ROYAL INSTITUTE
OF TECHNOLOGY

School of Electrical Engineering and Computer Science
Division of Theoretical Computer Science

LAB G

GPG: Secure e-mailing

Computer Security

DD2395 / HT2022

r-2
c202209021803

Contents

1 Introduction[1]

In 1991, Phil Zimmermann developed an e-mail encryption software called Pretty Good Privacy, or PGP, which he intended peace activists to use while organizing in the anti-nuclear movement. Today, PGP is a company acquired by Symantec that sells a proprietary encryption program by the same name. OpenPGP is the open protocol that defines how PGP encryption works, and GnuPG (GPG for short) is free software, and is 100% compatible with the proprietary version. GPG is much more popular than PGP today because it's free for everyone to download, and probably more trustworthy because it's open source. The terms PGP and GPG are often used interchangeably. Each person who wishes to send or receive encrypted e-mail needs to generate their own GPG key, called a keypair. GPG keypairs are split into two parts, the public key and the secret key (the latter is sometimes called private key as well). If you have someone's public key, you can do two things: *encrypt messages* that can only be decrypted with their secret key, and *verify signatures* that were generated with their secret key. It's safe to give your public key to anyone who wants it. The worst anyone can do with it is encrypt messages that only you can decrypt. With your secret key you can do two things: *decrypt messages* that were encrypted using your public key, and *sign messages*. It's important to keep your secret key secret. An attacker that has access to your secret key can decrypt messages intended only for you, and she can forge messages on your behalf. Secret keys are usually encrypted with a passphrase, so even if your computer gets compromised and an attacker steals your secret key, she would also need your passphrase in order to be able to use it.
In this lab you will use GPG to encrypt and decrypt e-mail messages, to sign messages, verify others' signatures and check the validity of the keys used for these operations. The lab can be fully done **remotely**. All you need to do is to send five e-mails to a dedicated server that will automatically check the submitted e-mails and notify if they are correct. You can, however, come to the scheduled lab sessions and get help. The lab should be solved **individually**, which means that you are not allowed to solve the lab for another student.

Deadline

The deadline for the lab can be found on the course website
`https://kth.instructure.com/courses/35479`
It applies for the steps you have to perform against the automatic checking system.

1.1 Preparations

If you use the GNU/Linux environments in the CSC lab rooms (e. g., Ubuntu), GPG is already available and you can use the command `gpg` in a terminal right away. If you are using your own machine, most GNU/Linux operating systems have GPG already installed. For Windows, you can either install Gpg4win (`http://www.gpg4win.org/`) and use `gpg` from the command line or install windows subsystem for linux (`https://docs.microsoft.com/en-us/windows/wsl/install-win10`) with any Ubuntu installation available and use `gpg` from there, for Mac OS X you can download GPGTools (`https://gpgtools.org/`).
GPG itself does not come with a graphical user interface. You can test if it works correctly by typing `gpg --version` in a terminal. The output should look similar to the following listing:

[1] The introduction is partially based on the PGP section in Micah Lee's publication "Encryption Works: How to Protect Your Privacy in the Age of NSA Surveillance" (`https://freedom.press/news-advocacy/encryption-works-how-to-protect-your-privacy-and-your-sources-in-the-age-of-nsa-surveillance/`).

```
gpg (GnuPG) 1.4.16
Copyright (C) 2013 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
...
```

Listing 1: GPG version output

GPG GUI

For this lab, it is not recommended to use a GUI application instead of a command line for GPG. It defies the goal of the lab: understanding what is happening under the hood as the messages are encrypted, decrypted, and signed. So it is counterproductive. Also, the bonus point of Lab G requires on the command lines that you have used throughout the lab, however the GUI does not give you the opportunity of writing them.

There are several GPG front-end applications for e-mail clients such as the EnigMail add-on for Thunderbird, GPGMail for Apple's Mail or scripts for integrating GPG in Pine. For the lab, we **recommend to use GPG from the command line** and to send messages using Thunderbird or KTH's webmail. Note that you need to set the format of your e-mails to **plain text and not use HTML or rich text**. If you are not familiar with the command line, you find some useful commands in Section 6.
We have not tested the lab for other setups and many of the tools and front-ends found on the web are broken in various interesting ways, including giving you limited, conflicting, or even completely wrong output. We are aware of some front-ends not reporting all keys used in signing and encryption, automatically importing keys and other bad stuff. Many clients also mess up the formatting of e-mails (we are aware of issues with Gmail's webmail for example), preventing correct decoding of the data, which in turn, will not allow you to successfully complete the lab; therefore, the course administration cannot help you with software issues you might encounter, except for those related with using the latest version of GPG and the above mentioned e-mail clients.

