then be attached to that key. To sign a key you could also edit that key. The GnuPG command to edit keys is:

```
$gpg --edit-key keyID
```

This invokes the interactive menu system that allows you to modify the properties of a particular key. To sign the key with your secret key, in turn allowing your signature to be exported, use the menu command '`sign`'. Just as when using the '`--sign`' and '`--lsign`' commands from the shell's command prompt, you will be asked for your passphrase. When correctly supplied, it will attach your signature to the chosen public key when you then use the '`save`' command, or '`quit`' and respond that you want the changes saved.

The interactive menu system used when editing keys gives you a greater degree of flexibility when signing a key. For example, if you have a key that contains more than one user ID, you may want to sign only one or more IDs for that key, but not all of them. You might choose to do this if you weren't able to verify the email address associated with that ID, or want to keep the size of your keyring smaller than it would be if your signed keys had your signature attached to every ID (the key will still be considered valid, and as your keyring grows, signing only one ID on locally signed keys helps keep their size down). When you first display a key with '`--edit-key`', you will see something similar to:

```
pub  1024D/2B94CF89  created: 2001-01-01 expires: never      trust: -/q
sub  1024g/12E688D4  created: 2001-01-01 expires: never
(1)  Close Friend (GnuPG Gnut) <close.friend@some-email.com>
(2). Close Friend (Webmaster) <close.friend@someother-email.com>

Command>
```

The period after "`(2)`" indicates that this ID is the primary user ID. If you were to sign the key now, your signature would also be attached to that user ID as well. To mark a different ID, or several IDs for signing, the menu option '`uid`' followed by the chosen number is used. Once marked, the ID number will have an asterisk placed after it. If you were to mark the first ID to attach your signature to, the resulting output would look like:

```
pub  1024D/2B94CF89  created: 2001-01-01 expires: never      trust: -/q
sub  1024g/12E688D4  created: 2001-01-01 expires: never
(1)* Close Friend (GnuPG Gnut) <close.friend@some-email.com>
(2). Close Friend (Webmaster) <close.friend@someother-email.com>
```

Signing the key at this point with the '`sign`' menu command attaches your signature to the chosen ID. The menu command '`save`' will then save the signed key and exit, or you could use the menu command '`quit`' to exit without saving any changes.

Other commands you'll find useful in the interactive menu for editing keys are '`fpr`' to list a key's fingerprint (which you'll do when verifying the key with the owner), '`check`' to list the signatures on a key, '`adduid`' and '`deluid`' to add or delete a user ID on a key (name, comment or email address, which comes in handy if your email address changes), and '`trust`', which is used to assign a trust value to a key. For a list of commands that can be used with this interactive menu, simply type the word '`help`'. After pressing the [Enter] key, a list of possible commands will be displayed.

When should you use the '`lsign`' command in place of the '`sign`' command? Anytime you cannot refutably verify that the key you're signing belongs to its owner! This is perhaps the most vitally

important safeguard you can take in keeping your "web of trust" sound. Unless you have personally checked and verified the identity of the person against the key you were given with complete confidence, or one of your trusted fellows has (more on this below, in "Level of Trust"), don't sign it with an exportable signature! You may have found Werner Koch's (the main developer of GNU Privacy Guard) key on the Internet, but unless you can personally can verify the key, don't sign it with an exportable signature. Doing so will only weaken the level of trust others have placed in you to vouch for another's key validity, in turn weakening your web of trust.

I would like to add that, while on the subject of building your web of trust, many "famous" people have their public keys posted on key servers and the Internet (Linus Torvalds, Philip Zimmermann, Richard Stallman, Wichert Akkerman, Theodore T'so, Eris S. Raymond, etc.). If not common sense, then at least politeness should dictate that unless you actually know this person or will be conducting business with them, don't disturb their peace by invading their personal life in an attempt to verify their key just so you can sign it with your exportable signature. I'm sure they're kept busy enough as it is.

How does your web of trust grow? By getting to know other GnuPG/PGP users, and spreading the value that public-key encryption and authentication can bring to a person's personal and business dealings. You could add your public key's fingerprint to your email signature line perhaps (which sometimes brings about questions as to what it is by unknowing recipients, giving you the chance to evangelize), or once you've created a key pair you're happy with, publish your public key to a key server, or post the key block on your web site.

Many computer user groups may also hold key signing events where their member's keys are checked for accuracy and the holder's identity verified with a photo ID or driver's license. The keys are then later downloaded or emailed to the participants, key fingerprints checked, signed and returned to the user or key server as requested. Check your local user groups to see if they're holding one in your area. Linux user groups in particular often hold key signing events, and even parties in some localities, where the participants are invited to mingle and get to know each other in a more informal setting after the verification process is over.

