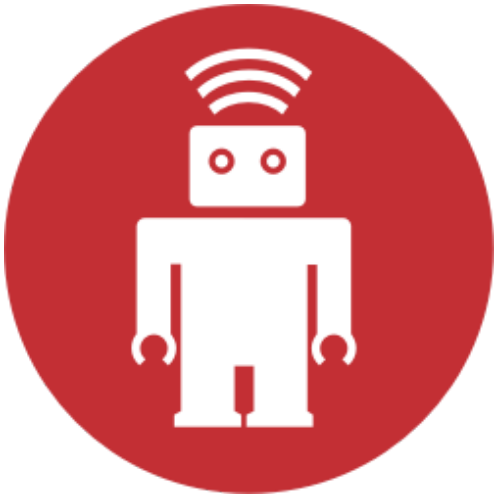
A pretty good introduction to Pretty Good Privacy

by **George Brocklehurst** (@georgebrock)



Presented at **New York Emacs Meetup**, **8th December 2014**

In this talk

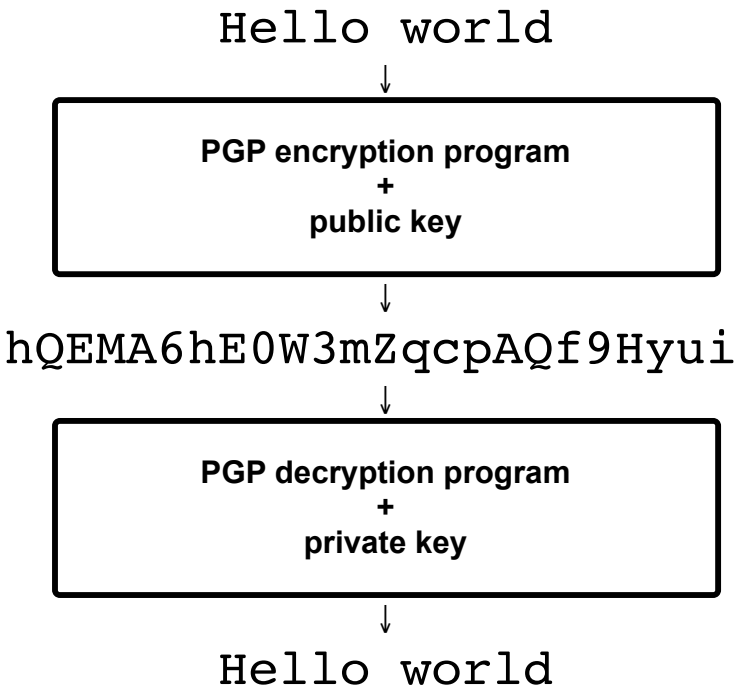
What is it?

Public key cryptography

- PGP uses public key cryptography.
- Users generate a *key pair*, consisting of a *private key* and a *public key*.
- The private key is kept private, while the public key is distributed on the Internet.
- The keys are linked: a document encrypted with one half of a key pair can only be decrypted with the other half.

Encryption

The first useful function PGP provides is *encryption*: if I have a user's public key, then I can encrypt a document so that only they can decrypt it.