In the next sections, you will go through a series of tasks and interact with an automatic checking system. These tasks are sequential, you will get input for the next step once the current step is completed.
You will find a checklist of questions to think about inside the information boxes at the end of some subsections. This should help you to understand what is going on in more depth. The milestone boxes are there to help you keeping track of having completed all necessary steps to pass the lab.

Important

When you send messages to the server and you cannot explain an error message, remember to try the following first. This rules out formatting issues. There are also erratic encoding issues when your web interface is set to a language other than Swedish or English but, e.g., Chinese.

1. Go to KTH's webmail.
2. Create a new email, don't reply to an email.
3. First, change to plain text encoding and add the recipient accordingly.
4. Then, copy your message from a text editor that doesn't support formatting or the terminal, and paste it into the browser.
5. Finally, send the email.

If the steps above don't get the problem fixed, please check Section 5.

2 Key management

The following section will help you understand the quirks of key management, including key creation, signing, etc. You will generate your own pair of keys, import the course public-key from the course web site, send your public key to the server for signing, add additional user identities to your key, get your public key signed by another classmate, and send the result again to the server for verification.
You need to complete all tasks described in the following subsections.

2.1 Creating keys

To be able to decrypt and sign messages you need to generate your own key pair. GPG lets you tune the cryptographic security of the keys by choosing a specific key length. The longer the key the more secure it is against brute-force attacks but encryption and decryption will be slower. The website `http://www.keylength.com/` offers recommendations for suitable key-lengths depending on the security required.

Key Settings

When you generate the key, you can choose the **default cipher** algorithm (RSA) with the recommended key size of 2048 bits, but make it **valid for at most 3 months**. The other parameters of the key should be as follows:

- **Real name**: Your name and last name(s).
- **E-mail address**: Your account at KTH (`your-username@kth.se`).
  Note that the system will not send any reply if you do not use a KTH e-mail address.
- **Comment**: Set the course code (`DD2395`).

Finally, you will protect your key with a **passphrase** when you generate it. This passphrase will prevent anyone from using your key without your authorization.

Key generation can take from a few seconds up to few minutes; if you get a message asking you to generate more randomness, either type on the keyboard or move the mouse around. You can also open new terminals on the same machine you are trying to generate the new key and run commands such as `ping kth.se` or `du -h /`

Think about...

- What command did you use to generate the keypair?
- Which user identity/ user identities are assigned to the key?
- What is the fingerprint of your key?
- What subkey(s) are attached to your key? What is the purpose of the subkeys?

2.2 Importing keys

In the subsequent sections of the lab you will need to verify e-mails sent from the server. For this you will need the course key (public-key) from the course website: `https://kth.instructure.com/courses/35479`, or alternative from our dasak server: `http://dasak-vm-lab-server.eecs.kth.se/labg/coursekey.html`

Download the key, import it to your keyring and verify its fingerprint. Usually you would compare the fingerprint using a more trusted channel like meeting in person or making a voice call, but for this exercise it is ok to trust the information on the course website. Note that good security can only be achieved when comparing the whole fingerprint because it is possible for an attacker to create a fake key that starts or ends with the same characters as the original (see Section 4 for more information about this issue). When you successfully verified the fingerprint you should sign the course key with your secret key (so you can detect unauthorized changes to your keyring later). Use a local signature for this purpose.

Think about...

- What commands did you use for importing, checking and signing the key?
- If you would not trust the fingerprint information on the course website but you would trust Sonja Buchegger's (buc@kth.se) key, which is available from the course website but also from her personal website, what steps and commands would you need to perform to check the course key's validity?
- What is the difference between a normal and a local key signature?

2.3 Submitting keys

Export the (public) key that you generated and paste it into an e-mail to `gpg-key@dasak-vm-lab-server.eecs.kth.se`, do not send it as an attachment. While there are plenty of format schemes, it is usually a good idea to use ASCII armor (`--armor`). This is an encoding that avoids characters known to cause problems with e-mail systems. Remember to use plain text as format for all your e-mails to our server. Also, make sure you send it from your e-mail address at KTH (@kth.se). There should **only be a single identity** in the key, with your address at KTH (@kth.se). If the submission is successful, you will get your key signed with the course key in return (from `dasak-gpg-key@sys.kth.se`). Note that it might take up to an hour before you receive the response due to KTH's grey-listing. After receiving the signed key, import it again to your keyring to update your local copy of the key with the new signature.

Think about...

- What commands did you use for exporting the key and importing the signed key?
- What command can you use to list all signatures on your key?
- The default for exporting a key is to only export the public key. In what situations would you want to export your private key?

