DesktopLinux.com
. . . join the Desktop Linux revolution!

The Open Source Desktop Leader

home | news | articles | forum | polls | sponsors | search | directory | printable

Articles & white papers about using Linux on the desktop ...

Keywords: Match: Any keyword View Sort by: Newest first

An Introduction to GNU Privacy Guard (Part 1)

by David D. Scribner (September 23, 2002)

DesktopLinux.com guest author David Scribner has penned an article introducing new users to GnuPG on GNU/Linux (and UNIX) systems. Scribner focuses on how this powerful encryption package can play a vital role in personal and business communications by increasing security. This very detailed article will be available in two parts this week on DesktopLinux.com . . .

An Introduction to GNU Privacy Guard
by David D. Scribner
"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, 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

Ogg Vorbis

- ShaoLin Microsystems and SuSE announce strategic technology partnership
- Enjoying Music in Mandrake 9.0
- LindowsOS 3.0 release slated for Monday
- LinuxGuru: Structured Editing in Emacs
- KNOPPIX gets unoffical website with support forum
- LindowsOS Tablet PC to launch in early 2003
- DesktopLinux.com Poll -- November Update
- Tuning Red Hat for maximum performance
- Film Gimp - Lights, Camera, Linux!
- OpenOffice.org Project Update [Linux Journal]
- Sawfish 1.2 released
- Xandros Desktop -- Not Your Father's Linux Distribution [LinuxPlanet]
- Sun aims to oust MSOffice from UK schools [The Register]
- Red Hat, Oracle Team on Desktop Linux
- Ximian's new Evolution available
- KDE 3.1-RC2 Preview [OSNews]

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:

- Linux and Learning [eSchoolNews]
- 101 things that the Mozilla browser can do that IE cannot [XulPlanet]
- Concatenating files with 'cat' [developerWorks]

More . . .

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

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
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 previously explained, 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.

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.

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. Most messages are signed using this method. Clear-signed signatures are often referred to as ASCII-armored signatures. I'll be covering ASCII-armoring in the next section.

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 filename.asc
--detach-sign filename
```

Part Two will cover topics including Keyrings & Keyservers, Levels of Trust, and GUI-based utilities for harnessing GnuPG.

About the author: David Scribner has been an IT consultant since 1984. Scribner advises small and mid-size businesses on a wide range of services from training to technical services.

An earlier version of this article appeared at [Linuxguru.net]

Talk back! Do you have comments or questions about this story?
talkback here

. . . For further information