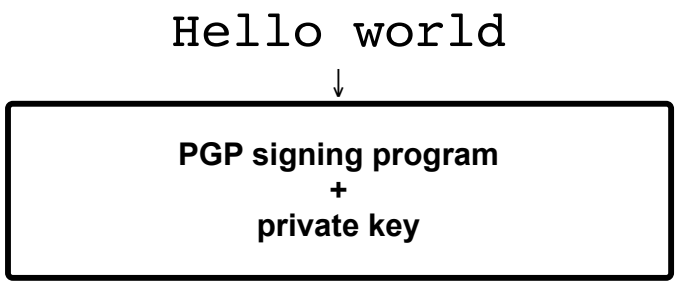


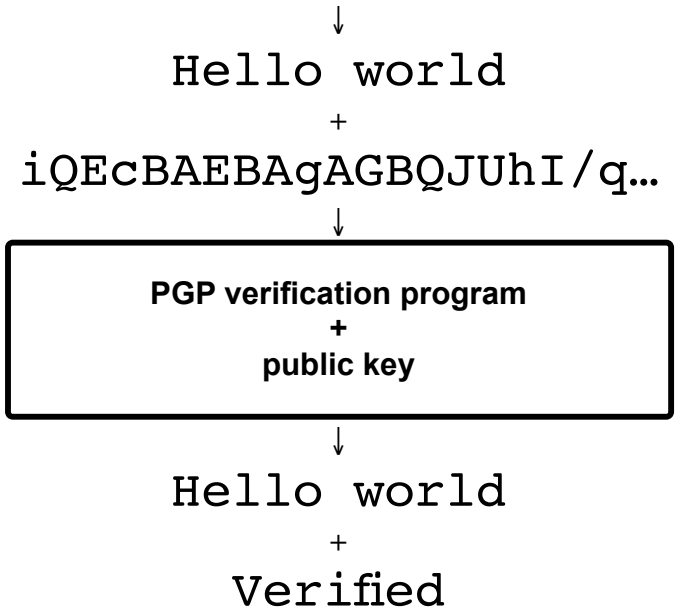
Signing

The second useful function PGP provides is *signing*: I can sign a message or document using my private key, and anyone with my public key can verify the signature.

A verified signature means two things:

1. The signature was made using my private key, and
2. the document has not been modified since it was signed.





Behind the scenes, signatures use encryption:

1. The signer generates a digest (e.g. SHA1) of the document
2. The digest is encrypted using the signer's private key, producing the signature
3. The verifier decrypts the signature using the signer's public key
4. The verifier generate a digest of the document and compares it to the decrypted signature
5. If the digests match, the signature is verified

Why should I care?

In other words: what practical uses do **encryption** and **signing** have in your every day interactions with computers?

Email security

- Encrypted email is a secure way to send secret information over the Internet, e.g. a password.
- Signed email allows you to verify the content of an email hasn't been tampered with, and who sent it.

Signed Git commits

- Git makes it very easy to commit under another user's name (this is a feature, not a bug).
- Signing commits makes it difficult for someone to sneak malicious code into your Git repository.
- Git servers can be compromised, e.g. Egor Homokov hacked GitHub.
- For more information, read Mike Gerwitz's "Git Horror Story"

Encrypted files

- As well as using PGP to send other people encrypted messages and files, you can use your own public key to encrypt files for your own use.
- Many PGP users store private information—things like passwords—in PGP encrypted files on their hard drive.