Milestone
Make sure you received an e-mail with your key signed by the course key and import it to your keyring. □

2.4 Augmenting identities

It is quite common to have multiple identities (name, e-mail address combinations) bound to the same key so that your contacts do not need to keep track of several keys in case you are using more than one e-mail address. So your

next task is now to add an additional identity to the key with a second e-mail address (e. g., your private address if it is not the one provided by KTH).

Think about...

- What command(s) did you use to add an identity?
- How can you list all identities of a key?

2.5 Signing keys

By now you have your key signed by the automatic checking system, but this system has not performed authentication (checked whether you are who you say you are) before signing your key. To give the system more evidence that you are not an imposter, let another user sign your key. So find a classmate and get her/him to sign your key.

Think about...

- What steps and commands did you and your classmate need to perform in order to sign your key?
- Should the system trust your key now, after you presented another signature on it? Why/why not?
- When you sign another classmate's key, you should compare the fingerprint before signing it. Why is this important?

At this point you should have your key signed with the course key, but it is also now signed by your classmate and contains multiple identities.

2.6 Re-submitting keys

You will now export your double signed public key (again using ASCII armor) and submit it to `gpg-key@dasak-vm-lab-server.eecs.kth.se`. Remember that the source e-mail address should be your address at KTH.
Before submitting the exported key, you can double-check its content using the following command:
`gpg -v KEYFILENAME`.
The server will check the submitted key and respond with an e-mail containing the status of the verification. If your key was accepted the server will send three more e-mails for the next tasks.

Milestone
Make sure you received a confirmation e-mail stating that your key was accepted (plus three more e-mails for the next task). □

3 E-mail encryption and signing

The following section will help you understand the quirks of sending and receiving secure e-mails by signing and encrypting the content.

By this point of the lab you should have received **four** e-mails, one status e-mail saying that the double signed key is accepted, one with signed data, another one with encrypted data, and the last e-mail with both signed and encrypted data. Listing 2 shows an example for an e-mail with signed data.

Each e-mail contains several messages that are independent. These messages can be encrypted or/and signed. They are separated with lines equal to `============= separator =============`. The separator lines are **not** part of the messages. So in Listing 2, line 33 represents one message, lines 35 to 49 another one, line 51 another one, and so on. The reason for including several messages in a single e-mail is simply to save you from getting about 30 e-mails in your inbox.

Each message contains one random string (e. g., `3d8b2bb98d9adb541b1e4681ab22918abf583498`), representing the plaintext data we want to communicate securely. You will have to return the strings that were properly secured in order to prove that you have verified the signatures correctly, decrypted the strings correctly, or both.

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v1.4.10 (GNU/Linux)