As well, the Internet has come to the aid of some as there are sites (Biglumber.com for example) that will post lists of users wanting to have their keys verified, as well as users willing to sign another's key. Checking the lists for your area might put you in touch with someone local so the verification process can be completed.

If GnuPG were to stop here however, your "web of trust" would really be no more than a "wagon wheel of trust", as the only trusted keys on your keyring would be those to which your signature is attached. As in real life, the concept of trust used with GnuPG extends beyond that. Say for example there was a local GNU/Linux convention that you attended with one of your good friends, Greg. At the convention the two of you run into a buddy of his, Tom, operating one of the booths and were introduced. You don't know Tom personally, but you *do* know Greg and trust that if he said this person was indeed Tom, you would have few doubts as to his identity.

GnuPG utilizes trust levels in much the same way. You may acquire keys for which you don't personally know the individual they belong to, yet due to the interaction with other keys on your keyring that are trusted, they may be "introduced" and therefore treated as valid to varying degrees. This is what establishes the "web" in the web of trust model. These keys may not be linked directly to your key, but by the interconnection with other keys on your keyring, they in fact become a part of your web.

Levels of Trust

Many new to public-key use are sometimes overwhelmed by the word "trust" when referring to the keys in their public keyring. In simpler terms, there are basically two types of trust available to you, "validity trust" and "owner trust".

Validity trust, also called calculated trust, is based on to what extent a key can be considered valid. When you sign a key it's considered fully valid. If you can't verify with absolute certainty that the key belongs to its owner, and the owner is who they say they are, you can either sign the key for local use only (so your signature is not exportable, but the key is still considered valid), or refuse to sign the key and allow the validity to be calculated. If you do verify the key as being authentic, you can sign the key with an exportable signature and the key is then not only considered valid by you, but can also be treated as valid by others that have placed trust in you.

This type of trust is referred to as owner trust, and as the name implies, has to do with a key's owner and is used to specify just how much trust you have in a person's key to validate, or introduce, other keys on your keyring. As mentioned, this is because although a person may not be able to personally verify the validity of another's key, GnuPG allows trusted keys to validate to various extents other keys containing that trusted key's signature. The validity of these introduced keys depends on the amount of owner trust placed on the introducing key. A greater amount of owner trust yields a greater amount of calculated validity trust.

The default configuration for GnuPG specifies that if you haven't personally signed a key yourself, it can still be considered valid as long as two conditions are met. The first condition specifies that either the key has been signed with another key on your keyring that has been granted full owner trust, or the key has been signed by at least three other keys on your keyring that have been granted marginal trust. The second condition requires that the path of signed keys leading back to your own key (and inclusive of) include no more than six keys, or five "hops". These conditions can be altered in the GnuPG options, however, and can be made as tight (or flexible) as you wish them to be, but for most users the defaults are adequate and have withheld the test of time.

How do these levels of "owner trust" work? Let's say you have personally validated and signed Greg's key and have assigned full owner trust to that key. You've since imported Sandra's key. Although you can't possibly validate her key personally since she lives in a distant city (and you've never met or spoken with her before), as Greg has signed her key it will be considered fully valid.

In a perfect world, the owner of every key on your keyring would be fully trusted, but that just isn't the case. It would be nice, and is why I try to stress the importance of keeping your web of trust sound, but there will be keys on your keyring whose owner you simply might not want to place trust in either due to their lack of knowledge in dealing with public-key security, or the seriousness they place in trust values. There might also be keys belonging to people that you just don't know enough to grant trust to, or people you simply don't know at all. They're all a part of your web of trust however, so GnuPG addresses this by allowing you to assign levels of owner trust to each of your keys.

Assigning owner trust to a key on your public keyring uses the same interactive menu system that can used to sign keys ('`gpg --edit-key keyID`'), and there can be one of five levels of trust placed on a key. If you recall, when you first chose to sign a key with the interactive editing menu, you were displayed with something similar to:

```
pub  1024D/2B94CF89  created: 2001-01-01 expires: never      trust: -/f
sub  1024g/12E688D4  created: 2001-01-01 expires: never
(1). Close Friend (GnuPG Gnut) <close.friend@some-email.com>
```

The two types of trust are displayed on the right. The owner trust level is displayed first, and the validity, or calculated trust level is displayed second. In this example, a "-" appears for the owner trust level, and an "f" appears for the validity trust level, indicating the key has been signed, but no owner trust level has been established.