passwordstore.org: the standard unix password manager

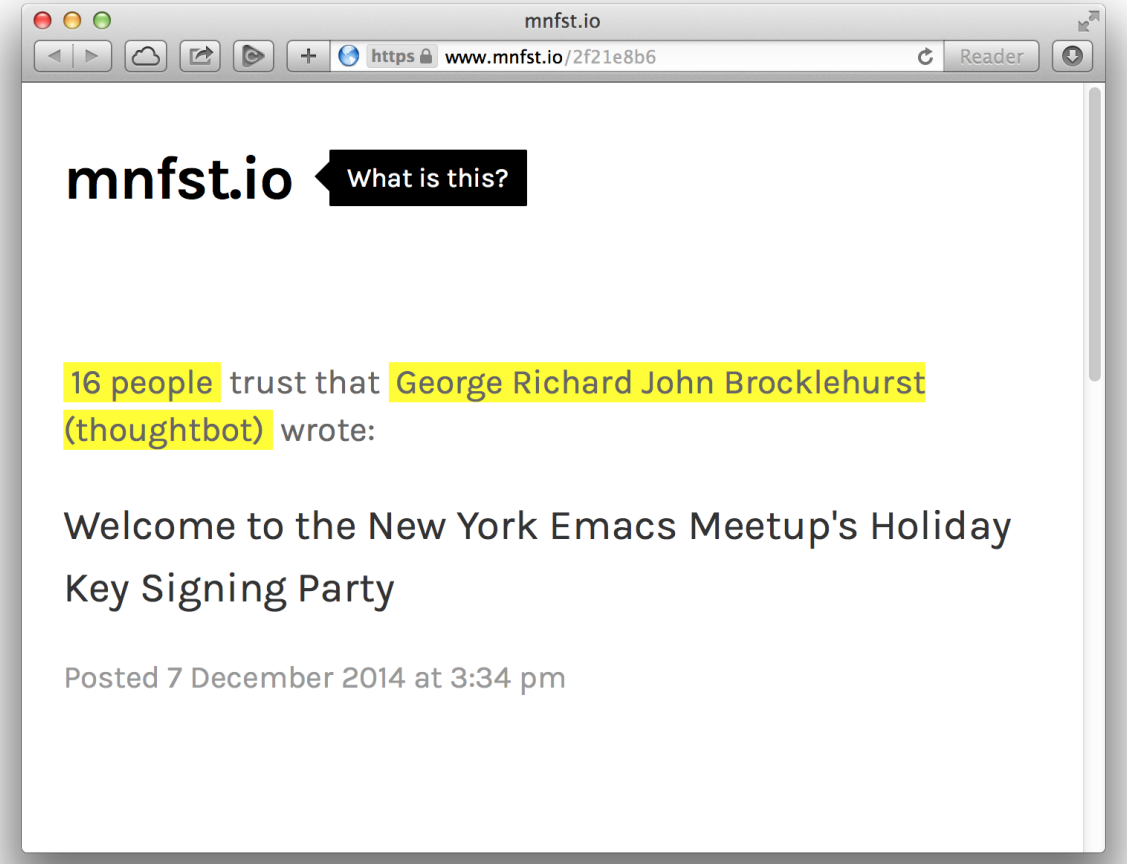
- The `pass` command-line utility provides convenient tools for storing your passwords in PGP encrypted files.

Verifiable downloads

- Verifying the SHA1 of MD5 of a download only goes so far: if the Web site hosting a download is compromised, the digest could have been substituted as well as the download.
- Verifying a download using a PGP signature is safer: if you know which key you expect to have created the signature, you can verify that the signature has not been tampered with.
- The Debian apt repository does this for you automatically.

Web sites without passwords

- Logging in to a Web site involves sending your identifying information—a username and password—over the network. Anyone who intercepts that message can impersonate you.
- Sending signed data is safer. The Web site can verify that the data comes from you without you having to send anything that can be used to impersonate you.
- Still experimental, but this demonstrates the power of PGP signatures as a general purpose identify verification mechanism.



mnfst.io: A password-less Web site built with PGP

The chicken, or the egg?

There's a problem with all of this, though: to send someone an encypted message, or verify their signed message, I need to know that I have their public key. Unfortunately, since I don't have their public key yet, I don't have a verifiable way to get their public key from them.

PGP provides a solution to this problem.

The Web of Trust

- The Web of Trust allows users to act as trusted introducers: if Ann trusts Brian, and Brian trusts Charlotte, then Ann can trust Charlotte to some degree.
- Trusting a PGP user is indicated by signing their public key with their private key, after checking that they are who they say they are, and that you have the public key that they claim to own.

- You can also indicate a trust level for users: if someone is *fully trusted* then their signature is enough to trust the keys they have signed, but if they are only *marginally trusted* then several other users have to agree before you trust the keys they have signed.
- Henk Penning has published some interesting tools for exploring the Web of Trust.

How do I use it?

PGP, OpenPGP & GnuPG

- PGP is the original program, written by Phil Zimmerman. It is free-as-in-beer, but not free-as-in-Stallman.
- OpenPGP is the standard derived from PGP. PGP is now just one of several implementations of OpenPGP.
- GnuPG is the Gnu project's free-as-in-Stallman implementation of OpenPGP. Since this talk is at an Emacs meetup, and this is the implementation I tend to use, this is the one I will demonstrate.

GnuPG demo

To follow these examples, you will need to have `gpg2` installed.

