GNU Privacy Guard Howto

A short presentation on encryption in general and GNU Privacy Guard

What We Will Cover

- Some history of encryption
- Some important definitions you will need to use GNU Privacy Guard and encryption in general such as:
 - What a key is
 - Symmetric/Asymmetric keys
- Authentication and Digital Signatures.
- I will include a bibliography.
- If we have time, we can interactively use gpg.
- I'll discuss anything at the after-meeting meeting at The Cambridge Brewery.

What We Will Not Cover

- Details of the specific encryption methods, such as the mathematics behind encryption
- System security.
- How to configure an email client. Each email client is different. Most Linux email clients support GPG as do most Windows clients.

A brief history of encryption

- As long as there has been a written language there have been attempts to encrypt (or scramble) the text so that only a select few could read the data. One example is Nostradamus' Quatrains.
- Years ago while working in a retail store, we used a word to encrypt prices on some merchandise. The sales people could easily decrypt the numbers.

A brief history of encryption

- During wartime, important messages were manually encrypted using code books.
- In World War II, encryption/decryption became automated by using machines, such as the German Enigma. More recently computers can (obviously) produce complex encryption algorithms that are very difficult to crack.

Cracking

- I maintain that given time a resources, there is no encryption method that cannot be cracked.
- What we need are encryption methods that are relatively easy for everyone to use to encrypt messages.

Definitions

- Cipher Text. This is the text that has been encrypted.
- Plain Text. The text to be encrypted.
- Key. This is what the encryption program uses to convert the plain text block to the cipher text.
 Keys may be *symmetric* or *asymmetric*.
 (Discussed later).
- key ring. This is your repository for public keys.

Definitions

- Session Key. This is a key that is created for the duration of a session.
- Authentication. The process of validation that indicates who you are actually communicating with.
- Key server. A server on the Internet that stores public keys.

Key Signing Parties

- I mention key signing parties in passing elsewhere, but I think we need a bit more explanation.
- Before we can use a public key reliably, we need to know who belongs to the public key. The best way is a face-to-face meeting.

Key Signing Parties

- Various groups, including the BLU, have periodic key signing parties. What we normally do is for everyone who wants his/her key signed register their keys with the BLU ahead of time.
- Then at the party, all the participants show their photo ids to eachother.
- After the party, we download eachothers keys to our keyring. Sign the keys, then send the keys back to the server.

Symmetric Vs. Asymmetric

- A symmetric key is one where the same key is used to both encrypt and decrypt the data. Methods such as DES (and 3DES), Blowfish and others use this. Symmetric keys are shorter than asymmetric keys resulting in better performance.
- An asymmetric key is a pair of keys. One used to encrypt the data and one used to decrypt the data. Normally this consists of a public and a private(secret) key.

What problems do we want to solve.

- 1. The privacy of the data in the message.
- 2. The guarantee that only our intended recipient can view the message.
- 3. That the message I receive comes from the person I think it does.

The solutions in a capsule

- 1. We encrypt the message using a good encryption method so that only the recipient can view the message.
- 2. We make sure that we use the encryption key that we know belongs to the recipient.
- 3. We add a digital signature so that our recipient knows the message comes from us.

Public Key method

- We use a public key encryption method so that we can give our key to our correspondents. The public key is one part of the asymmetric key pair. If I give each of you my public key, then I can send you a message with a digital signature, and you know the message must be from me if you can validate the signature with the public key.
- Public key encryption was invented in 1976 by Whitfield Diffie and Martin Hellman in 1976.

Encrypting with asymmetric keys.

• To encrypt a message, you use the recipients public key so that the message can only be decrypted by the recipients private key.

Digitally signing with asymmetric keys.

- A message that is digitally signed accomplishes 2 things:
 - 1. Authenticates the sender.
 - 2. Guarantees that the message has not been altered.
- The sender uses his/her private key to create the digital signature such that only the sender's public key can be used to decrypt the signature.
- The message contains a hash contained in the signature that is compared when decrypted.

Digital Signature Example (blank lines removed)

From: Jerry Feldman <gaf@blu.org>

To: gaf@gaf.blu.org

Subject: This is an example of an inline Digital Signature

Date: Wed, 16 Jul 2003 16:32:42 -0400

Organization: Boston Linux and Unix

X-Mailer: Sylpheed version 0.9.0claws (GTK+ 1.2.10; i686-pc-linux-gnu)

-----BEGIN PGP SIGNED MESSAGE-----

Hash: SHA1

This is a short example of an inline digital signature. You can also use

MIME so that the signature is sent as an attachment.

Digital Signature Example (blank lines removed)

Jerry Feldman <gaf@blu.org>

Boston Linux and Unix user group

http://www.blu.org PGP key id:C5061EA9

PGP Key fingerprint:053C 73EC 3AC1 5C44 3E14 9245 FB00 3ED5 C506 1EA9

-----BEGIN PGP SIGNATURE-----

Version: GnuPG v1.2.1 (GNU/Linux)

iD8DBQE/FbZq+wA+1cUGHqkRAsq6AKCDlua/xubDH71OgaZ5EPyrSb Bi/wCfa9G5

ensE/iuyBIubhvMa4TVYZhc=

=L1qc

-----END PGP SIGNATURE-----

Secure Shell (nonsequitur)

Just an aside before I get into GPG. The Secure Shell uses a combination of public/private key to establish authentication, and symmetric key methods to encrypt the actual messages. There are two protocols (ssh1 and ssh2). SSH2 is the more secure of the two.