In dealing with owner trust, if the level is signified by a dash ("-"), it simply means that no level of trust has been assigned as yet. This is the default level placed on any new key imported into your keyring. If you don't know this person and have no level of trust to place on the key, simply leave this level of trust where it is. As there is no level of trust yet established, this key will never be used in validating another's key.

The second level is signified by an "n" and means no trust, or to *not* trust this person to correctly verify others signatures. Since these levels can be changed at any time you might assign this level to someone that is new to public-key use until they've grasped the hang of it. Then again, the key may belong to someone you know that is careless in the way they carry out key verifications and don't wish to have possible falsely validated keys pollute your web of trust. With a trust level of this grade, this key won't be used in validating another's key either.

The third level is signified by an "m" for marginal trust. Assigning marginal trust to a key indicates that you feel pretty confident that the key's owner understands the importance of proper key verification before placing their signature on a key. Bringing our example users back into the picture again, Greg mentioned to you that he has assigned full owner trust to Sandra's key on his keyring. You trust Greg's judgment of course, but since you don't personally know Sandra, her key might be a good candidate for marginal trust consideration until she's proved herself otherwise. With a trust level of marginal, if your public keyring contains a key, which in turn has been signed by at least three other keys on your keyring of marginal standing, that key will be considered valid. Quite often, the majority of the public keys on your keyring will likely be granted this level as your web of trust grows and your keyring expands.

The fourth level, signified by an "f" for full trust, is the level you would give to a key who's owner you have no doubts about their understanding of public-key use and verification checks, and is someone who's trust you would consider as good as your own. Our friend Greg would be a good example of someone worthy of full trust. He's been using GnuPG or PGP for quite a while, has established a good web of trust, and definitely takes every precaution in validating keys so as to not spoil that web of trust. With a full owner trust level, any key on your public keyring signed by this person's key will be considered fully valid.

The fifth level, signified by a "u" for ultimate trust, is used only for keys in which there is a corresponding secret key available. In other words, *your* personal public keys will have the status of ultimate trust when placed on your keyring. Needless to say, any key on your public keyring signed by you is considered a fully valid key, and is why it's important that you properly validate keys via proper methods before you sign them. Get carried away by signing anyone and everyone's key without properly verifying them first and you will quickly pollute your web of trust. I'm sure I've said that enough so I'll leave that topic alone now.

GUI Front Ends

Since GnuPG is command-line oriented (which actually adds flexibility to the utility) some users may be disappointed or miss a graphical user interface. This may especially hold true for newer Linux users transitioning from a graphical desktop OS like Windows. Although there are graphical front-ends to the application similar to those in earlier PGP days, they are still in the development stage and may lack the "polish" you expect, or have grown accustomed to. Although several GnuPG interfaces exist, I will only touch on a few of them here.

GPA – GPA, or GNU Privacy Assistant, is actually being developed by the GnuPG organization. Utilizing GTK (the GIMP Tool Kit) for its interface's widgets, GPA allows you to encrypt/decrypt files, view your keys, signatures, sign keys, edit trust levels, and import/export your keys to a public-key server among other things. Like many GUI front-ends, GPA is early work in progress, yet the version I'm currently using, 0.4.3, seems to be very stable. One area that GPA currently lacks is a system for the application's Help button (it's unresponsive). The interface is very intuitive however, so this certainly isn't a major drawback, but indicates there is still work to be done.

Seahorse – Seahorse provides a graphical interface much like GPA, with perhaps one of the nicest graphical front-ends to GnuPG that I've run across so far. Still very early in the development stage and currently at version 0.5.0, this GUI front-end provides much of the same functionality of GPA. Although the interface is very appealing to me, I did find that in its current state the application does suffer from a rare lockup now an then when used on my Debian 2.2 and Red Hat 7.1 systems. No data is corrupted, and killing the application is easy, but like most early projects this is to be expected. The author has recently announced a request to take over the Seahorse project, as development has stagnated for over a year now. Hopefully, someone will come to this worthwhile application's rescue.

TkPGP – Another nice GUI interface for GnuPG. TkPGP is written with Tcl/Tk, and also allows you to sign and encrypt documents in a text window, select files for encryption, decryption, signing and verification, but provides minimal key management functions. The application is capable of using a sizable number of GnuPG commands and configuration settings, providing a powerful interface to many of the GnuPG commands and options available, and includes the ability to remember your passphrase for a set time (handy when you're working with GnuPG for an extended period).

