ESIR 3rd year

Systems and Data Security (SSD)

Lab Exercise #1

A practical look at encryption with GnuPG

Objectives

— Learn Pretty Good Privacy (PGP) using GnuPGP, a complete and free implementation of the OpenPGP standard
— Encrypt files, share public keys, import public keys, decrypt files.
— Encrypt email

1 The GNU Privacy Guard (GnuPG)

GnuPG (`https://gnupg.org/`) is a complete and free implementation of the OpenPGP standard as defined by RFC4880 (also known as PGP). GnuPG allows you to encrypt and sign your data and communications; it features a versatile key management system, along with access modules for all kinds of public key directories. GnuPG, also known as GPG, is a command line tool with features for easy integration with other applications. A wealth of frontend applications and libraries are available (`https://www.gnupg.org/software/frontends.html`). GnuPG also provides support for S/MIME and Secure Shell (ssh). GnuPG is very well documented, the handbook can be found at

`https://gnupg.org/gph/en/manual.html`.

1.1 Install GnuPG

Depending on your operating system, the instructions to install GnuPG will be different. GnuPG exists for all major operating systems (including Windows). Please search for how to install. For Ubuntu you may use `apt` :

```
$ sudo apt-get install gnupg
```

1.2 Generate a public/private key pair

Run the command :

```
$ gpg --full-generate-key
```

The `-full-generate-key` flag provides you with extended choices. If you wish to accept the default values you may simply use `gpg -gen-key`.

Please answer all of the questions and provide a passphrase to your key. The end of the output should indicate the creation of `.gnupg` directory and the creation of your keys (including their fingerprints) similar to the following :

```
gpg: /home/walter/.gnupg/trustdb.gpg: trustdb created
gpg: directory '/home/walter/.gnupg/openpgp-revocs.d' created
gpg: revocation certificate stored as
 ↪  '/home/walter/.gnupg/openpgp-revocs.d/CB08C2BC955CD3A2EB2AE1C90636F12C6814785D.rev'
public and secret key created and signed.

pub   rsa3072 2023-01-12 [SC]
      CB08C2BC955CD3A2EB2AE1C90636F12C6814785D
uid   Walter Rudametkin <walter.rudametkin@univ-rennes1.fr>
sub   rsa3072 2023-01-12 [E]
```

You can see the files containing these files like this :

```
$ cd
$ ls -l ~/.gnupg
total 12
drwx------ 1 walter walter   88 Jan 12 01:41 openpgp-revocs.d
drwx------ 1 walter walter  176 Jan 12 01:41 private-keys-v1.d
-rw-r--r-- 1 walter walter 2024 Jan 12 01:41 pubring.kbx
-rw------- 1 walter walter   32 Jan 12 01:39 pubring.kbx~
-rw------- 1 walter walter 1240 Jan 12 01:41 trustdb.gpg
```

Note the directories and files, in particular `private-keys-v1.d` and `pubring.kbx`.

You can interact with your keys and keyring using the `gpg` command. For example, to list **all** of the keys you have in your keyring, including those from other people, use the following command :

```
$ gpg --list-keys
```

To list your personal keys, use

```
$ gpg --list-secret-keys
```

Please note : currently your keyring only has your key so there should be no differences between the output of either command.

A single private key ? You are not limited to a single private key. Indeed, in many cases you might want to use multiple keys for different operations or to separate your online identities. GnuPG (and SSH) support multiple keys.

One interesting feature of PGP is the ability to *sign* keys. This means that you can ask a third party you trust, and more importantly, who trusts you, to use *their private key* **to sign** *your public key*. This is a way for them to say "*I believe this person really is who they say they are, and here's my proof*".

Please take a minute to answer and discuss these questions :

- Why is it necessary to sign keys ?
- Can anyone create a key and pretend to be another person ?
- Can you think of a way to make sure that a given key really belongs to the person listed on the key ?
- What do you think are the benefits of signing keys ?

1.3 Encrypt with GPG using public key

Create or obtain a new file, any file you wish to encrypt will do. The following command creates a text file with a credit card number :

```
cat > my-secrets.txt <<EOF

My name is "Firstname Lastname"
My credit card number is 1234-5678-9012-3456
The password for my phone is 42

EOF
```

Let's encrypt the file (but don't specify a recipient) :

```
$ gpg -e my-secrets.txt
```

However, this will result in an error as you didn't specify a recipient or a user ID for who you wish to encrypt this file.

```
1 You did not specify a user ID. (you may use "-r")
2
3 Current recipients:
4
5 Enter the user ID.  End with an empty line:
6 gpg: no valid addressees
7 gpg: my-secrets.txt: encryption failed: No user ID
```

Why is it necessary to specify the recipient ?

