

Using GNU Privacy Guard

10 April 2003

David Van Horn <dvanhorn@cs.uvm.edu>

29EF BA1C A8A8 FA48 FE4A 661A 1F6B 6310 C604 D418

Copyright © 2003, David Van Horn. Licensed under the Academic Free

License version 1.2

Foreword

<http://www.cs.uvm.edu/~dvanhorn/>

These slides are available in PDF (gnupg.pdf) format. The file gnupg.sig contains the signature for the \LaTeX source file, gnupg.tex. My public key is contained in the file dvanhorn.asc.

I would like to dedicate these slides to my mother: a self-described conservative, bean-counting, capitalist.

Motivations

Identity We want to have identities. We want to identify people. We name them. Want to be able to address a person.

Authenticity Want to be sure something is authentic. It is not a forgery. The person claiming authorship is in fact the author. Information has not been altered.

Privacy Want to be able to protect information from those not intend to see it. We have the right to withhold our associations and communications from **anyone** as we see fit. Want to grant access to named, trusted, authentic entities.

These are old concepts.

Example: Official Transcript ...

What is Public Key Encryption?

Public key encryption is an approach to realizing the concepts of **identity**, **authenticity**, and **privacy** in a digital context.

What is OpenPGP?

RFC 2440 specifies the OpenPGP Message Format*. This is the formal specification of the PGP public encryption scheme. Much of this tutorial is derived from that document. As an implementor of OpenPGP software, you should adhere the RFC 2440 standard.

*Other examples include 2822 **Internet Message Format** and 2045 **Multi-purpose Internet Mail Extensions**.

What is GNU Privacy Guard?

GnuPG is a RFC2440 compliant application. GnuPG is Free Software. It can be freely used, modified and distributed under the terms of the GNU General Public License. Because it does not use the patented IDEA algorithm, it can be used without any restrictions.

There are other OpenPGP compliant applications, but GnuPG is all that we will consider today.

A considerable amount of today's material is from the GNU Privacy Handbook.

Why give a shit?

Privacy is considered a **right**. But rights may and will be violated. Public key encryption provides a **means** to exercise your right. It protects privacy. Furthermore, it is the freedom of speech which ensures free public, uncensorable **access** to that means. Because the GNU project has taken the stance that code is speech, it is afforded the same protection as such.

GNU Privacy Guard is a tangible public asset created by the Free Software movement.

I thought this was a tutorial!?

Public Key Encryption Concepts

Public key The public side of your digital duality. Used to identify you. Your mark. Always reveal. Means of identity.

Private key The private side of your digital duality. Used to sign and encrypt. Your mark making “stamp”. Never reveal. Means of identity.

Fingerprint Used to verify public keys. Means of identification.

Passphrase Decrypts your private key. Protection of your identity.

Revocation certificate Invalidates your public key. For use if private key is lost or compromised. Don't reveal unless you truly desire to revoke key. Anyone can publish revocation certificate.

Key servers Public key databases. Facilitate the distribution of public keys.

Encryption Method of protecting digital data from being viewed by unintended parties. Means of privacy.

Signature Method of authorship and ensuring integrity (has not been changed). Means of authenticity.

Today's Tasks

Today we go through the process of creating a new key pair. We will encrypt, decrypt, sign and verify documents with that key pair. We will import a key from a key server and show how to export to a key server as well.

Unfortunately, it is highly encouraged that you **not** use today's key pair as your permanent key pair.

Why?

- You should not store your private key on a University machine. Backups are made daily. It's an institution of the state.
- The installation of GnuPG is faulty. Memory may be flushed to disk while running the program. (This is fixable, and I will notify help).
- You should only use a installation from **verified** tar balls.
- *Again, You should not store your private key on a University machine. Backups are made daily. It's an institution of the state.*

What to do

- Download, verify (with MD5 checksum), and install on your own computer.
- Use GnuPG on EMBA **only** to verify and encrypt messages. Don't sign or decrypt (will need private key).
- Store your encrypted private key and revocation certificate using read-only medium (eg. cd-rom, hard-copy) in safe place (eg. safe deposit box).
- Don't ever copy your private key (even if encrypted) onto a computer you do not trust.

- Do not trust EMBA machines.
- Choose a good pass-phrase.
- Publicize your public key.