Other graphical front-ends exist for GnuPG and are worthy of consideration, so be sure to check into them before settling on one or two you like. There are also many character-based wrappers, written in Perl for example, that provide front-ends to this great utility. If you're operating at a character-based terminal and wanting a menu interface for GnuPG, be sure to check into these offerings as well. A rather comprehensive list of front-ends for GnuPG can be found at the GnuPG web site, or freshmeat.org.

Caveats

When working with sensitive data, whether you're using GnuPG or not, certain "common sense" caveats should apply. In other words, you should be aware if your sensitive data is winding up in shared memory locations, temporary files, etc., and whether you are working with your documents over a network or on shared systems. The same concerns apply when using GnuPG or any other public-key encryption system.

Keep in mind that simply deleting a file does not actually remove the file's contents from the hard drive, and that it can often be recovered with a little work. Also, if the application you're using to create a sensitive document uses "timed" saves or writes buffers to an unsecured location on your system, even after thorough deletion of the file there may be enough of the original document left behind in these areas to thwart your security efforts. Various file-wiping utilities exist for most platforms (such as 'shred' or 'wipe' for Linux), so if such a scenario applies to your situation, you should seriously consider looking into using one of these.

If you're using a shared terminal or operating over a network, you may consider using removable media such as a floppy or zip disk for your document's creation and encryption location... just don't leave it behind! You may also want to look into the features GnuPG has for encrypting text and messages straight from the terminal's keyboard. No plain-text file is left behind to worry about, as it's saved straight to an encrypted file.

Don't forget that your GnuPG secret keyring should be kept protected at all times. This file, `secring.gpg`, is kept in the `~/.gnupg` directory by default. If your terminal is shared or if your home directory is stored on a server somewhere, you'll definitely want to configure an alternate location for this file in your `~/.gnupg/options` configuration file. It could be kept on a floppy diskette, or even renamed and moved to another directory location if needed, as long as that location is secure.

For those that utilize shared terminals at work, a web café or truck stop, there's even a small distribution of Linux, Tinfoil Hat, developed especially to operate in conjunction with GnuPG. A bootable floppy that for the really paranoid even utilizes a wrapper for GnuPG called gpggrid to circumvent keystroke loggers, Tinfoil Hat Linux not only provides a reasonably secure place to store your keys and encrypted documents (reasonable as it's not handcuffed to your wrist), but also provides secure encryption and decryption of your files in RAM.

Keystroke loggers, or key loggers, can be software, or even small devices or plugs that attach between the system and the keyboard, logging every keystroke entered by the user (thereby capturing any text, including any passwords and passphrases that may have been entered). The hardware loggers, being small and usually attached at the back of a system, are many times overlooked and can go unnoticed. Software key loggers may save the keystrokes to a file, or transmit them over a network to another system.

If your data is sensitive, and your system accessible by others, be aware of the security measures that need to be taken into consideration in protecting your passphrase. A weak passphrase, or one that can be discovered easily, nullifies the security available with public-key encryption. The point is that your passphrase, as mentioned, is the weakest link in your encryption. Guard your passphrase well, and use common sense in keeping it safe or your efforts will be futile.

Also, if you haven't done so already, copy your keyring files to the same diskette as your key's revocation certificate and place it in a safe or deposit box for security's sake. Since the revocation certificate generated when you first created your key pair probably only includes "generic" comments explaining your reasons for revoking the key, keeping a backup copy of the keys allows you to generate a new certificate with details should your secret key ever become compromised or lost.

Conclusion

We've now covered your introduction into using GnuPG. As you have seen, it's not difficult to use, and in fact most of it is rather intuitive. When its features are used from within many other applications such as email clients or supportive applications or plug-ins, much of what goes on in the background is transparent, simplifying matters even more. The steps to using GnuPG are just as simple... come up with a good passphrase, create a key pair and its revocation certificate, experiment with it, publish your public key to a key server when you're satisfied, establish and build your "web of trust," and last but certainly not least, *always* keep security in mind.

Although the use of public-key security has grown tremendously since those early PGP days, it still has a long way to go to gain popularity in many countries, perhaps due to the overall sense of freedom those countries enjoy within their boundaries. With the tremendous growth of the Internet, thefts of corporate data and the proliferation of crackers and script-kiddies, those boundaries have dissolved. Once you've started using GnuPG with regularity you'll perhaps wonder what you did without it. With the numerous worries about the need for increased security defenses in businesses, corporations and homes, you may even find that it brings a bit of stress relief to your life!