I'm a hacker, and I love to build stuff for the Web.

TWITTER ★ LANYRD ★ GITHUB ★ G+ ★ EMAIL ★ MEAT

« ALL POSTS

OPENPGP FOR COMPLETE BEGINNERS

Thursday 20th August, 2009

This is a rather comprehensive post detailing how to set up PGP encryption onyour machine using GnuPG, an open-source implementation of the OpenPGP standard.I'll go through the core concepts of OpenPGP, generating a keypair andrevocation certificate, backing up your configuration and data, and carrying out*cryption and signing easily from both the command-line and your favouritee-mail client.

> Please note: the installation instructions are mostly geared towards Mac users, but the command-line interaction and concepts are applicable to allsystems.

A QUICK INTRODUCTION TO ENCRYPTION

There are two main types of electronic encryption: *symmetric* and *asymmetric*.In symmetric encryption, a single, shared key is used to encrypt a document fromcleartext to ciphertext, and the *same* key is used to decrypt from ciphertextback to cleartext. Obviously there is a problem using this to encrypt e-mails,since everyone has to know the shared key in order to send you an e-mail, and ifthe key is public then it offers you zero protection (since any third partymight be able to read the 'encrypted' data). Symmetric encryption is only worthanything if *only* the sender and receiver know the secret. It's also usefulwhen there is only one party involved; for example, if you are encrypting fileson your hard drive that only you will see, or if a website is encrypting users'data to put into a session cookie, etc.

process. The way this form of encryption works is that documentsencrypted with one key can *only* be decrypted using the other key (they cannoteven be decrypted with the same key that was used to encrypt them). So if I havekeys A and B, which constitute a keypair, and I encrypt a document (such as ane-mail) with key A, it can *only* be decrypted by someone with key B, and viceversa. Algorithms which implement this kind of encryption are known as trapdooralgorithms, since a single keyonly works in one direction. In addition, another property of asymmetricencryption is that if you know for certain that key B*decrypts* a piece ofciphertext, key A *must* have been used to encrypt it.

CONFIDENTIALITY

The way asymmetric encryption is used in maintaining confidentiality (i.e. theprivacy of your precious documents) goes something like this: you generate a'keypair' (a pair of keys such that documents encrypted with one can bedecrypted with the other, and vice versa). One of these is your *public* key,the other is your *private* key. So, you share your public key with the world,and keep your private key, well, private. People who want to send you an encrypted e-mail find your public key, write the e-mail, encrypt it using thepublic key and send it to you. You, on receiving the e-mail, whip out yourprivate key, decrypt the ciphertext and read it. This is how **confidentiality**is obtained with OpenPGP, and it's relatively simple.

In actual fact, it doesn't quite work like this. As it happens, and quiteunfortunately so, asymmetric encryption has a few drawbacks. It is quitecomputationally intensive, and some algorithms can produce ciphertext up totwice the size of the original text. For these reasons, a technique known as *hybrid encryption* is employed. Rather than just encrypting a document withyour public key, a unique, random *session key* is generated for that number used*once*). Thesession key document.This is also sometimes referred to as a *nonce* (needs to be completely random, essentially 'unguessable'; this iswhy having a good source of randomness (a.k.a. entropy) is very important insystem security.

The document is encrypted *symmetrically* with that session key, and then thesession key is encrypted with your public key. This will probably be muchshorter than the document itself, so the computational load of asymmetricencryption is lightened significantly. The encrypted session key is attached tothe symmetrically-encrypted document, and the session key discarded forever.Upon receiving the encrypted document, you use your private key to decrypt thesession key, and then use the session key to decrypt the entire document.

AUTHENTICITY

One of the other uses of asymmetric encryption is for verifying the authenticityof data. For example, how do I know that this e-mail was *actually* sent to mefrom Joe Bloggs and not some imposter? We can take advantage of the fact that,if one key can decrypt a piece of ciphertext, the other key must necessarilyhave encrypted it. First, the sender will take the e-mail and construct a *hash*using a *cryptographic hash function* . This function must have severalproperties:

- It should be impractical to find two distinct messages with the same hash.
- It should be impractical to obtain the original message given a hash (a CHFis a *one-way function*).
- It should be easy to compute the hash value for any given message.

The sender then takes that hash and his/her own secret key, and *encrypt* thehash with the secret key. This encrypted hash, known as the signature, isbundled along with the message when it is sent. The recipient can then take thesignature, the original message, and the sender's public key, and attempt todecrypt the signature using the public key. The recipient will hash the messagethemselves, and compare the decrypted signature to the hash of the message. Ifequal, the recipient can know for certain that the message was created by someone with a certain private key (without knowing the key themselves). This ishow **authenticity** is obtained using OpenPGP.