Since you don't have anyone keys in your keyring other than your own, encrypt the file for yourself. To do so, please rerun the command and specify the email you used to create your keys. It then asks you if there are other recipients. Just press RETURN to continue without adding more recipients. You should now have an encrypted version of your file that ends in `.gpg` :

```
1 $ ll my-secrets.txt*
2 -rw-r--r-- 1 walter walter 103 Jan 12 02:15 my-secrets.txt
3 -rw-r--r-- 1 walter walter 568 Jan 12 02:27 my-secrets.txt.gpg
```

NOTE : Armored ASCII (whose filename suffix is `.asc`) is the most portable data format `gpg` uses, in contrast to `gpg`'s default binary format (which uses the filename suffix `.gpg`). Unlike this binary format, Armored ASCII can be copied and pasted, into e-mail for example. Most `gpg` commands accept the `--armor` or `-a` flags to manage keys and files!

The `my-secrets.txt.gpg` is a binary encrypted file. Try and open it with `vim` or another editor, you'll see it's gibberish. This is of course not convenient for all cases, as is sending email, since the binary format doesn't transport well. In such cases, you can instead encrypt the file with the armored ASCII encoding (using `-armor` or `-a`), and this time we'll save time and specify the recipient directly with the '-r' flag :

```
$ gpg --armor --encrypt --recipient walter.rudametkin@univ-rennes1.fr
 ↪  my-secrets.txt
```

By the way, do you notice anything ? (Hint : Did you have to specify the passphrase at any point to ENCRYPT ?)
Please check your directory again, you'll now see an `.asc` file. Please open it with an editor.

```
$ ll my-secrets.txt*
-rw-r--r-- 1 walter walter 103 Jan 12 02:15 my-secrets.txt
-rw-r--r-- 1 walter walter 833 Jan 12 02:34 my-secrets.txt.asc
-rw-r--r-- 1 walter walter 568 Jan 12 02:27 my-secrets.txt.gpg
```

Now, you can delete the original .txt file.

```
$ rm my-secrets.txt
```

1.4 Decrypting files

To decrypt a file with GnuPG/PGP, all you have to do is type :

```
$ gpg my-secrets.txt.asc
```

GnuPG/GPG automatically figures out who the file is encrypted for, and checks to see if you are in possession of the private key (you are), and you are prompted for your passphrase :

```
gpg: WARNING: no command supplied.  Trying to guess what you mean ...
gpg: encrypted with 3072-bit RSA key, ID 4D4EA3CA7E4219C0, created 2023-01-12
      "Walter Rudametkin (This is a new key for dellykeikis in walter account) <rudametkin@gmail.com>"
```

Look at the contents of the file `my-secrets.txt` and confirm that they are correctly decrypted !

Pro Tip!
GPG can also do symmetric encryption! You can do symmetric encryption with either OpenSSL or GPG. However, your public/private key pair isn't used : you will be prompted for a passphrase at encryption time, just like with OpenSSL. For example : `gpg --output filename.enc -z 0 --symmetric filename.txt`

Compression
By default GPG always compresses the file. To disable compression use `-z 0` as specified in the example above (for speeding up encryption on very large files, or media like sound, video, otherwise leave compression activated to reduce file size).

1.5 Exporting your public key with GPG

Let's make a copy of our public key, and place it in a text file, ready to be sent to our friends and colleagues. Verify what keys you have in your keyring with `gpg --list-keys`.

Let's make a copy of our public key, and place it in a text file, ready to be sent to our friends and colleagues.

Note : a key can be addressed in one of several ways :
— using the fingerprint (such as `CB08C2BC955`) - this is the preferred method as it's guaranteed to be unique,

— using an email address, for example, walter.rudametkin@univ-rennes1.fr
— using part of the name ("walter"), however even if unlikely, there can be many people called "walter" in your keyring !