mQENBE6tk1sBCADFTOlNAUZwl+jkb27PbEPLp6zQBtrDlTVvsoRHMipE2z0Vv3qd
UHXaknDQ9xxOjKZSkMmaPQKNjVhQxpRfcmlIafRtgxX0CymqFBQNrZxa9ay22lqI
wggJpY2AZWpPmIvbJ13TZ8xFotO/nAsXAQFeprhK53RCvqoMFoy9z6iWTqFObCwm
Yfrem1uXX+SktBTkT16OpnC35jY9OyUU4DNHVOEY0T5YGxy7PUlv3JBl1YtKfX0z
ffXRUwzcHh/K0ZuFAPmqYH/8KtoWN3sFJu6iMdIAimDBiIykeQE05IRYk0F2UlZJ
7FJ7uK5GKTevvwaBcq5x1dchZxcuN9VHZddRABEBAAG0OENvbXB1dGVyIFNlY3Vy
aXR5IChjb3Vyc2Uga2V5KSA8Z3BnQGRhc2FraDExLmNzYy5rdGguc2U+iQE+BBMB
AgAoBQJOrZNbAhsDBQkJZgGABgsJCAcDAgYVCAIJCgsEFgIDAQIeAQIXgAAKCRC4
+uj6J2H//jhSB/9VGtQSh6WwjQVoxtJJ6J1x13Xv0mccImwFum5fuKrV5JZSYd5S
wKG/MS+u3sN+2TpItMZBudpn6wX2u2GuWxu5Xvbs0tAuUT05KRQb30sayDDqclOB
3fdioIyX40HlC5120kgnm1ubWKQVak/4Am4aDRrxbxjINhXZkQnpZty9GN/uYss1
kUJbI2PgQm940SDfmCh0tm3QgMqU+G+B61lixqMWxLKnuT5ytZ0kfZHJ0K2UQc4w
sMYXfY8y6WTgcEQ35zy+4pudrXhzflvL6L16eJ/ChjfNwJ4IXX+4i87BogVFZs3i
sO7mxiGumfTm1jiN03ah6I1negHsUA0nKwEOuQENBE6tk1sBCAC246INHzhntKT0
rJ7U7GshAjzj+mcO3DebYpmEc3BL0FiasDwQ+rTHegq3cEJIibBTzWfLmhntp2Rl
DH4k7r6MuTpRD0cgJlMvWO5uiOTo3qdouqW3uxRGy4hWxCn1OxSE3QBq1DErQsyf
IZR7d77h64O8rmhyhmaZwTuKkFMh1mEtdY7i9+M2r49StPQ2N69p9lVhST2Ps+qS
N9b68WS7sW/pm2ekXbcs+UN927a3WUT5Uh/p+vi2BY3FwlE6yBTkYIlrmMHfzEvH
xaCvAbpvXhWUvn9cXEiI12CNxAtQ2Xe6rQtDCmOvXG85L4T8+fkfLQifqRtGf0mG
i2gxBwnzABEBAAGJASUEGAECAA8FAk6tk1sCGwwFCQlmAYAACgkQuPro+idh//6r
Mwf+K9Tcn66bpKTlrPwo7RHMFQox4rwG9YypoDyzKlPtYQkY0PgOXd8K8bKqHZXG
05lJjBei9xhZPH8yrc2sI9Z05im8/syRwyM9b/+NbyxiA7iw6yP6JHNQsB7mvWub
96vtLh4QXZZxYjImjyJsR35XgDWMHGoU1BkbUGKYPkLNDWOtHT/vUc+KsYCb0SHY
7cLS5ocm3fPtB22SrrTsVy4YjJMEZ4XUbXjw7qKJxlFDUwDMW6/ko8Gh7yxw0TZ/
BruQIj3AebHHBZ2XuMTcbvnszxEuDLvFP2vzLuKANgo6apDVbPHgOC4oKmkbSM9w
OEAUlxsmxkmVlaNa9CTAK63jig==
=cJDK
-----END PGP PUBLIC KEY BLOCK-----
============= separator =============
4dc20e6a50ce08203e2d29fdb806cdbb91169a9b
============= separator =============
-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA1

c2c4dba004abeae7cbee7ee37ffc42509e25b803
-----BEGIN PGP SIGNATURE-----
```

```
Version: GnuPG v1.4.10 (GNU/Linux)

iQEcBAEBAgAGBQJOstHwAAoJEFf2Co+pBNejE7UH/iwkQhAr5WnR6xlYuKXjQWvt
nxrLPS0JyK4PZvsuxZYiRZyH4yYDGgQg6gvesma2+X/rINMZGzCp6Yi6BQOgpimP
K4cmfHBWAC2yIcsFAKfU3LFJ7m4jqEz/6JD8bnxsKVkiQqMnOvJqQoQTzNuVWyUi
0vyg+NvkDIG0Ic7oGrL2ZCQt6A7ukCol3XIh1Lj5+r0ytuZ61+rrKgvBZw/H1jMZ
kM/Lqma52n67dsbofXDVVJu3n4Ox/tNi6CxrgEQ6HFxAt0IsWGvJCpQllWKEagsn
pBLar+gMM3szynp1SRgGJN6MEb9xw04a9mID9VMn50aa28wNw4eYT3Tx0GPUnWY=
=sEMB
-----END PGP SIGNATURE-----
============= separator =============
3d8b2bb98d9adb541b1e4681ab22918abf583498
============= separator =============
-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA1

77b95b92b02541079b8b06dfdf0e58ea0c61d5f5
-----BEGIN PGP SIGNATURE-----
Version: GnuPG v1.4.10 (GNU/Linux)

iQEcBAEBAgAGBQJOstHwAAoJEFf2Co+pBNejlPEH/0n3leNNqic8qOIgh3ZTI3cx
ZdLH+AGRCLIlyCgZAlH0I9oV7gg7hSnnCPU+5Px3LCp4QgAISgIdsnPudpHd6SBl
U/VQM0zxNfUl3auVz0+oItllH26lqSgdlZ7MLPWhTJxqAFf0L9HN4+A72UqIp4I6
IgjjCanmkZ2L6HsBC5NBPmxnud48x/XyK+oVnIdzLoqVfvzZD1u8SZ48HTS1GXq5
tI3F7Cv8+rnelRV/0Dg/1AzQETbD8E5IWIe252PGd9GnrpPOHy6ix0B6a8/3gGVQ
cHuTeAW6JGhPsggXq9AqTP3NFyi7tbAjowrmN4j5/r2/3/yevdgqX4aQDi8ZTGg=
=Y1Fl
-----END PGP SIGNATURE-----
...
```

Listing 2: Example e-mail with signed messages

You should check which of the random strings were secured correctly, and then return only these strings to the server in the reply e-mail. In the example of Listing 2 only three strings were signed correctly:

