

Introduction to GnuPG

Michael Meffie
Canton Linux Enthusiasts

October 2001

Overview

- Need for Privacy and Verification
- Conventional and Public Key Encryption
- Digital Signatures and the Web of Trust
- PGP and GnuPG
- Using GnuPG to Encrypt and Sign Documents
- GnuPG Front-Ends

Your Right to Privacy

The right of the people to be secure in their persons, houses, papers, and effects, against unreasonable searches and seizures, shall not be violated, and no Warrants shall issue, but upon probable cause, supported by Oath or affirmation, and particularly describing the place to be searched, and the persons or things to be seized.

US Constitution, 4th Amendment

Privacy in the Digital Age

- Paper based mail is sealed by envelopes.
- Email is transmitted in the *clear*.

Kinds of legitimate, private info you may need to send:

- credit card numbers, social security numbers

- price lists, customer lists
- passwords
- proprietary source code

Need for Verification

How do you know if a message was forged?

How do you know if someone tampered with a message (or program)?

Conventional Crypto

- Message to be encrypted is called the plain text (or clear text)
- After encryption it is called cipher text
- Encryption and decryption using a well known algorithm
- Encryption AND decryption with the same key value
- The algo is public but the key is secret

Conventional Crypto Algos

- DES - Data Encryption Standard (56bits), obsolete
- Triple DES - extension of DES
- AES (Rijndael) - new NIST standard

- IDEA - used by PGP, patented
- Blowfish - used in ssh, gpg
- Enigma - used by the Germans military in world war II

Conventional Crypto Problem

The main disadvantage of conventional crypto is the sharing of the secret key.

- How do we communicate the initial key in a secure way?
- How do we scale? We'd need a key for EACH pair of humans.

Public Key Crypto

What if we could encrypt with one key and decrypt with another? This is called "asymmetric cryptography".

- Public Key to encrypt the message
- Secret Key to decrypt the message

The public key can be distributed IN THE CLEAR. The secret key cannot be disclosed.

Generating Keys

The public and secret key pairs are generated by a key generation algo.

- The keys are related, but you cannot derive the secret key from the public key
- Both keys are created at the same time.
- Requires lots of high quality random numbers.

Public Key Encryption

Suppose Bob needs to send a private message to Alice.

- Bob obtains Alice's public key (from a key server for example)
- Bob encrypts the message with the public key and sends the ciphertext to Alice
- Alice uses her secret key to decipher the message

Only Alice can decipher the message, because she has the secret key.

Hybrid Encryption

- Practical systems use a mix of public and conventional encryption.
- Public key encryption is used to exchange a temporary session key
- The session key is a conventional encryption key (such as IDEA)
- This is done because public key encryption of large messages is too slow

Digital Signatures

Digital signatures can be used to verify electronic documents. Let's say Alice wants to send a signed response back to Bob.

- Alice computes a digital signature by encrypting the response with her secret key.
- Alice sends the response and the signature to Bob.
- Bob decrypts the signature with Alice's public key
- Bob compares the decrypted signature to the response received.

If the decrypted signature does not match the response, then Bob concludes the message was corrupted or it was not from Alice.

Signed and Sealed

In the previous example, Alice's response was sent in the clear. Signed messages can also be encrypted if Bob has a public key pair also.

1. Alice signs the message with her secret key and appends the signature to the message.
2. Alice then encrypts the signed message with Bob's public key.

Key Forgery

Public key forgery is a main attack of public key encryption. For example, imagine a Chuck wants to read Alice's mail.

- Chuck generates a public/secret key pair.
- Chuck impersonates Alice, sending the forged key to Bob via email.
- Bob encrypts a message with the forged key and Chuck intercepts the message.
- Chuck decrypts the message with his secret key and reads the message.
- Finally, Chuck encrypts the message with Alice's real public key and sends it to her.

Alice nor Bob are not aware of the deception.

Key Fingerprints

- Public keys are very long, which makes it hard to check by hand.
- For all practical purposes each key has a unique fingerprint.
- A fingerprint is a hash of the key.
- You can give you key fingerprint to the people that need your public key.

Key Certificates

- It is not possible to directly communicate your key fingerprint to everybody.
- So we sign keys for distribution.
- Certificates contain the key plus one or more signatures.
- The signatures can chain, as long as each signature is valid, the key is valid.
- There are two models: Centralized and Distributed
- PGP and GnuPG use the distributed model, called the web of trust

Web of Trust

- Who do you trust to properly sign public keys?
- You choose
- You are responsible to sign public keys that you have verified.

PGP

- Pretty Good Privacy
- written by Phil Zimmerman
- developed for ordinary people
- freeware and commercial versions
- as of last week, NAI is in the process of dumping the product line.

GnuPG

- A free version of PGP
- no patent encumbrances
- GPL

- current version is 1.0.6
- mostly compatible with PGP, but there are some issues
- gpg command line tool
- gpa front end
- gpg-me library for email clients

```
gpg --clearsign myfile.txt
gpg --detach-sign myfile

gpg --verify signedfile
```

Installing GnuPG

- <http://www.gnupg.org>
- user manual is available on the web site
- autoconfig style make

```
tar xzf gnupg-1.0.6.tar.gz
cd gnupg-1.0.6
./configure
make
make install
```

Encrypting with gpg

```
gpg --recipient Slats \
  --encrypt myfile.bin

gpg --armor \
  --recipient Slats \
  --encrypt myfile.txt

gpg --armor --clearsign \
  --recipient Slats\
  --encrypt myfile.txt

gpg --decrypt encryptedfile

# conventional encrypt
gpg --symmetric privatefile
```

Using gpg

```
# create a public/private key pair
gpg --gen-key

# import public or private keys
gpg --import

# show the keys on the keyrings
gpg --list-keys

# show key fingerprints
gpg --fingerprint

# modify key trust
gpg --edit-key
```

Using Front Ends

- GPA is a gui client for gpg.
- Not production ready yet, version 0.4 is available

Using Email Clients

- gpgme is a library for email clients that support gpg
- Not production ready yet, verion 0.2 is available

Signing with gpg

```
Example:
gpg --sign myfile.bin
```