TUTORIAL

How To Use GPG to Encrypt and Sign Messages

Security Email

By Justin Ellingwood

Published on May 26, 2017 251.3k

Introduction

GPG, or GNU Privacy Guard, is a public key cryptography implementation. This allows for the secure transmission of information between parties and can be used to verify that the origin of a message is genuine.

In this guide, we will discuss how GPG works and how to implement it. We will be using an Ubuntu 16.04 server for this demonstration, but will include instructions for other distributions as well.

How Public Key Encryption Works

A problem that many users face is how to communicate securely and validate the identity of the party they are talking to. Many schemes that attempt to answer this question require, at least at some point, the transfer of a password or other identifying credentials, over an insecure medium.

Ensure That Only the Intended Party Can Read

To get around this issue, GPG relies on a security concept known as public key encryption. The idea is that you can split the encrypting and decrypting stages of the transmission into two separate pieces. That way, you can freely distribute the encrypting portion, as long as you secure the decrypting portion.

This would allow for a one-way message transfer that can be created and encrypted

by anyone, but only be decrypted by the designated user (the one with the private decrypting key). If both of the parties create public/private key pairs and give each other their public encrypting keys, they can both encrypt messages to each other.

So in this scenario, each party has their own private key and the other user's public key.

Validate the Identity of the Sender

Another benefit of this system is that the sender of a message can "sign" the message with their private key. The public key that the receiver has can be used to verify that the signature is actually being sent by the indicated user.

Set Up GPG Keys

GPG is installed by default in most distributions.

If for any reason GPG is not installed, on **Ubuntu** and **Debian**, you can update the local repo index and install it by typing:

```
$ sudo apt-get update
$ sudo apt-get install gnupg
```

On **CentOS**, you can install GPG by typing:

```
$ sudo yum install gnupg2
```

To begin using GPG to encrypt your communications, you need to create a key pair. You can do this by issuing the following command:

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

This will take you through a few questions that will configure your keys:

- Please select what kind of key you want: **(1) RSA and RSA (default)**
- What keysize do you want? **4096**
- Key is valid for? **1y** (expires after 1 year. If you are just testing, you may want to create

a short-lived key the first time by using a number like “3” instead.)

- Is this correct? **y**
- Real name: **your real name here**
- Email address: **your_email@address.com**
- Comment: **Optional comment that will be visible in your signature**
- Change (N)ame, ©omment, (E)mail or (O)kay/(Q)uit? **O**
- Enter passphrase: **Enter a secure passphrase here (upper & lower case, digits, symbols)**

At this point, `gpg` will generate the keys using entropy. *Entropy* describes the amount of unpredictability and nondeterminism that exists in a system. GPG needs this entropy to generate a secure set of keys.

This process may take a long time depending on how active your system is and the keysize you selected. To generate additional entropy more easily, you can use a tool called haveged. Open up a new terminal and SSH into the server again to set up `haveged` on your server.

Create a Revocation Certificate

You need to have a way of invalidating your key pair in case there is a security breach or in case you lose your secret key. There is an easy way of doing this with the GPG software.

This should be done as soon as you make the key pair, not when you need it. This revocation key must be generated ahead of time and kept in a secure, separate location in case your computer is compromised or inoperable. To generate a revocation key, type:

```
$ gpg --output ~/revocation.crt --gen-revoke your_email@address.com
```

You will be asked to confirm the revocation key creation and then prompted for the reason that it is being revoked. This information will be visible to other users if the revocation is used in the future. You can choose any of the available options, but since this is being done ahead of time, you won’t have the specifics. Often, it is a good idea to create a revocation certificate for each of the likely scenarios for maximum

flexibility.