```
c2c4dba004abeae7cbee7ee37ffc42509e25b803
44fc571ad1ff6be21e886ed50ba41c602b6c93c9
fc39f05607e925a82ec2c23d6103e265f03a2504
```

Listing 3: Example list of correct strings (before signing)

Before returning the correct set of strings to the server, you must sign or/and encrypt your reply (depending on the task you are solving). In the example of Listing 2 the response has to be signed, so the reply message should look like the following:

```
-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA1

c2c4dba004abeae7cbee7ee37ffc42509e25b803
44fc571ad1ff6be21e886ed50ba41c602b6c93c9
fc39f05607e925a82ec2c23d6103e265f03a2504
-----BEGIN PGP SIGNATURE-----
Version: GnuPG v2.0.17 (MingW32)
Comment: Using GnuPG with Mozilla - http://enigmail.mozdev.org/

iQEcBAEBAgAGBQJOsxnTAAoJEBirngyjOjx6pNgIAI14+r9Elx8H27ede7oQ/bt2
rFHj3Jqa1tOvPfrD5iKZV3BfMJbABme1eJGpOoiUHrOZ2Y96BfGAC1kaCvOn45Ci
WBzh/1RY7cq39e4Hpnyu+PcY06jwZmyW+Zatt+HkVVX9YLCJRK/XIH8gfjgNo6Kp
```