Generating a key

```
$ gpg2 --gen-key
gpg (GnuPG) 2.0.26; Copyright (C) 2013 Free Software Foundation, Inc.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

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)
Your selection?
```

You probably want to stick with the default here: just hit `return`

```
RSA keys may be between 1024 and 4096 bits long.
What keysize do you want? (2048)
```

The key size is a balance between a large, secure key, and small, easily transmitted signatures and encrypted documents. Again, if you don't have a strong preference the default is probably fine.

```
Please specify how long the key should be valid.
         0 = key does not expire
        = key expires in n days
      w = key expires in n weeks
      m = key expires in n months
      y = key expires in n years
Key is valid for? (0) 2y
Key expires at Wed Dec  7 23:38:11 2016 EST
Is this correct? (y/N) y
```

In this case, it's useful not to follow the default. If you lose your private key for any reason, the the public key could live forever, inviting people to send you encrypted messages you can never decrypt. If the key expires, this problem will go away on its own in time, and you can always extend the expiry date if you still have the private key.

```
GnuPG needs to construct a user ID to identify your key.

Real name: Testy McTest
Email address: test@example.com
Comment:
You selected this USER-ID:
    "Testy McTest <test@example.com>"

Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit? o
```

After entering your identifiable information, you will be prompted for a *passphrase*. You should pick something longer than a password.

After that, you may need to use your computer for a while for GPG to collect enough random data to generate they key. Just move your mouse a lot; it will finish eventually.

The final output you will see contains your key ID:

```
pub   2048R/12C595E9 2014-12-09 [expires: 2016-12-08]
      Key fingerprint = 6105 5591 CD4B 6D5A 4142  AC0C D387 8005 12C5 95E9
uid       [ultimate] Testy McTest
sub   2048R/CE1346C0 2014-12-09 [expires: 2016-12-08]
```

In this case the key ID is `12C595E9`. We can use that key ID to publish the newly generated public key to a public key server:

```
$ gpg2 --send-key 12C595E9
```

Since this is just a demonstration, I won't really be publishing this key.

Finding and signing a key

Before you sign someone's key, you should verify their identity as best you can. Once you're confident you know who you're talking to, you can download their public key. In this case, I want to sign Harry's key:

```
$ gpg2 --search harry@thoughtbot.com
gpg: searching for "harry@thoughtbot.com" from hkp server keys.gnupg.net
(1)     Harry R. Schwartz <harry@thoughtbot.com>
        Harry R. Schwartz <hello@harryrschwartz.com>
          4096 bit RSA key 25AE721B, created: 2014-01-30
```

```
          Keys 1-1 of 1 for "harry@thoughtbot.com".  Enter number(s), N)ext, or Q)uit >
gpg: requesting key 25AE721B from hkp server keys.gnupg.net
gpg: key 25AE721B: "Harry R. Schwartz <harry@thoughtbot.com>"
```

Now that I have the key, I can sign it. Using the `--ask-cert-level` flag lets me specify the level of certainty I have in signing this key. The first thing I want to do is verify that I have the right key:

```
$ gpg2 --ask-cert-level --edit-key harry@thoughtbot.com
gpg> fpr
pub   4096R/25AE721B 2014-01-30 Harry R. Schwartz
 Primary key fingerprint: 1B41 8F2C 23DE DD9C 807E  A74F 841B 3DAE 25AE 721B
gpg>
```

At this point I should stop and check that the fingerprint I have in front of me matches the one that Harry gave me when he told me which key is his. When I'm sure they match, I can move on to signing the key.

```
gpg> sign
Really sign all user IDs? (y/N) y
```

GPG will now prompt you for your passphrase, and sign the keys. Finish the process by saving the signature, and pushing the key you just signed up to a key server:

```
gpg> save
$ gpg2 --send-key 25AE721B
```

Any questions?

Ask now, or later: **@georgebrock** on Twitter or email **george@thoughtbot.com** using

PGP key fingerprint

```
0750 F6BF 8064 E22C 68D2
3D90 0C64 3A97 B51F FCFB
```