Quick'n easy gpg cheatsheet

If you found this page, hopefully it's what you were looking for. It's just a brief explanation of some of the command line functionality from gnu privacy guard (gpg). Please email me if you find any errors (scout3801@gmail.com).

Filenames are italicized (loosely, some aren't, sorry), so if you see something italicized, think "put my filename there."

I've used User Name as being the name associated with the key. Sorry that isn't very imaginative. I *think* gpg is pretty wide in it's user assignments, ie. the name for my private key is Charles Lockhart, but I can reference that by just putting in Lockhart. That doesn't make any sense, sorry.

to create a key:
gpg --gen-key
generally you can select the defaults.

to export a public key into file public.key:
gpg --export -a "User Name" > *public.key*
This will create a file called public.key with the ascii representation of the public key for User Name. This is a variation on:
gpg --export
which by itself is basically going to print out a bunch of crap to your screen. I recommend against doing this.
gpg --export -a "User Name"
prints out the public key for User Name to the command line, which is only semi-useful

to export a private key:
gpg --export-secret-key -a "User Name" > *private.key*
This will create a file called private.key with the ascii representation of the private key for User Name.
It's pretty much like exporting a public key, but you have to override some default protections. There's a note (*) at the bottom explaining why you may want to do this.

to import a public key:
gpg --import *public.key*
This adds the public key in the file "public.key" to your public key ring.

to import a private key:
NOTE: I've been informed that the manpage indicates that "this is an obsolete option and is not used anywhere." So this may no longer work.
gpg --allow-secret-key-import --import *private.key*
This adds the private key in the file "private.key" to your private key ring. There's a note (*) at the bottom explaining why you may want to do this.

to delete a public key (from your public key ring):
gpg --delete-key "User Name"
This removes the public key from your public key ring.
NOTE! If there is a private key on your private key ring associated with this public key, you will get an error! You must delete your private key for this key pair from your private key ring first.

to delete an private key (a key on your private key ring):
gpg --delete-secret-key "User Name"
This deletes the secret key from your secret key ring.

To list the keys in your public key ring:
gpg --list-keys

To list the keys in your secret key ring:
gpg --list-secret-keys

To generate a short list of numbers that you can use via an alternative method to verify a public key, use:
gpg --fingerprint > *fingerprint*
This creates the file fingerprint with your fingerprint info.

To encrypt data, use:
gpg -e -u "Sender User Name" -r "Receiver User Name" *somefile*
There are some useful options here, such as -u to specify the secret key to be used, and -r to specify the public key of the recipient.
As an example: gpg -e -u "Charles Lockhart" -r "A Friend" mydata.tar
This should create a file called "mydata.tar.gpg" that contains the encrypted data. I think you specify the senders username so that the recipient can verify that the contents are from that person (using the fingerprint?).
NOTE!: mydata.tar is not removed, you end up with two files, so if you want to have only the encrypted file in existance, you probably have to delete mydata.tar yourself.
An interesting side note, I encrypted the preemptive kernel patch, a file of 55,247 bytes, and ended up with an encrypted file of 15,276 bytes.

To decrypt data, use:
gpg -d *mydata.tar.gpg*
If you have multiple secret keys, it'll choose the correct one, or output an error if the correct one doesn't exist. You'll be prompted to enter your passphrase. Afterwards there will exist the file "mydata.tar", and the encrypted "original," mydata.tar.gpg.

NOTE: when I originally wrote this cheat sheet, that's how it worked on my system, however it looks now like "gpg -d mydata.tar.gpg" dumps the file contents to standard output. The working alternative (worked on my system, anyway) would be to use "gpg -o outputfile -d encryptedfile.gpg", or using mydata.tar.gpg as an example, I'd run "gpg -o mydata.tar -d mydata.tar.gpg". Alternatively you could run something like "gpg -d mydata.tar.gpg > mydata.tar" and just push the output into a file. Seemed to work either way.

Ok, so what if you're a paranoid bastard and want to encrypt some of your own files, so nobody can break into your computer and get them? Simply encrypt them using yourself as the recipient.

I haven't used the commands:
gpg --edit-key
gpg --gen-revoke

- --gen-revoke creates a revocation certificate, which when distributed to people and keyservers tells them that your key is no longer valid, see http://www.gnupg.org/gph/en/manual/r721.html
- --edit-key allows you do do an assortment of key tasks, see http://www.gnupg.org/gph/en/manual/r899.html

Sharing Secret Keys

NOTE!: the following use cases indicate why the secret-key import/export commands exist, or at least a couple ideas of what you could do with them. HOWEVER, there's some logistics required for sharing that secret-key. How do you get it from one computer to another? I guess encrypting it and sending it by email would probably be ok, but I wouldn't send it unencrypted with email, that'd be DANGEROUS.

Use Case *.1 : Mentioned above were the commands for exporting and importing secret keys, and I want to explain one reason of why maybe you'd want to do this. Basically if you want one key-pair for all of your computers (assuming you have multiple computers), then this allows you export that key-pair from the original computer and import it to your other computers.

Use Case *.2 : Mentioned above were the commands for exporting and importing secret keys, and I want to explain one reason of why maybe you'd want to do this. Basically, if you belonged to a group, and wanted to

create a single key-pair for that group, one person would create the key-pair, then export the public and private keys, give them to the other members of the group, and they would all import that key-pair. Then a member of the group or someone outside could use the group public key, encrypt the message and/or data, and send it to members of the group, and all of them would be able to access the message and/or data. Basically you could create a simplified system where only one public key was needed to send encrypted stuffs to muliple recipients.