Creating a key pair

```
% gpg --gen-key
```

```
Please select what kind of key you want:
```

```
(1) DSA and ElGamal (default)
```

```
(2) DSA (sign only)
```

```
(4) ElGamal (sign and encrypt)
```

```
Your selection? 1
```

```
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) 2048
```

The expiration date specifies when the key will be outdated, if ever. Most of the time, you'll want a key pair that does not expire. The expiration may be changed, although it is difficult to do because you must communicate the new expiration date to all who have your public key.

The User-ID identifies the **real** you.

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) 0

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>"

Choosing a passphrase

A good passphrase is crucial to the secure use of GnuPG.

Encryption is a game of weakest link. Your passphrase can certainly be the weakest link. It is the last defense against access to your private key. Your private key is stored in encrypted format on your machine. The passphrase decrypts it.

Take time when you choose your real passphrase. There's no limit on the length. Any characters can be used, including whitespace. Don't use dictionary words. Mix case. Mix alphabetic and non-alphabetic characters. Should you save it somewhere, do not save it on the same machine or disk as your private key.

Data files

Your key pair is created and signed (more on this later). A new directory **.gnupg** is created. The following files are created.

pubring.gpg Contains the your public key and the public keys that you import.

secring.gpg Contains your (encrypted) private key and any other private keys you import or create, although you typically have one private key.

trustdb.gpg Contains signatures of keys.

Listing your keys

You can list the keys stored by GnuPG.

```
% gpg --list-keys
```

```
/home/cs/csugrads/dvanhorn/.gnupg/pubring.gpg
```

```
-----  
pub 1024D/C604D418 2002-03-02 David Van Horn <dvanhorn@cs.uvm.edu>  
sub 4096g/EA9979E2 2002-03-02
```

Exporting your key

We would now like to distribute our public key. To export a key from GnuPG we do the following. You can email your key to friends, post it on your web page or send it to a key server (more on this later).

The **armor** option causes output to be ASCII plain-text as opposed to the default binary representation. The **output** option names the file to write to. If omitted the standard out would be used. The **dvanhorn** argument is the *key-specifier*, either the key ID or any part of a user ID that sufficiently identifies the key.

```
% gpg --output dvanhorn.asc --export --armor dvanhorn
```

Example: Public Key

This is an abbreviation of my public key. Notice this was run on an EMBA machine.

```
-----BEGIN PGP PUBLIC KEY BLOCK-----  
Version: GnuPG v1.0.0 (SunOS)  
Comment: For info see http://www.gnupg.org  
  
mQGIBDyBQZURBADiu0FmvV0t85mVlWSf2HvvHnCOFXLhpDMgpdich7e3mP8oYES  
NJthWpntwfW1xWER7M+1rGPx2jsaXggg+2QvLitvxgsFVORogBb8I2Z6HzYpwZW0  
:  
CRARhPoPq6kWV4B3AJ9AJxMEX9GisHHhr7FJKWrBZWeIQcfZqR3rT+7Im5feWuJ  
3P+GHNhy8iY=  
=gnWe  
-----END PGP PUBLIC KEY BLOCK-----
```

What if...?

In the case that your private key is compromised or you should happen to forget your password, there needs to be a mechanism for “disowning” a key. This is accomplished via **revocation certificates**. You should generate the certificate when you create the key. Keep in a safe place. Anyone can publish this.

```
% gpg --output revoke.asc --armor --gen-revoke dvanhorn
```

Importing other public keys

There are several ways of importing other keys.

Import a file containing the key.

```
% wget http://www.cs.uvm.edu/~dvanhorn/dvanhorn.asc  
% gpg --import dvanhorn.asc
```

Copy and paste to the standard in. Ctl+D for EOF.

```
% gpg --import
```

Import from a keyserver. The argument to **recv-key** is a key ID, this can be found by searching the keyserver.

```
% gpg --keyserver pgp.mit.edu --recv-key C604D418
```

Verifying imported keys

Fingerprints are used to verify keys. Communicate directly with key holder to exchange fingerprints.

```
% gpg --edit-key dvanhorn
```

```
pub 1024D/C604D418  created: 2002-03-02 expires: never      trust: -/q
sub 4096g/EA9979E2  created: 2002-03-02 expires: never
(1) David Van Horn <dvanhorn@cs.uvm.edu>
```

```
Command> fpr
```

```
pub 1024D/C604D418 2002-03-02 David Van Horn <dvanhorn@cs.uvm.edu>
      Fingerprint: 29EF BA1C A8A8 FA48 FE4A 661A 1F6B 6310 C604 D418
```

```
Command> sign
```

```
:
```

```
Command> check
```

```
uid David Van Horn <dvanhorn@cs.uvm.edu>
sig!      C604D418 2002-03-02  [self-signature]
sig!      135B049E 2003-04-10  Your name <your@email.org>
```

Confidentiality via Encryption

1. The sender creates a message.
2. The sender specifies a set of recipients. This is done by supplying the public key of each intended recipient.
3. The message is encrypted using the sender's private key.
4. The intended recipients and only the intended recipients can view (decrypt) the message.