It also helps to enforce data **integrity**, since the hash check will fail ifthe document has been changed in transit, so either data corruption ormalicious intent will be detected.

This is usually used in tandem with encryption, providing confidentiality andintegrity, but it's also pretty useful for things like software releases, wherethe maintainer will sign the release file with his/her private key. You'lldownload the release file and the corresponding `.sig` or `.asc` file, andverify that the package was downloaded intact and that it was published by thebona fide maintainer.

KEYSERVERS

Obviously, if you have your public/private keypair, you'll want to somehowpublish the public key somewhere where everyone can access it. Keyservers are arelatively simple way to do so. A keyserver is literally just a repository ofpublic keys; I'll show you later how to

push to and fetch from a keyserver,since this is an important way of keeping your keyring (i.e. the public keys ofothers you have saved locally) up-to-date, and also keeping others' keyringsup-to-date (such as when you make a change to your key).

WEB OF TRUST

One of the most important concepts in OpenPGP is that of a Web of Trust. Let'ssay Alice decides to generate a keypair using your identity (name and e-mailaddress). How are people to decide which belongs to who? Not only might someonesend you e-mail encrypted for Alice, rendering you unable to read it, but Alicecould also read that e-mail, constituting a major breach of security (not tomention a *serious* inconvenience). Alice could also sign documents, trickingrecipients into thinking that you created them.

Fortunately, using the concept of signing as detailed above, you can sign apublic key, as if to say 'I trust this key—owner relationship'. The more keyssign that key, the more that key becomes 'trusted'. Eventually, you will havewhat is known as a 'Web of Trust'; where key A is signed by key B and C, key Bis signed by keys A and D, and key C is signed by keys A, B and D, for example.The set of keys and signatures makes up the Web of Trust. You can determinewhether an identity is to be trusted or not by following the chain of trusted friends. This way, anyone trying to pose as a user will not be able to get muchtrust; you'll always go with the key that works out as most trusted.

I'll give an example. Let's say you and Bob are best friends. You trust himcompletely, so you've signed his key. Bob has a friend called Chris, who *he*trusts completely, so he has signed his key. If Chris sends you a signed e-mail,you're very likely to believe that signature indeed belongs to Chris, becausethe authenticity of the key with which it is signed has itself been attested toby your best friend Bob (and you can verify*that* by checking Bob's signature against his public key). It can get confusing for even relatively smallnetworks, but fortunately a simple computer algorithm can calculate a 'trustlevel' for a given signature, based on following the sequence of connections oftrust linking you to that signature.

:change their key fingerprints (since each

You might also be happy to know that OpenPGP is a good reason to have a party.At key-

OpenPGP key has a unique identifier); the morning after, everyone goes onlineand signs the keys they accepted the night before.

SUBKEYS

Subkeys are quite a subtle but powerful concept. The problem with the keypair isthis: what happens when a key is compromised? You'll have to revoke your oldkeypair (a process which involves you verifying that you are the owner of thekey, after which it will be marked as revoked), thus losing your entire Web ofTrust, and then re-generate a key and publish that. Clearly not a very goodsituation to be in.

OpenPGP uses a concept known as a subkey to mitigate this problem. Rather thanhave a single keypair, you'll have a 'master' keypair which acts as youridentity, and a 'subkey' pair which does the actual *cryption. If for somereason a subkey pair becomes compromised (i.e. someone finds out yourpassphrase), you can revoke just that subkey and create a new one, withoutlosing your identity or WoT.

GETTING STARTED WITH OPENPGP

This'll just be a brief guide to getting GnuPG2 installed and running on yourMac. I haven't included full instructions for GNU/Linux, *BSD, other good OSesor Windows here because I'm not running any of those platforms right now; youcan probably find some good guides to installing GnuPG2 elsewhere. Since a lotof this involves command-line work, which is likely to be the same on mostplatforms, it may be applicable to you anyway.

INSTALL GNUPG2

This is pretty easy to do on Mac; download the ZIP file from .mpkgfile here, unzip, install the and you're done.

Alternatively, if you use MacPorts, you can just run `sudo port install gnupg2`

Systems with `apt`, `rpm`, `emerge`, et cetera can also use their respectivepackage managers to obtain a copy of GnuPG2.