Afterwards, you will then be asked to supply a comment and finally, to confirm the selections. Before creating the revocation certificate, you will need to enter your GPG key's passphrase to confirm your identity. The revocation certificate will be written to the file specified by the `--output` flag (`revocation.crt` in our example):

```
Output
Revocation certificate created.

Please move it to a medium which you can hide away; if Mallory gets
access to this certificate he can use it to make your key unusable.
It is smart to print this certificate and store it away, just in case
your media become unreadable.  But have some caution:  The print system of
your machine might store the data and make it available to others!
```

You should immediately restrict the permissions on the generated certificate file in order to prevent unauthorized access:

```
$ chmod 600 ~/revocation.crt
```

The revocation certificate must be kept secure so that other users cannot revoke your key. As the message states, you should consider backing the certificate up to other machines and printing it out, as long as you can secure it properly.

How To Import Other Users' Public Keys

GPG would be pretty useless if you could not accept other public keys from people you wished to communicate with.

You can import someone's public key in a variety of ways. If you've obtained a public key from someone in a text file, GPG can import it with the following command:

```
$ gpg --import name_of_pub_key_file
```

There is also the possibility that the person you are wishing to communicate with has uploaded their key to a public key server. These key servers are used to house

people's public keys from all over the world.

A popular key server that syncs its information with a variety of other servers is the MIT public key server. You can search for people by their name or email address by going here in your web browser:

```
https://pgp.mit.edu/
```

You can also search the key server from within GPG by typing the following:

```
$ gpg --keyserver pgp.mit.edu  --search-keys search_parameters
```

You can use this method of searching by name or email address. You can import keys that you find by following the prompts.

How To Verify and Sign Keys

While you can freely distribute your generated public key file and people can use this to contact you in a secure way, it is important to be able to trust that the key belongs to who you think it does during the initial public key transmission.

Verify the Other Person's Identity

How do you know that the person giving you the public key is who they say they are? In some cases, this may be simple. You may be sitting right next to the person with your laptops both open and exchanging keys. This should be a pretty secure way of identifying that you are receiving the correct, legitimate key.

But there are many other circumstances where such personal contact is not possible. You may not know the other party personally, or you may be separated by physical distance. If you never want to communicate over insecure channels, verification of the public key could be problematic.

Luckily, instead of verifying the entire public keys of both parties, you can simply compare the "fingerprint" derived from these keys. This will give you a reasonable assurance that you both are using the same public key information.

You can get the fingerprint of a public key by typing:

```
$ gpg --fingerprint your_email@address.com
```

```
Output
pub   4096R/311B1F84 2013-10-04
      Key fingerprint = CB9E C70F 2421 AF06 7D72  F980 8287 6A15 311B 1F84
uid                  Test User <test.user@address.com>
sub   4096R/8822A56A 2013-10-04
```

This will produce a much more manageable string of numbers to compare. You can compare this string with the person themselves, or with someone else who has access to that person.

Sign Their Key

Signing a key tells your software that you trust the key that you have been provided with and that you have verified that it is associated with the person in question.

To sign a key that you've imported, simply type:

```
$ gpg --sign-key email@example.com
```

When you sign the key, it means you verify that you trust the person is who they claim to be. This can help other people decide whether to trust that person too. If someone trusts you, and they see that you've signed this person's key, they may be more likely to trust their identity too.

You should allow the person whose key you are signing to take advantage of your trusted relationship by sending them back the signed key. You can do this by typing:

```
$ gpg --output ~/signed.key --export --armor email@example.com
```

You'll have to type in your passphrase again. Afterwards, their public key, signed by you, will be displayed. Send them this, so that they can benefit from gaining your "stamp of approval" when interacting with others.

When they receive this new, signed key, they can import it, adding the signing information you've generated into their GPG database. They can do this by typing:

```
$ gpg --import ~/signed.key
```

They can now demonstrate to other people that *you* trust that their identity is correct.

How To Make Your Public Key Highly Available

Because of the way that public key encryption is designed, there is not anything malicious that can happen if unknown people have your public key.

With this in mind, it may be beneficial to make your public key publicly available. People can then find your information to send you messages securely from your very first interaction.