```
5YHQlsQoI9gLnbeM315wRjRWvklYkW2seO5Z80soy3nkZRfwAAdqcBZNWZ3RseX/
iaycutxs8D2M+UMsaG8QrDBrSzeiKpTxUrHGAh18MrNC9R2mdjDZjOApFiVX0HLO
6KMVq87Q+c88eKqoDBwKaxDRM/7JAzd7S01ul7RHqfSMT+Ey08Y3UF6sKFXtshU=
=OEfH
-----END PGP SIGNATURE-----
```

Listing 4: Return message after processing (signing in this case)

In total you will send three e-mails to the automatic checking system, each one of them containing the correct set of strings of each e-mail you received. You will have to prove that you understand which parts of the e-mail you received are secure and which parts are not.
The checking is quite thorough. While the system will accept some extra whitespaces (outside strings), any other extra or missing data will get your submission rejected.

Requirements for passing this lab

You will know that you are finished with the lab when you have received a 'correct' e-mail reply from the automatic checking system for each of the e-mails you send. That is, one 'correct' e-mail for successfully completing Section 3.1, another one when you successfully complete Section 3.2, and a last one when you successfully complete Section 3.3.

3.1 Message signing

You have received one e-mail from `dasak-gpg-sign@sys.kth.se` with signed data. Check each message part in that e-mail if it represents a string that was properly signed by the course key.
Now send a new single e-mail to the `gpg-sign@dasak-vm-lab-server.eecs.kth.se` address that contains a list of correct strings (one per line). The message you send also has to be **signed by you** (using clear sign makes it easier to see what you are doing). Remember to use ASCII armor for the output format and to send your reply message inline (i. e., in the e-mail body, not as file attachment).

Think about...

- What command(s) did you use to verify a signed message?
- What does it mean, if a message is correctly signed?
- Which key did you use to sign the reply?

Milestone

Make sure you have received a confirmation e-mail stating that your signed response was correct. ☐

3.2 Message encryption

You have also received one message from `dasak-gpg-crypt@sys.kth.se` with encrypted data. Now send an e-mail to `gpg-crypt@dasak-vm-lab-server.eecs.kth.se` with a list of all strings that were properly encrypted **only for you, maybe additionally for the course administrator, but no one else** and **encrypt your response**

message.

When encrypting, it is very common to **add yourself to the recipients list** along with the intended recipients. This will ensure that you can still read your sent e-mails after they have been stored encrypted in your sent folder. In this lab you have to do this to get your reply accepted by the automatic checking system (also for the task in the next section). Again only you and the course administrator should be able to read the reply, no one else.

Think about...

- What command(s) did you use to decrypt an encrypted message?
- How did you check that no one else was able to read the message?
- Which key did you use to make the reply confidential?

Milestone
Make sure you have received a confirmation e-mail stating that your encrypted response was correct. □

3.3 Message signing and encryption

The most common use case of GPG is to provide both message integrity and confidentiality. You have received an e-mail from `dasak-gpg-both@sys.kth.se` contains signed and encrypted data. Now, send a single email to the address `gpg-both@dasak-vm-lab-server.eecs.kth.se` with only those strings that were properly signed and encrypted **only for you, maybe additionally for the course administrator, but no one else** and **sign and encrypt your response**.

Think about...

- What command(s) did you use to verify a signed and encrypted message?
- What does it mean, if a message is correctly signed and encrypted?
- Which key(s) did you use to secure the reply?

Milestone
Make sure you have received a confirmation e-mail stating that your signed and encrypted response was correct. □

If you received all three e-mails stating that your responses were correct, you are done with the lab.

4 Getting Real

We wanted you to limit the lifetime of your key because the signatures on it do not express real trust statements but are just part of the lab exercise. Therefore you should not use the keypair you created in the lab for real communication outside the lab. If you want to continue to use e-mail encryption you find some hints and useful information below.

4.1 Delete all Lab Keys and Create a New Keypair with Unlimited Lifetime

If you are completely done with the lab, a good idea might be to delete all lab keys from your keyring, including your secret key, so that you will not confuse them later with real keys.
Then you can repeat the process of creating a keypair in Section 2.1 but this time not limiting the lifetime of your key. You might want to add several identities (e-mail addresses) just as in the lab exercise. Note, however, that all people getting the public key, can deduce that all these e-mail addresses belong to the same person – so for anonymous e-mail addresses you might want to create separate keypairs.
Also think about how to maintain a secure backup of your secret key, as in case you are loosing it (e. g., due to a device failure) you won't be able to read your encrypted e-mails anymore (both those you sent and received).

4.2 Key Distribution: via E-Mail or Keyservers

You can send your public key either directly to those contacts that might want to use it (in an e-mail inline as GPG block or attached as file – Thunderbird recognizes both as public keys and offers to import them when receiving such an e-mail), publish it on your personal homepage or upload it to one of the public keyservers which serve as public directories mapping from e-mail addresses to public keys. Most key managers offer a convenient way to upload the key to a pre-defined set of popular keyservers – note, however, that once uploaded, you can never remove the key again. For that reason, you might want to create and store a revocation certificate (that can invalidate your key on the keyserver once you lost the key, the passphrase or when the key got compromised).
Remember that only after verifying the fingerprint of the key, you and your contacts can be sure that you got the right keys. If you cannot meet in person, reading the fingerprint via phone when recognizing the voice of your contact is a pretty safe way, as a voice is much harder to fake than modifying e-mail communication for a man-in-the-middle attack. Note that it is important to actually compare the whole fingerprint. An attacker can easily create a fake key with a fingerprint that partially matches the original. In particular, it is not enough to compare key IDs, because they are formed by taking only the last 32 bits of the full 160 bits key fingerprint. On dedicated hardware it takes only a few seconds to create a key with such a lookalike ID, as demonstrated here: https://evil32.com/.