GENERATE A KEYPAIR

This will involve using the command line. Launch Terminal.app (or your preferredterminal emulator) and do this:

```
gpg2 --gen-key
```

You'll see:

```
gpg (GnuPG/MacGPG2) 2.0.12; Copyright (C) 2009 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?
```

Hit enter, since the defaults tend to work fine. Then:

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

I personally go with 4096 at this point, but you do need to make a decisionhere. Whilst longer keys are more secure, they can also take a long time togenerate (*especially* on older hardware; 4096-bit keys can take up to an hourto be generated on a PowerPC-based Mac), and *cryption will also be slower.This might not matter if you and all your contacts are running relatively recentand powerful hardware, but once OpenPGP support for embedded and mobile devicesbecomes more available, it will present a serious issue. For this reason you maywant to go for the default. After deciding, you'll be presented with thisscreen:

```
Requested keysize is 4096 bits
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
Key is valid for? (0)
```

Hit enter again. This should be fine: if somehow your key becomes compromisedyou'll be able to revoke it using the revocation certificate (more on thatlater). It'll also ask you for confirmation; just hit `y` then enter again. Thegenerator will then ask you to answer a series of questions:

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

Real name: <Type your full name, hit enter>
Email address: <Type your email address, hit enter>
Comment: <Optional: type a comment or your homepage URL, hit enter>
You selected this USER-ID:
```

```
    "Real Name (http://example.com) <name@example.com>"
```

After that it will ask you to confirm:

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

Just type `O` (that's the capital letter, not the numeral) and then enter. Atthis point a dialog box will appear asking for a passphrase. This is necessaryto make sure that even if someone does gain access to your secret key, theymight not be able to use it. The normal rules for strong passwords apply. Tryand make it at least 15 characters long, memorable but not obvious to anyone whoknows you, include a mixture of upper- and lower-case letters, numbers andpunctuation. You will, of course, be asked to repeat the password.

At this point, you'll see something like this:

```
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
```

Do what it says. It's generating your keys, but needs a lot of random numbers todo so. I recommend playing around on the keyboard, fiddling with the mice, orhaving a game of Quake Live. If you chose a key size of4096 bits earlier, this might take a while, but don't worry because it'll onlyhappen once per keypair that you generate. After this is done, you should seesomething along the lines of:

```
gpg: checking the trustdb
gpg: 3 marginal(s) needed, 1 complete(s) needed, PGP trust model
gpg: depth: 0  valid:   1  signed:   0  trust: 0-, 0q, 0n, 0m, 0f, 1u
pub   4096R/<8-CHAR KEY ID> 2009-07-17
      Key fingerprint = <36-CHARACTER KEY FINGERPRINT>
uid                  Real Name (http://example.com) <name@example.com>
sub   4096R/<8-CHAR KEY ID> 2009-07-17
```

The 8-character Key ID on the line beginning with `pub` is the ID of your publicmaster key (the one used for identity and signing), the one on the linebeginning with `sub` is that of your public subkey (the one used for*cryption), and your primary user ID is on the line beginning with `uid`. Notethat you can actually add user IDs to this key later. This is useful if you wantto do *cryption on multiple e-mail addresses (like if you have a personal and awork e-mail address). Take a lookat the typing at the console) for more information.

SENDING YOUR KEY TO A KEYSERVER

To publish your public key to a keyserver, run:

```
gpg2 --send-keys PUB_KEY_ID
```

Where `PUB_KEY_ID` is the public master key ID (as shown towards the end of theprevious step). GnuPG2 will default to using the `keys.gnupg.net` publickeyserver; most public keyservers synchronize with one another, so you cansafely leave this as the default.

CREATING A REVOCATION CERTIFICATE

If your key becomes compromised or useless somehow, you'll need to be able torevoke it (i.e. declare that it has been compromised or is no longer used). Thisis simple enough; just run:

```
gpg2 --gen-revoke PUB_KEY_ID > revocert.asc
```

Where `PUB_KEY_ID` is the public master key ID. The right angle-bracketcharacter (`>`) will cause the output of the command (i.e. the certificateitself) to be saved into the file `revocert.asc`. You might want to change thisto include the key ID itself (i.e. `rev-KEY_ID.asc`).

To revoke your key, run:

```
gpg2 --import REVO_CERT_FILENAME
```

This will mark your key as revoked *locally*. In order to let the keyserverknow, just send the key ID to it using:

```
gpg2 --send-keys PUB_KEY_ID
```

Note that this is exactly the same command as when you sent it your key thefirst time.

A lot of people like to print out (as in *dead tree*) an ASCII-armored versionof their revocation certificate and keep it safe in storage. Should they everneed to use it, they can just scan the printout and use OCR to retrieve thecertificate.

SIGNING KEYS

Signing keys is crucial to creating a Web of Trust. Thankfully it's relativelyeasy to sign someone else's key. First you have to grab the user's key. If it istored on a keyserver, run:

```
gpg2 --recv-keys KEY_ID
```

If it's in a text file on your local machine, you can just do `gpg2 --import FILENAME`.To sign the key, run:

```
gpg2 --sign-key KEY_ID
```

It'll ask you to confirm the signature operation, and then it'll ask you toinput your passphrase. Once this is done, you'll have added your signature tothe key. Since the key is now modified, you'll want to send it back to thekeyserver. To do this, run:

```
gpg2 --send-keys KEY_ID
```

This is the same command used when you send your own key.

*CRYPTING DOCUMENTS

After all that setup, it's actually quite simple to encrypt and decryptdocuments. If you have a file called `document.txt` on your hard drive thatyou'd like to encrypt, just run:

```
gpg2 --encrypt document.txt
```

You'll be prompted to enter the user IDs to whom you're sending the encrypteddocument:

```
Enter the user ID.  End with an empty line: <Type User ID here>
```

If you have only one key in your keyring for a person called "Joe Bloggs", youcan just type

that and it'll get the correct key for you. At each stage thecommand will print all the recipients you've specified so far. To finish, justhit enter on a prompt without typing anything. Alternatively, you can specifyall of the User IDs on the command line, like so:

```
gpg2 --encrypt document.txt -r "Recipient 1" -r "Recipient 2"
```

Where `-r` is short for `--recipient`. You can pass `-r` as many times as youlike in one command.

GPG2 will encrypt the document and save the encrypted ciphertext as `document.txt.gpg` (mutatis mutandis). This file will, however, be in a binaryformat. If you need a plaintext version, just pass the `--armor` flag to the`gpg2` invocation. This produces an *ASCII-armored* version of the ciphertextcalled `document.txt.asc`(m.m.), suitable for inclusion in e-mails, web pageset cetera.

To decrypt a document that someone has sent you, you can just run:

```
gpg2 --decrypt document.txt.gpg
```

This will prompt you for your private key's passphrase, and save the decrypteddocument in `document.txt`(m.m.). It doesn't matter if the document is in anASCII-armored or binary format, since GPG2 will recognize it either way.

SIGNING AND VERIFYING DOCUMENTS

Signing a document to prove its authenticity is also simple with GPG2. You don'tneed to specify any recipients, but you will need to provide your private key'spassphrase. To sign a document, just run:

```
gpg2 --sign document.txt
```

You can also pass in `--armor`, if you need plaintext output. This will create afile called `document.txt.gpg`or `document.txt.asc`depending on whether ornot you armor the output. To verify a signature, just run:

```
gpg2 --verify document.txt.gpg
```

For this to work, `document.txt`has to be in the same directory as itssignature, and you also must have imported the public key of the person whosigned it (since the public key is used in verification).

MAKING BACKUPS

At this point you'll want to make regular back-ups of your keyring andassociated files. This is easy enough; simply back-up the contents of the `.gnupg`directory in your home folder, and also create a single back-up of therevocation certificate you generated in the last step. You'll actually want toback-up all the files *not* ending in either `.gpg-agent`or the tilde character(`~`). The first denotes a *file lock* which a process on the system called `gpg-agent` uses, and files ending in a tilde are back-up files (which obviouslyyou won't need if you are actually *making* a back-up). You probably have yourown back-up strategy, but I will tell you how best to make backups so that thirdparties can't easily access your backed-up data.

I use (and recommend) a piece of software called TrueCrypt to secure all of my backups. TrueCrypt allowsyou to create an encrypted drive in a file, which can then be *mounted* by theapplication, read from/written to, and then unmounted again. To most eyes itwill look like it is made of completely random data. You can also encrypt wholepartitions, allowing you to create an encrypted USB stick or external harddrive.

I'd recommend that you use TrueCrypt to create the drive on which you carry outall your back-ups. I won't go into the details of setting up an encrypted drive,since I was able to figure out what to do on my first try (it is a GUIapplication after all).

In addition, another nifty trick you can use is to *move* the external drive, and just make it into a symbolic link like so:

```
ln -s /Volumes/EncDrive/gnupg ~/.gnupg
```

When the drive is mounted, GnuPG2 will work fine; when ejected, trying to useGnuPG2 will just fail with an error (which is what you want it to do anyway).

USING OPENPGP WITH AN E-MAIL CLIENT

OpenPGP's main area of application is in securing e-mails. Most important andconfidential information is still sent via e-mail, so you need to be able toquickly *crypt your e-mails and verify that they are from trusted sources. Inthis respect, e-mail client integration is key. Fortunately, most mail clientshave mature and readily-available OpenPGP plugins.

If you want to be able to encrypt, decrypt, sign and verify e-mails from withinApple Mail 3.0 (the Leopard version) in a very simple way, you can use the freeand fantastic GPGMail software. Theinstallation instructions on its homepage work fine; just walk through the installer and restart Mail, and you should have all the good PGP functionalityright there in Mail.

Thunderbird users can install the Enigmail plugin to get PGP goodnessfrom within

can't offer much guidance, but I hear that it'spretty much the most complete solution available (and it works on all theplatforms that Thunderbird does).

If you use GMail from Firefox, you can install FireGPG, which will let you do all the useful OpenPGP operations from within your browser.

`.gnupg`directoryto the encrypted