Secure Your Secrets with 4096-Bit Encryption

As you progress in the world of information security, you'll find yourself in situations where data protection is paramount. No doubt you will have files to hide and secrets to share, so I'm going to show you how to use the GNU Privacy Guard (GnuPG or GPG for short) to encrypt and decrypt as you need. GPG is a great open-source version of Pretty Good Privacy (PGP), a similar application used for encryption, but licensing and patent problems led to the development of GPG in its wake.

Obtaining GPG

GPG can be downloaded for free at its website, if you do not already have it on your system. As many Linux flavors already include it on a default installation, you can test this by opening up a terminal and typing **'gpg -h'**. If the help list scrolls across your screen, you are on the right track! If you receive a 'command not found' error, click the link above and follow the instructions for your operating system to get the latest version.

I will be talking about the command line options for GPG here, although several GUI interfaces exist such as KGPG for KDE and Seahorse for Gnome. Debian/Ubuntu users may **'sudo apt-get install xxx'** to obtain these packages as well. Windows users may download GPG4win.

Getting Started

The basic idea of public-key cryptography is that you have two keys. One key is your public key and one is your private key. When you encrypt a message to send to someone, it uses the public key of that person in the actual algorithm. When they receive the message, they use their private key to decrypt and read it. Because the private key can not be determined from the public key, you can post and share your public key with anyone who needs it.

Step 1 Choosing a Key

Open up a terminal window and type **'gpg –gen-key'** and you should see:

Please select what kind of key you want:

1. **RSA and RSA (default)**
2. **DSA and Elgamal**
3. **DSA (sign only)**
4. **RSA (sign only)**

Choose option **1** to allow for both encryption and signing.

Step 2 Choosing a Key Size

What key size do you want? (2048)

The default is 2048 bits long. As a default, this value is alright, but the larger your key the harder it will be to break. Also, increasing this value increases the computing cycles to create it, but as I said, we want max protection, so let's go ahead and set this to **4096**. It would take the combined processing power of every computer in the world thousands of years to crack 4096-bit encryption.

Step 3 Choosing Its Expiration

Please specify how long the key should be valid.

- **0 = key does not expire**
- **<n> = key expires in n days**
- **<n>w = key expires in n weeks**
- **<n>m = key expires in n months**

- **<n>y = key expires in n years**

This is a personal choice. You are setting a timer on your key-set. After X amount of time from the creation date, the keys will no longer be valid and you will need to create a new set. I personally set this value to one month, but you can choose whatever you like here.

Step 4 Create a User ID

You need a user ID to identify your key; the software constructs the user ID from the Real Name, Comment and Email Address in this form:

"Heinrich Heine (Der Dichter) <heinrichh@duesseldorf.de>"

Here, as above is more of a personal choice. You can fill these fields out depending on your use. If this is for business, you might want your real name and email, if not, well... be creative.

If you entered the data as follows:

Real name: Allen Freeman / Email address: Allen@nullbyte.com / Comment: me

In this case, your ID would be **Allen Freeman (me)**, and this would be what you would use to address messages after you've imported someone's public key. That's a topic we'll discuss in the next article.

Note: That is not a valid email address for me.

Step 5 The Achilles Heel — Your Passphrase

You need a passphrase to protect your secret key.

This is the critical part. The weakest link in the entire process is your passphrase (not 'word', but 'phrase', because it allows spaces). If you happen to forget it, your key is useless, and if it's simple and easily broken, even more more so. I recommend to my clients a passphrase NO LESS then ten characters, alternating between case, numbers and symbols. If your passphrase looks like a Perl statement, you are doing it right.

After this is finished, GPG will need to generate entropy to create your key. In a (very small) nutshell, it's using the "random noise" of your computer, such as files opening and closing, processing, memory swapping, etc. to create random values. This might

take a few moments and you might be told you need to generate more entropy. Just wiggle your mouse around and bang some keys on the keyboard until it lets you know it has finished.

pub 2048R/D7FC65D2 created: 2012-02-20 expires: never usage: SC

trust: ultimate validity: ultimate

sub 2048R/389AA63E created: 2012-02-20 expires: never usage: E

[ultimate] (1). Allen Freeman (Comments go here) <allen@nullbyte.com>

When you are finished, your user ID is the second string after the slash in the public list (pub 2048R/D7FC65D2 – so D7FC65D2 would be your UID) and you now have your own set of encryption keys.

Step 6 Export the Public Key

One last step before you can share the public key—you need to export it. You need to type **'gpg -a –export [UID]'** and simply save the fingerprint that appears. All someone needs is that key-block to send you a file that only you can open.

Is that all? Not by any means.