GNU Privacy Guard

 We can use GNU Privacy Guard to both encrypt and sign messages. GNU Privacy Guard is an implementation of Open PGP (RFC 2440). PGP (Pretty Good Privacy) was invented by Phil Zimmerman in 1990. Essentially, Phil made his program share ware and was sued by the federal government in 1993 and dropped in 1996.

What Does GPG buy you

- Essentially you can use GPG to
 - 1. Digitally sign messages.
 - 2. Validate digitally signed messages
 - 3. encrypt and decrypt messages
 - 4. manage your keyring
 - 5. export and import keys
 - 6. create new keys.

Installing GNU Privacy Guard

 I recommend that you download and install GNU Privacy Guard from the GPG site or mirrors :http://www.gnupg.org The reason is that the versions included with your distribution may be out of date. I normally download the tarball and build from there.

- The command to generate your key is:
 - gpg --gen-key

gpg (GnuPG) 1.2.1; Copyright (C) 2002 Free Software Foundation, Inc.

This program comes with ABSOLUTELY NO WARRANTY.

This is free software, and you are welcome to redistribute it under certain conditions. See the file COPYING for details. Please select what kind of key you want:

(1) DSA and ElGamal (default)

(2) DSA (sign only)

(5) RSA (sign only)

Your selection?

DSA keypair will have 1024 bits. About to generate a new ELG-E keypair. minimum keysize is 768 bits default keysize is 1024 bits highest suggested keysize is 2048 bits What keysize do you want? (1024) Requested keysize is 1024 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) 10

Key expires at Sat 26 Jul 2003 01:47:02 PM EDT

Is this correct (y/n)? y

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

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

Real name: Boston Linux

Email address: bogus@blu.org

Comment:

You selected this USER-ID:

"Boston Linux <bogus@blu.org>"

Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit? o You need a Passphrase to protect your secret key.

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. public and secret key created and signed. key marked as ultimately trusted.

pub 1024D/B2DDAD69 2003-07-16 Boston Linux <bogus@blu.org>
 Key fingerprint = F626 CDBB 98DE 9722 A0FE 0857 2555 C890 B2DD AD69
sub 1024g/E62F59DB 2003-07-16 [expires: 2003-07-26]

- Some points about the previous slide. We can generate keys of different sizes. The longer the key, the more secure.
- I expired my key after 10 days. Normally you set some reasonable expiration so if you forget your pass phrase, the key will be revoked automatically.
- You pick a pass phrase you use to work with your secret key.

- Your key id is the underlined text.
 pub 1024D/<u>B2DDAD69</u> 2003-07-16 Boston Linux <bogus@blu.org> Key fingerprint = F626 CDBB 98DE 9722 A0FE 0857 2555 C890 B2DD AD69 sub 1024g/E62F59DB 2003-07-16 [expires: 2003-07-26]
- You can also identify the key by its fingerprint.

Sending a key toa Key server

 After you have generated your key, you probably want to send it to a key server.
 I use keyserver.kjsl.comBut there are many others

all networked together, such as pgp.dtype.org. You can place your key server in ~/.gnupg/optionsunder keyserver.

 The command to send your key to the key server is: gpg --send-keys keyid [...] You can use the --keyserver option to specify a keyserver.

Receiving Keys from a key server

- You can receive a key from a key server by using the command: gpg --recv-keys key-id [...] If you do not have a default key server, you can use the --keyserver option.
- This adds those keys to your keyring.

Signing keys

- Once you send your key to a key server, how does your recipient know that it is your key and vice versa.
 - We sign eachother's keys. If I can prove to you who I am, I can give you my key id. You then load it into your keyring, and sign that key, then send it to the key server.
 - We have periodic key signing parties where we exchange key ids and picture id cards.

Signing Keys

 You use the --sign-keys option to sign keys in your keyring. gpg --sign-key keyid.

Other subcommands

- There are many subcommands of gpg that can be used to list, encrypt, decrypt, revoke, add more user ids. The man page is pretty complete, but we are running out of time.
- You can export and import keys in ASCII armor format, that you can use to send your public key to others.

Bibliography

- http://www.gnupg.org/ GNU PG page
- http://www.state.ma.us/itd/legal/e-sig.htmDan Greenwood's page on Digital Signatures
- http://www.faqs.org/rfcs/rfc2440.html Open PGP
- http://www.faqs.org/rfcs/rfc1991.html PGP messaging (Our own Derek Atkins participated in this one).
- http://www.faqs.org/rfcs/rfc3156.html PGP MIME Security
- http://www.faqs.org/rfcs/rfc1750.html Randomness
- http://world.std.com/~franl/crypto.html Fran Litero's Cryptography page.
- http://www.pgp.com PGP Corporation
- http://www.RSA.com RSA Corporation