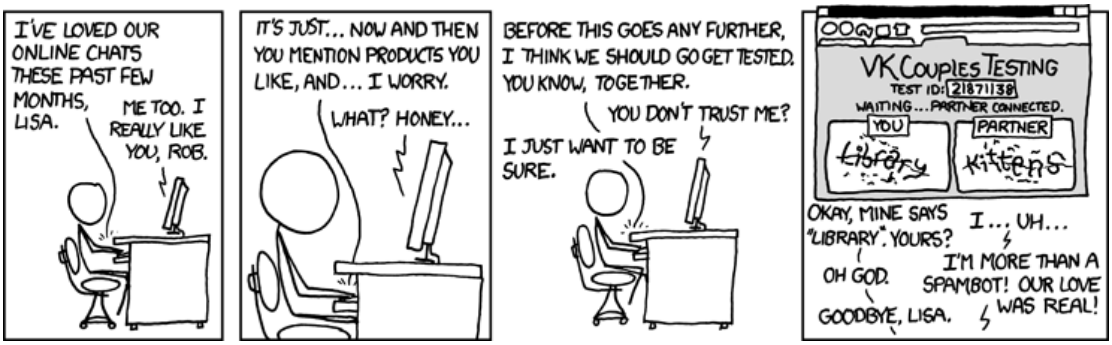
Encrypt, Decrypt, and Authenticate

Lab 2 in TSIT03 Cryptography,
Institutionen för systemteknik, Linköpings universitet

Niklas Johansson*

September 11, 2017



– http://xkcd.com/632/

*niklas.johansson@liu.se

1 Before the lab

Kick your feet back and remind yourself about the lectures regarding RSA and public key (asymmetric) cryptography.

2 During the lab

This lab is *not* intended to be a problem solving exercise, but rather to act as an assurance that all participants of the course have had a hands-on experience with asymmetric cryptography. Therefore, for some of you who are already familiar with the subject, this lab will be finished well in time.

During the lab you will generate your own key-pair and learn how to use it. The following sections will work as a step-by-step introduction, and you will be answering questions along the way. The lab will be performed in groups of two, but only one of you will be the owner of the key-par (the one logged in to the edu. network).

2.1 Generating key-pair

To generate the key-pair you will use the pre-installed software GnuPG. Head to a terminal and write:

```
>> gpg --gen-key
```

GnuPG will then prompt you some questions about the generation parameters.

- The type of key you want: `RSA and RSA`,
- the key length: `4096`,
- the expiration time, default is "`key does not expire`",
- your real name: `Your Name`,
- your email address: `your@address.com`,
- a comment: `"My student mail"`,
- and a passphrase used to protect your key.

Question: What is you user ID constructed from?

The keys are generated in a hidden directory `.gnupg` in your home folder. You can view it by navigating to your home folder and type:

```
>> ls -a
```

To see the file permissions for this folder, write:

```
>> ls -la
```

The first character indicates whether it is a file (`-`) or a directory (`d`). The next 3×3 characters specify the file permissions for the *user*, *group*, and *others* respectively.

Question: What are the permissions for the `.gnupg` directory, and what do they mean?

2.2 The revocation certificate

You will need a way to invalidate the key if it gets compromised. Therefore, you will now generate a revocation certificate.

```
>> gpg --gen-revoke your@address.com
```

This command will ask a number of questions before printing the certificate to the screen. If you instead want to write it to a file, say `revoke.asc`, then type

```
>> gpg --output revoke.asc --gen-revoke your@address.com
```

The dialog will ask you the reason for why you want to revoke the key, this is a bit weird since the certificate should always be generated ahead of time, just ignore

that and proceed. Generating the revocation key should always be the second next thing you do after generating a key-pair, because if the key actually gets compromised, then it may be too late. The revocation certificate should be stored where others have no access, since anyone can published it and render your key useless.

Question: Is it a good idea to store the revocation certificate on the same device as your key-pair?

2.3 Exchanging keys

You will now, with Pretty God Privacy, communicate with other lab groups. And to do that, you must exchange public keys. To get hold on your public key you export it from your keyring using the command-line option `--export`

```
>> gpg --export your@address.com > your_public.gpg
```

The key is exported in binary format, which is not very well suited as a visual representation. By also giving the option `--armor`, `gpg` will use a binary-to-ASCII encoding to output a more readable file.

```
>> gpg --export --armor your@address.com >
your_public.gpg
```

The output from the above command will be something like:

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version:  GnuPG v1

mQINBFe3X4MBEAC3j6hsjqoM/p0s05wXi6CqMWpdsWYDeojWe8ZuPn4+Kff+jKzF
X1dwauXeHtePAhckaISZqM0vKa83nSF7VoA08PXaF9PnvGTCZhwKkiqs9awI+3z4
kPbK6mdcwWGZ87lAU2Vx5fFcYQKkwQZQ/YT45e1dKEzmk19bS8Ps/bYYbHg643mv
vzNd4W1RmrTjBebA+wGl9v0ZBTY5JYvGzpajm/o3xIR8H1ba4U+Oq/0VM6zMf...
```