Encrypting a message

Again, we can encrypt from a file, or the standard input. Ctl+D for EOF.

```
% gpg --output mymessage.asc --armor --encrypt --recipient dvanhorn
```

Now you can email this message to the recipient who can decrypt the message with the complementary private key to the public one you've encrypted with.

Example: Encrypted Message

This is an encrypted message. Mailcrypt is an Emacs Lisp package to interface GnuPG. It can be used with Emacs to compose encrypted email messages. Note the version number and OS are different.

```
-----BEGIN PGP MESSAGE-----
```

```
Version: GnuPG v1.2.1 (GNU/Linux)
```

```
Comment: For info see http://www.gnupg.org
```

```
Comment: Processed by Mailcrypt 3.5.8 <http://mailcrypt.sourceforge.net/>
```

```
hQIOA3Loigy3Yt64Eaf/eeL3Hiwmdem/phlq9QkKOSr4G4isrvvmch01NZQ17+z3m
```

```
WSE0xUT2vZTCm3hBG8GwNkEeXW97oUwMOUqhDktd5PF1GA6hwhgLNapv6zmxIk9C
```

```
:
```

```
04A3FcpHkcRWRe13/mPGJy3cS6JiUD119z0ZRixmEcRg/kLwveeB0NIIVrk30qQm
```

```
fqA2y6oymeSR2kiKcfg15XPBDK/T
```

```
=kekI
```

```
-----END PGP MESSAGE-----
```

Decrypting a message

Again, we can decrypt from a file, or the standard input.

```
% gpg --decrypt
```

Authentication via Digital signature

1. The sender creates a message.
2. The sending software generates a hash code of the message.
3. The sending software generates a signature from the hash code using the sender's private key.
4. The binary signature is attached to the message.
5. The receiving software keeps a copy of the message signature.
6. The receiving software generates a new hash code for the received message and verifies it using the message's signature. If the verification is successful, the message is accepted as authentic.

Signing documents

This will create **mymessage.txt.asc**, a signed, plain-text version of the input file.

```
% gpg --clearsign mymessage.txt
```

Example: Signed Message

This message is signed by Aaron S. Hawley. Feel free to verify it.

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

```
Hash: SHA1
```

```
On Mon, 3 Mar 2003 dvanhorn@emba.uvm.edu wrote:
```

```
> Let's call it a picnic.
```

```
i'll agree to that
```

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

```
Version: GnuPG v1.2.1 (AIX)
```

```
iD8DBQE+ZASvk0vbeuDv+wORAmABAKDq3mh6MYUx+gXDFH1UISCIpyIcogCg0fje
```

```
Z2tgBpHcfxcjNsLMKxX0oIA=
```

```
=0v7R
```

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

Detached signature

Sometimes you don't want the signature to be a part of the original document. You can also create detached signatures.

```
% gpg --output mymessage.sig --armor --detach-sig mymessage.txt
```

Verify signatures

Verify a signed document will indicate whether or not the document has been altered since it was signed.

For a document that contains the signature within it, just do:

```
% gpg --verify mymessage.txt.asc
```

```
gpg: Signature made Thu 10 Apr 2003 12:50:37 PM EDT using DSA key ID 135B049E
```

```
gpg: Good signature from "Your Name"
```

In the case of a detached signature, do:

```
% gpg --verify mymessage.sig mymessage.txt
```

```
gpg: Signature made Thu 10 Apr 2003 12:59:15 PM EDT using DSA key ID 135B049E
```

```
gpg: Good signature from "Your Name"
```

Detailed Look: Confidentiality via Encryption

1. The sender creates a message.
2. The sending OpenPGP generates a random number to be used as a session key for this message only.
3. The session key is encrypted using each recipient's public key. These "encrypted session keys" start the message.
4. The sending OpenPGP encrypts the message using the session key, which forms the remainder of the message. Note that the message is also usually compressed.

5. The receiving OpenPGP decrypts the session key using the recipient's private key.
6. The receiving OpenPGP decrypts the message using the session key. If the message was compressed, it will be decompressed.

Advanced Assignment

Verify the authenticity of the source document for these slides.

The End