4.3 Signing Keys: the Web of Trust Privacy Issue

The Web of Trust – i. e., the concept of signing each others' keys and using these trust statements to check the key authenticity – has a major privacy drawback: when publishing the key signatures (e. g., by uploading the signed public key to keyservers or passing it on manually) they effectively describe your social relations to all involved contacts. The only way to avoid this is not to use this feature at all, that is to authenticate keys only via fingerprints and not to sign other keys (GPG offers a "local, non-exportable" signature option that prevents the signature from being published but avoids security warnings about unsigned keys of some e-mail clients). For some use cases, the Trust On First Use (TOFU) principle might be a suitable alternative, compromising security with usability. According to this principle, a public key is assumed to be correct without any further authentication when it is received for the first time. For all future interactions the user only checks that the same key was used. This is vulnerable to man-in-the-middle attacks but they will be detected as soon as a communication channel is used that is not under the adversary's control.

4.4 Usability Considerations

Using a front-end that integrates GPG support in an e-mail client (such as EnigMail for Thunderbird or GPGMail for Apple's Mail, see Section 1.1) makes it very easy to use GPG in everyday communication – you don't need to touch the command line any more. These front-ends usually come with convenient key managers or call those of the operating system which help with creating keys, setting trust levels, importing and exporting keys, searching and uploading keys from popular keyservers, etc. If you are used, however, to read your e-mails on different devices via webmail interfaces, GPG might be inconvenient to use. You can only decrypt messages sent to you on those devices where you have both an e-mail client that supports GPG and your secret key available. There are several projects out there trying to close that gap, such as browser add-ons (e. g., Mailvelope https://www.mailvelope.

com/ or WebPG https://webpg.org/, both available for Firefox or Chrome) or smartphone apps (e. g., the Android apps OpenKeychain https://www.openkeychain.org/ or APG https://github.com/thialfihar/apg that integrate among others with the K9 e-mail client). But they all come with usability constraints and security drawbacks, e. g., browser add-ons only work if you transferred your secret key to the device running the browser and carrying your secret key on a phone might be less secure.

4.5 Security Considerations

Keep in mind that GPG does not have forward secrecy. If your GPG secret key is compromised (e. g., the file and the passphrase are available to an attacker) and the attacker has copies of encrypted e-mails you have sent or received in the past, she can go back and decrypt them all. Furthermore, GPG does not provide you with deniability and authentication at the same time – if you sign the message, you cannot deny that you wrote it (without giving up your secret key) and if you do not sign the message it's authenticity cannot be checked by the receiver. You might want to have a look at Off-the-Record Messaging (OTR, https://otr.cypherpunks.ca/) for instant messaging protocols if you are interested in forward secrecy and deniability. If you regularly want to use e-mail encryption on devices that you do not own, have a look at the secure live operating system Tails: https://tails.boum.org/. It can be carried on and booted from a USB flash drive and comes with e-mail encryption support together with many other privacy tools like Tor and OTR enabled chat clients.

5 Frequently Asked Questions

Since GPG is not the friendliest tool and combined with the complexity of cryptography it can take some time to get used to it. A great start is the documentation for GPG, which can be found at the official homepage.

5.1 Some common problems

In this section we list some (common) problems previous years' students have had trouble with.

- **Source e-mail address domain**
  Every single e-mail you send **must** come from your e-mail account at KTH, that is, `your-username@kth.se`. No other source address will be accepted nor processed by the system.

- **E-mail answer**
  All e-mails must have the answers in the **body** of the e-mail as **plain** text.
  No other format will work (such as Word, OpenOffice, RTF, PS, PDF, DVI, HTML,...). It is quite common for e-mail programs (including web-based) to send HTML or RTF by default; therefore, you should deactivate this functionality because the system does not recognize nor decode attachments, PGP/MIME, SMIME,... which means that in case of using a front-end you should make sure that it can do traditional in-lining. If you do not see something such as `----BEGIN GPG <TYPE>----` at the beginning of the e-mail's body you should verify how you are sending encrypted e-mails.

- **Extra output**
  It is common not to ask GPG for all information and few front-ends print all information you need. Using the option `-vv` (very verbose) may be very helpful as it prints extended information about what is going on.

5.2 Error messages

The following is a list of the common error messages you might get from the system and accompanied with a brief description on how to solve the cause related to these errors.

- **Could not import user id's. Typically caused by UIDs created in the future (wrong system time)**
  The system parsed your key correctly but failed to import your UIDs. This is most likely due to the creation date of the UID being in the future or too far into the past. Check your system time and generate a new key.

- **Did not find any keys to import/verifying signature failed/decrypting message failed/decryption and verification failed**
  The system tried to verify or decrypt the message, but GPG returned a non-zero return code and no data.
  This is commonly caused by formatting errors, such as HTML mail, the key in attachment, multiple GPG blocks and similar. Make sure you only include a single, properly armored, block and it is in the body e-mail.

- **Did not see any signatures not made by course key**
  Did you really get another student to sign your key? Did you import that signature?