```
...gIMwOyRVTPUZSHqs7M97EN+4/tU0vAs+Fa8MuuDtI+YRnNmGsAeNorwARAQAB
c29uQGxpdS5zZT6JAjgEEwECACIFAle3X4MCGwMGCwkIBwMCBhUIAgkKCwQWAgMB
q0U0nvKwTGCnYXG7oSt9SPaDSmRPDSoZak5s0S0SyJ8H+buC8c3iFeDzmWhmb86N
SAfkjdO2hR8kYjpZdG3IaTjOID2CC+rWHl85l9nXzJxuqh5Um5HqwW+mgl1Nk26l
=yPDw
-----END PGP PUBLIC KEY BLOCK-----
```

Send your public key (how does not matter) to the lab group/groups you want to communicate with.

To import a key:

```
>> gpg --import your_friends_public.gpg
```

And to view the keys available in your public keyring:

```
>> gpg --list-keys
```

Once you have imported a key it should be validated. This is done by first generating a fingerprint from the imported key, and then verify that to be in a one-to-one correspondence with the fingerprint generated by the owner.

```
>> gpg --fingerprint friend@address.com
```

If all checks out, then you can be confident that you have a correct copy of the key, and you certify that by signing the copy with your own key.

You should be very careful to *always* verify the key's fingerprint with the owner before you sign.

```
>> gpg --sign-key friends@address.com
```

Question: If you send your public key together with the fingerprint that goes with it, do you open yourself up to a man-in-the-middle attack?

2.4 Encryption and decryption

To encrypt and share a document you need to have the recipient's public key signed in your public keyring. (If it is not signed it will work anyway, after a few security questions have been prompted and bypassed, but it is not a good practice.) Signing a key tells your software that you trust the key that you have been provided with, and that you have verified that it is associated with the correct recipient.

If you want to encrypt the file `plain.txt` only for `friend@addres.com` to read, then type

```
>> gpg --encrypt --output cipher.txt --recipient
friend@address.com plain.txt
```

which will create the encrypted `cipher.txt`. The recipient may then retrieve `plain.txt` by the command-line

```
>> gpg --output plain.txt --decrypt cipher.txt
```

You should now transmit/receive a message (perhaps by mail) securely to/from another lab group.

Question: What were those messages transmitted and received from the other lab group? (If they are long, show them onscreen to the assistant)

2.5 Signing and authenticating

There are three basic ways of creating signatures: `--sign`, `--clearsign`, and `--detach-sign`. Typing

```
>> gpg --sign doc
```

will create a compressed file of the document together with the signature called `doc.gpg`. To verify the signature, use

```
>> gpg --verify doc.gpg
```

and

```
>> gpg --output doc --decrypt doc.gpg
```

to verify and retrieve the document.

The command-line option `--clearsign` does not make a compressed file, instead it concatenates a ASCII coded signature to the document.

The `--detach-sig` will create the signature in a separate file. Both the detached signature and the document must be given to `--verify` as arguments; the signature first and the document second.

```
>> gpg --verify doc.sig doc
```

At the course page the assistant's public key is available. Verify that this document `RSA_lab_pm.pdf` have been signed by that key.

Question: At what time was this document signed?

Question: Can you be sure that the key belongs to the assistant?

2.6 Web of trust

Up to this point only keys that you have personally verified have been considered valid. The web of trust is a construction that allows for a more relaxed validation. Say, for instance, that you trust a friend to know how to (and carefully enough) verify keys before signing them, then you could choose to trust keys that your

friend has signed as well. To make things even more interesting, `GnuPG` has four levels of trust for you to use, privately grading owners in your public keyring. The levels are

1. **None:** the owner is known to improperly sign other keys.
2. **Unknown:** nothing is known about the owner's judgement in key signing. Keys on your public keyring that you do not own initially have this trust level.
3. **Marginal:** the owner understands the implications of key signing and properly validates keys before signing them.
4. **Full:** the owner has an excellent understanding of key signing, and his/her signature on a key would be as good as your own.

To rank a friend according to these levels, type

```
>> gpg --edit-key friend@adress.com
```

and when prompted to give a second command, write `trust`, make your decision, and then type `quit`.

Now, a key k can be considered valid if it fulfills the following two conditions:

1. it is signed by enough valid keys, meaning
   - you have signed it personally,
   - it has been signed by one fully trusted key, or
   - it has been signed by three marginally trusted keys; and
2. the path of signed keys leading from k back to your own key is five steps or shorter.

The number of fully and marginally trusted keys, as well as the distance between k and your key are set per default, but can be adjusted.

For this to work, public keys must carry information about who have signed it. You should therefore return a copy of the key to its owner after you have signed it. Exporting a signed key will carry the signature, and the owner updates his/her key by importing the signed copy. Note that the trust levels that you may have assigned to the key is regarded as private information and is not exported.

You can view the signatures of keys in you keyring by

```
>> gpg --list-signs
```

Question: Have you returned the signed keys in your keyring to their owners?

3 After the lab

If there is any time left you should engage in a key-signing party with your fellow students, i.e., you should authenticate and sign the keys of other students. This will strengthen the web of trust.

To make your key publicly available you can upload it to a key-server

```
>> gpg --send-keys --keyserver specific.server.com key_id
```

where `key_id` is the eight character string listed before the keys creation date in `--list-keys`. The command `--keyserver` is used to specify the server, if not used, it will default to `keys.gnupg.net`.

– http://xkcd.com/364/

```
CB54 B107 239D D604 FF41 4E19 5C6F A38F 239A D40D
```