You can send anyone your public key by requesting it from the GPG system:

```
$ gpg --output ~/mygpg.key --armor --export your_email@address.com
```

Output

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v1.4.11 (GNU/Linux)

mQINBFJPCuABEACiog/sInjg0O2SqgmG1T8n9FroSTdN74uGsRMHHAOuAmGLsTse
9oxeLQpN+r75Ko39RVE88dRcW710fPY0+fjSXBKhpN+raRMUKJp4AX9BJd00YA/4
EpD+8cDK4DuLlLdn1x0q41VUsznXrnMpQedRmAL9f9bL6pbLTJhaKeorTokTvdn6
5VT3pb2o+jr6NETaUxd99ZG/osPar9tNThVLIIzG1nDabcTFbMB+w7wOJuhXyTLQ
JBU9xmavTM71PfV6Pkh4j1pfWImXc1D8dS+jcvKeXInBfm2XZsfOCesk12YnK3Nc
u1Xe1lxzSt7Cegum4S/YuxmYoh462oGZ7FA4Cr2lvAPVpO9zmgQ8JITXiqYg2wB3
. . .
```

You can then send this file to the other party over an appropriate medium.

If you want to publish your key to a key server, you can do it manually through the forms available on most of the server sites.

Another option is to do this through the GPG interface. Look up your key ID by typing:

```
$ gpg --list-keys your_email@address.com
```

The highlighted portion in the output below is the key ID (look for the `pub` along the left-hand column if you're uncertain about which one to use). It is a short way to reference the key to the internal software.

```
Output
pub   4096R/311B1F84 2013-10-04
uid                  Test User <test.user@address.com>
sub   4096R/8822A56A 2013-10-04
```

To upload your key to a certain key server, you can then use this syntax:

```
$ gpg --send-keys --keyserver pgp.mit.edu key_id
```

The key will be uploaded to the specified server. Afterwards, it will likely be distributed to other key servers around the world.

Encrypt and Decrypt Messages with GPG

You can easily encrypt and decrypt messages after you have shared your keys with the other party.

Encrypt Messages

You can encrypt messages using the "–encrypt" flag for GPG. The basic syntax would be:

```
$ gpg --encrypt --sign --armor -r person@email.com name_of_file
```

This encrypts the message using the recipient's public key, signs it with your own private key to guarantee that it is coming from you, and outputs the message in a text format instead of raw bytes. The filename will be the same as the input filename, but with an `.asc` extension.

You should include a second "-r" recipient with your own email address if you want to

be able to read the encrypted message. This is because the message will be encrypted with each person's public key, and will only be able to be decrypted with the associated private key.

So if it was only encrypted with the other party's public key, you would not be able to view the message again, unless you somehow obtained their private key. Adding yourself as a second recipient encrypts the message two separate times, one for each recipient.

Decrypt Messages

When you receive a message, simply call GPG on the message file:

```
$ gpg file_name.asc
```

The software will prompt you as necessary.

If instead of a file, you have the message as a raw text stream, you can copy and paste it after typing `gpg` without any arguments. You can press "CTRL-D" to signify the end of the message and GPG will decrypt it for you.

Key Maintenance

There are a number of procedures that you may need to use on a regular basis to manage your key database.

To list your available GPG keys that you have from other people, you can issue this command:

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

Your key information can become outdated if you are relying on information pulled from public key servers. You do not want to be relying on revoked keys, because that would mean you are trusting potentially compromised keys.

You can update the key information by issuing:

```
$ gpg --refresh-keys
```

This will fetch new information from the key servers.

You can pull information from a specific key server by using:

```
$ gpg --keyserver key_server --refresh-keys
```

You may receive error messages if any of your keys cannot be found on the key server.

Conclusion

Using GPG correctly can help you secure your communications with different people. This is extremely helpful, especially when dealing with sensitive information, but also when dealing with regular, everyday messaging.

Because of the way that certain encrypted communications can be flagged by monitoring programs, it is recommended to use encryption for everything, not just “secret” data. That will make it more difficult for people to know when you are sending important data or just sending a friendly hello.