We'll use the email address :

```
gpg --export --armor --output rudametkin-public-key.asc
 ↪  walter.rudametkin@univ-rennes1.fr
```

This will produce a file named `rudametkin-public-key.asc`, please open yours in an editor. You should see a long random string similar to :

```
$ cat rudametkin-public-key.asc
-----BEGIN PGP PUBLIC KEY BLOCK-----

mQGNBGO/VzQBDADiQu8ONzLkSkqPBlbmqlqHNhHf7zEuZCWwyGBq0PcESp2pQXZh
LBf8TAXxEKSjf8h/vcwoyoXHLqxdoW1i4kefdEIx60QPo58vQ4OY0MH0RCdl4pLa
ldC7x1PFabHHJXzA3lDLCxU4UdkaY9Gifpftq3aRfGFR2NrfXaG+/TdoTtRM35Vv
[...]
bm2yy7aDg23llhNi+f9CPDrPNcDJ9jc/Irk=
=6MxG
-----END PGP PUBLIC KEY BLOCK-----
```

1.6 Exchanging keys (manually)

Let's stop for a second and think :

— To be able to encrypt files that only a certain person can decrypt, you will need a copy of **THEIR public key**.
— Therefore, if someone else in the class wants to send you an encrypted file or a message, then they will need a copy of **YOUR public key**.

This is a three step approach :

— Import the key of the person you wish to send encrypted files/messages to.
— Encrypt the message/file using the public key of that person (recipient).
— Communicate (copy) the message to the recipient.

So first, find another student to send messages to, and agree with them that THEY should send YOU their public key by copying the file created in the previous step (myname-key.asc). You may use any copy mechanism that you like (usb key, email, `scp`, `cp` from one account to another ...).

What risks and attacks can be associated with the different means of transferring files ?

NOTE : You don't have to send your key to the same person you received a key from.

Once you've agreed who you will send messages to, and which other person you will receive messages from, and how you will exchange them, please do so. Remember, you need only send and receive your **public key** obtained with the `gpg -export` command from earlier.

1.7 Receiving (and importing) someone's key

Once you have received a key, you may import it with :

```
gpg --import theirname-key.asc
```

The command should output something like :

```
$ gpg --import rudametkin-public-key.asc
gpg: key 0636F12C6814785D: public key "Walter Rudametkin <walter.rudametkin@univ-rennes1.fr>" imported
gpg: Total number processed: 1
gpg:               imported: 1
```

Verify the key has been added with :

```
$ gpg --list-keys
```

Note : you can verify that you still only have your own SECRET key in your SECRET keyring with the command :

```
$ gpg --list-secret-keys
```

... you should only see your own key. This is expected : you only imported the PUBLIC key of your colleague.

1.8 Encrypting using the imported key

Now, we will encrypt the file "my-secrets.txt" with the PUBLIC key we have just imported :

```
$ gpg --armor --encrypt --recipient walter.rudametkin@univ-rennes1.fr my-secrets.txt
```

The command should output something like :

```
$ gpg -a -e -r walter.rudametkin@univ-rennes.fr my-secrets.txt
gpg: B42B8E10E28F30D8: There is no assurance this key belongs to the named user

sub  rsa2048/B42B8E10E28F30D8 2018-06-16 Walter Rudametkin <rudametkin@univ-rennes.fr>
 Primary key fingerprint: 36FE 2B33 AEB5 2648 0E50  66CF DAEC D341 6536 E942
      Subkey fingerprint: E274 D0D4 D32C F903 268C  BF37 B42B 8E10 E28F 30D8

It is NOT certain that the key belongs to the person named
in the user ID.  If you *really* know what you are doing,
you may answer the next question with yes.

Use this key anyway? (y/N) y
```

Notice how you are being informed that you have no proof that this key really belongs to this person... **Think about the implications!**
Now send the encrypted file to whoever's key you used!

1.9 Decrypting a received file

Copy the encrypted file somewhere and run the following command (change the name to match the file) :

```
gpg my-secrets-from-sender.txt.asc
```

```
gpg my-secrets-from-sender.txt.asc
gpg: WARNING: no command supplied.  Trying to guess what you mean ...
gpg: encrypted with 2048-bit RSA key, ID B42B8E10E28F30D8, created 2018-06-16
      "Walter Rudametkin <rudametkin@gmail.com>"
```

Check your directory and read the decrypted file ! If the correspondent sent you a file encrypted for someone else, you would see the following error :

```
gpg: encrypted with RSA key, ID 3BE8FE75
gpg: decryption failed: secret key not available
```

If you compare the keyid with that of your own key (`gpg -list-keys`), you would see that it doesn't match : this file was encrypted with someone else's public key !
If it DID work, congratulations !