- **Found # User-ID:s in the file. The initial submission should be with one User-ID. Generate a new key and try again**
  Your initial submission should not contain multiple UIDs. Follow the lab instruction and submit a key with a single UID first.

- **Found more than one key in your submission**
  Typically caused by exporting multiple keys before sending the mail to the lab system.
  By specifying which key you want to export you avoid this error. You can also try importing the key you have exported to make sure it contains what you think it contains using `gpg --import -nvv`.

- **Incorrect. I read your answer as:**
  This means the lab system correctly verified/decrypted the gpg block in your submission and failed when it verified your submission against the expected answers. The lab system is doing a string matching of the rows.
  Make sure there is no extra data in the output (compared to what you encrypted/signed). Then make sure you have chosen the correct set of blocks as your answer. When checking the encryption or both part, make sure that the messages you include in the answer were only encrypted to you, or to you and the course-key, but to no other key. So when decrypting, check for lines starting with ”gpg: encrypted/krypterad ...”, if you see a keyid there that you don’t know, this means that a third party can read the message. This is easy to miss, as it is a short line hidden in a lengthy output listing.

- **Key cannot be used for encryption**
  You have generated a sign-only key. This key can not be used for the lab and you will have to generate a key allowing both signing and encryption.

- **Secret key was imported, please generate a new keypair and be more careful next time.../The secret key for this key was already imported. Please generate a new keypair and be more careful next time**
  You sent your secret key to the lab system. As it is no longer secure you will have to generate a new key pair and start over.

- **Signature was invalid (status xxxxxxx)**
  Your signature was parsed correctly but was found invalid. Make sure you don’t modify the data in transport.

- **Submission was encrypted with # keys, should be encrypted to exactly two keys/Submission was only encrypted with a single key, should be encrypted to two keys**
  You have to make sure you use exactly two keys to encrypt the submission. That is your own key and the course key. Nobody else may read your submission.

- **Subkey expires in more than 3 months/Subkey never expires**
  Make sure you set a correct expiry date on the key when you generate it. It is in the lab instructions. Note: If that happens, you need to contact a lab assistant, so that they delete your incorrect key from the server. Then generate a new key with a correct expiry date and send it to the system again. It is unfortunately not possible to just change the expiry date of your key and send it again (you will get a ”Found one key, but I already had it... (this is fine)” message, but the system does not send you a signed version of this key then).

- **Submission was not signed**
  You did not sign the submission correctly before sending it to the lab system.

- **The User-ID found is “Foo Bar <foo@bar.com>”. It should have an e-mail address in the @kth.se domain**
  Make sure your primary UID is in the `@kth.se` domain. This is important as the lab system will only speak to `@kth.se` addresses.

- **Tried to import something but failed. This can be caused by broken data, revocation certificates or broken UIDs (such as UIDs created in the future)**
  Catch all for parsing errors when importing keys. If you get this the key has been parsed but GPG failed to import your keys.

- **Validation of your result failed. Key correctly signed by course key? I read your answer as:**
  Catch all for validating submissions.
  If you get this, GPG returned something but also had an error. The answer part is what gpg returned. Typically caused by sending data to the gpg-sign/crypt/both before you have your key correctly signed by the course key.

6 Command Line Basics – in case you are not familiar with the shell

6.1 Manual Pager

You get a reference manual for almost all command line programs by issuing `man PROGRAMNAME`, e. g., `man gpg`. It uses the same shortcuts as the file viewer `less`, which provides the following actions: Pressing Q will exit the viewer (quit). To scroll up or down half a page press Ctrl+U or Ctrl+D (or Space/Shift+Space for whole page scrolling). `/keyword` starts a search for "keyword" (only downwards from current position). Pressing N or Shift+N will get you to the next or previous occurence of the keyword. G or Shift+G will goto the beginning or end of the file.

6.2 Viewing a File

Use `ls` to list the files in the current directory (`cd ..` and `cd DIRECTORY` to change the directory). For viewing a file there are several possibilities, `less FILENAME` being one of the more convenient ones (see Section 6.1 above for navigation shortcuts).

6.3 Command History – Repeating Commands

If you type `history` you will see a list of recently executed commands. If you press the arrow up key repeatedly, you can browse through these commands. To search for a recently used command that contains a certain string, press Ctrl+R and type the search term – repeatedly pressing Ctrl+R will browse through the search results.

6.4 Clipboard – Copy+Paste

Note that Ctrl+C on the command line is used to kill the current program. Most terminals support Ctrl+Shift+C (after selecting lines with the mouse) and Ctrl+Shift+V as shortcuts for copy and paste to and from the system clipboard. Using the middle mouse button to paste selected text is often supported, too.

7 History

Ver	**Contribution**	**Author** (Affiliation)	**Contact**
1.0	First development	Gunnar Kreitz (CSC/KTH)	
2.0	Development continuation	Pehr Söderman (ICT/KTH)	pehrs@kth.se
3.0	Inclusion of experiences from 2009 course		
3.1	Adaptation for DASAK	Sonja Buchegger (CSC/KTH)	buc@kth.se
3.2	Adaptation and clarifications for CSC	Sonja Buchegger (CSC/KTH)	buc@kth.se
4.0	Adaptation and simplifications for HT2011	Guillermo Rodríguez Cano (CSC/KTH)	gurc@kth.se
4.1	Adaptation and clarifications for HT2012	Oleksandr Bodriagov (CSC/KTH)	obo@kth.se
4.2	New Introduction, Getting Real Section (HT2013)	Benjamin Greschbach (CSC/KTH)	bgre@kth.se
4.3	Adaption for HT2017	Andreas Lindner (CSC/KTH)	andili@kth.se
4.4	Adaption for HT2018	Andreas Lindner (EECS/KTH)	andili@kth.se
4.5	Adaption for HT2019	Md Sakib Nizam Khan (EECS/KTH)	msnkhan@kth.se
4.6	Adaption for HT2020	Md Sakib Nizam Khan (EECS/KTH)	msnkhan@kth.se
4.7	Adaption for HT2021	Md Sakib Nizam Khan (EECS/KTH)	msnkhan@kth.se
4.8	Adaption for HT2022	Anoud Alshnakat (EECS/KTH)	anoud@kth.se