2 Keyservers

A key server is a computer that receives and then serves existing cryptographic keys to users or other programs. The users' programs can be running on the same network as the key server or on another networked computer. Keyservers can be public or private, and can share their keys with other servers. In PGP, the keyservers contain and share public keys that have been uploaded to (and in some cases validated by) the service. The keyserver thus distributes the keys and allows users to easily import them.

There are many PGP keyservers. Each keyserver can have different requirements for uploading a key. For example, some will require the email address be validated, while others might only publish keys submitted through their web interface. Here are three popular keyservers and their main properties :

- `keys.openpgp.org` : central, verification of email IDs, keys can be deleted, no third-party signatures (i.e. no Web of Trust support).
- `keyserver.ubuntu.org` : federated, no verification, keys cannot be deleted.
- `pgp.mit.edu` : federated, keys cannot be deleted.

For this section we propose using the `keyserver.ubuntu.org` keyserver since it does not require email verification and the keys are available quickly after upload.

`~/.gnupg/gpg.conf` . You're operating system likely provides a default keyserver but you can also specify a specific keyserver in each command, or you can change your default server by editing the `~/.gnupg/gpg.conf` file and adding the line `keyserver keyserver.ubuntu.org`.

2.1 Delivering your public key to GPG Keyservers

Find the fingerprint of the key you wish to submit to the server using :

```
$ gpg --list-keys
```

Then submit the key using the following command :

```
$ gpg --keyserver keyserver.ubuntu.com --send-keys CB08C2BC955CD3A2EB2AE1C90636F
```

Once submitted, you can search for your key using the same fingerprint, the email address, and some keyservers even support searching by name :

```
$ gpg --keyserver keyserver.ubuntu.com --search-keys CB08C2BC955CD3A2EB2AE1C90636F
$ gpg --keyserver keyserver.ubuntu.com --search-keys rudametkin
$ gpg --keyserver keyserver.ubuntu.com --search-keys vnixroot@netscape.com
```

2.2 Importing a key from a Keyserver

Once your public key has been uploaded, anyone can import it through either a search, your email address or the key's fingerprint. For example, using search :

```
$ gpg --keyserver keyserver.ubuntu.com --search-keys rudametkin
```

```
1 $ gpg --keyserver keyserver.ubuntu.com --search-keys rudametkin
2 gpg: data source: http://162.213.33.9:11371
3 (1)     Walter Rudametkin <rudametkin@gmail.com>
4           2048 bit RSA key DAECD3416536E942, created: 2018-06-16
5 (2)     Walter Rudametkin (This is a new key for dellykeikis in walter account
6           3072 bit RSA key 0636F12C6814785D, created: 2023-01-12
7 Keys 1-2 of 2 for "rudametkin".  Enter number(s), N)ext, or Q)uit >
```

If you select 1 or 2, the key will be imported.

You can also specify and import the key using the fingerprint, for example :

```
$ gpg --keyserver keyserver.ubuntu.com --receive-keys CB08C2BC955CD3A2EB2AE1C90636F
```

2.3 Getting more keys from your colleagues

Import a few keys from your colleagues. Compare that to the manual process performed earlier. Which is easier ? What safeguards are in place ? How can you verify their identify to ensure it's the right person ?

You can add multiple recipients when encrypting a file. Try doing that, such as

```
$ gpg --armor --encrypt --recipient rudametkin@gmail.com --recipient
 ↪  rudametkin@gmail.com --recipient walter.rudametkin@imag.fr my-secrets.txt
```

3 Other uses of GPG

3.1 Sign your commits

You can use GPG to sign your commits, given users the possibility to authenticate your work. This is mandatory in many companies and open source projects and highly recommended in general.

The official documentation to configure your Gitlab account with your GPG key is available at `https://docs.gitlab.com/ee/user/project/repository/gpg_signed_commits`.
The official documentation to configure your Github account with your GPG key is available at `https://docs.github.com/en/authentication/managing-commit-signature-verification/adding-a-gpg-key-to-your-github-account`.

3.2 Sign and/or encrypt your email

Many email clients support GPG. You are invited to see if the client you use supports PGP. However, we strongly encourage you to use `Thunderbird` with the `Enigmail` extension. The instructions are here : `https://www.enigmail.net/index.php/en/user-manual/installation-of-enigmail`