



LAB

SIGNING GPG PUBLIC KEYS

Copyright © CC-BY-NC-ND (2022)
Dr. Matthew R. Kisow


VERSION 2

Table of Contents

OVERVIEW	5
CONFIDENTIALITY, INTEGRITY, AND AVAILABILITY (CIA)	5
LEARNING OBJECTIVES.....	5
READINGS	5
RELATED LABS	5
LAB ENVIRONMENT	6
READ AHEAD	6
HOW-TO USE THIS LAB GUIDE	6
THE CLI	6
FOOTNOTES	6
LAB TASKS.....	7
TASK 1: THE COMMAND-LINE INTERFACE (CLI).....	7
THE WINDOWS COMMAND PROMPT.....	7
🍏 THE MACOS TERMINAL.....	8
TASK 2: VALIDATING THE LAB ENVIRONMENT	9
PART I: GPG VERSION	9
PART II: CHECKING KEYS.....	10
TASK 3: ESTABLISHING A WEB-OF-TRUST (INSTRUCTORS PUBLIC KEY)	11
TASK 4: UPLOADING YOUR INSTRUCTOR SIGNED PUBLIC KEY	20
TASK 5: ESTABLISHING A WEB-OF-TRUST (PEERS PUBLIC KEY).....	22
TASK 6: UPLOADING YOUR PEER SIGNED PUBLIC KEY	30
SUBMISSION	32
RUBRIC	33
APPENDIX A	34
GPG COMMAND LINE REFERENCE	34
NAME.....	34
SYNOPSIS	34
DESCRIPTION.....	34
RETURN VALUE.....	34
WARNINGS.....	34
INTEROPERABILITY	35
COMMANDS.....	35
COMMANDS NOT SPECIFIC TO THE FUNCTION	35
--VERSION.....	35
COMMANDS TO SELECT THE TYPE OF OPERATION	36
--SIGN	36
--ENCRYPT.....	36
--DECRYPT.....	36

--LIST-KEYS	37
--LIST-SECRET-KEYS	38
--CHECK-SIGS	38
--EXPORT	39
--SEND-KEYS KEYIDS	39
--IMPORT	40
--RECEIVE-KEYS KEYIDS	40
--SEARCH-KEYS	40
HOW TO MANAGE YOUR KEYS	41
--EDIT-KEY	43
--SIGN-KEY NAME	46
OPTIONS	47
HOW TO CHANGE THE CONFIGURATION	47
--ASK-CERT-LEVEL	51
--KEYSERVER NAME	55
--EXPERT	58
KEY RELATED OPTIONS	58
--RECIPIENT NAME	58
INPUT AND OUTPUT	60
--ARMOR	60
--OUTPUT FILE	60
OPENPGP PROTOCOL SPECIFIC OPTIONS	64
COMPLIANCE OPTIONS	65
DOING THINGS ONE USUALLY DOESN'T WANT TO DO	66
--PINENTRY-MODE MODE	70
--LIST-SIGS	71
DEPRECATED OPTIONS	74
EXAMPLES	74
HOW TO SPECIFY A USER ID	75
FILTER EXPRESSIONS	77
TRUST VALUES	78
FILES	79
BUGS	81
SEE ALSO	81

APPENDIX B 82

RECOVERY TASK 1: THE COMMAND-LINE INTERFACE (CLI)	82
THE WINDOWS COMMAND PROMPT	82
 THE MACOS TERMINAL	82
RECOVERY TASK 2: IMPORT YOUR KEYS	83
IMPORTING YOUR PUBLIC/PRIVATE KEY PAIR	83
IMPORTING YOUR INSTRUCTORS PUBLIC KEY	84
LISTING THE KEYS	85

Overview

Authentication is the process or method of determining, verifying, and/or validating whether a person is who they say they are. This lab will give you first-hand experience signing and publishing a peer's public key after determining their identity. This lab builds upon concepts learned in the previous lab (*Sending Encrypted Messages*), where you created your own public/private key pair. This lab will cover the following topics:

- Public-Key
- Private-Key
- Public-Key Infrastructure (PKI)
- Publishing a signed Public-Key to a key server
- Establishing a Web-of-Trust (WoT)
- Encrypting a message using an individual's Public-Key

Confidentiality, Integrity, and Availability (CIA)

This set of labs has been designed with the CIA triad in mind. These labs will teach you how PKI establishes **confidentiality**, **integrity**, and **availability**. In the *Sending Encrypted Messages* lab, you will establish your public and private key pair, then send and receive encrypted messages keeping those messages **confidential**. Once your public key is sent to the public key servers, you will establish **availability**; **your** keys will be available on multiple key servers, ensuring that peers can confidentially contact you. Finally, **Integrity** will be looked at in depth in the second and third labs, "*Signing GPG Public Keys*" and "*Creating and Verifying Digitally Signed Documents*" in these labs, you will establish a Web of Trust (WoT) by attesting a peer's public key and verifying and creating digitally signed documents to ensure they have not been tampered with.

Learning Objectives

Students should be able to gain a better understanding of how the Public Key Infrastructure works and how it can be appropriately implemented to ensure the tenants of the CIA triad.

Readings

Additional detailed information about PKI and digital signatures can be found in the following:

- Chapter 16 – Summarizing the Basics of Cryptographic Concepts.
- Chapter 25 – Implementing Public Key Infrastructure.
- Supplemental materials provided by your instructor.

Related Labs


- Sending Encrypted Messages

Lab Environment

This lab has been tested using the environment listed below. Any variation outside this environment may cause certain aspects of this lab not to perform as indicated or desired.

- A Windows laptop or desktop computer with **GPG4Win** installed.
- A macOS laptop or desktop computer with **GPG Suite** installed.
- An e-mail account, your student-issued e-mail account, is preferred¹.
- An Internet connection.

Read Ahead

You are required to submit a lab report for this lab. Look for this icon  you will need to take screenshots throughout each of the tasks in section two for submission in your lab report.

How-To Use This Lab Guide

This guide is broken down into tasks. Each task corresponds to a step in how to properly sign a peer's public key and then publish your signed public key.

The CLI

The CLI will be shown with the commands you should type written in **yellow bold and italic** text, with all informational text written shown **red**.

The screens shown throughout this lab are simulated; however, your input and output will look similar.

Footnotes

Footnotes highlight or bring context to instructions found in the lab.

¹ If you require assistance with accessing or setting up your student e-mail account, please reach out to the college help desk.

Lab Tasks

The following tasks and corresponding screenshots were written using the Microsoft Windows operating system and GPG4Win. If you are using macOS, differences will be noted using the 🍏 icon.

Deviation from the lab environment listed above is not recommended and may require outside research to complete this lab's tasks.

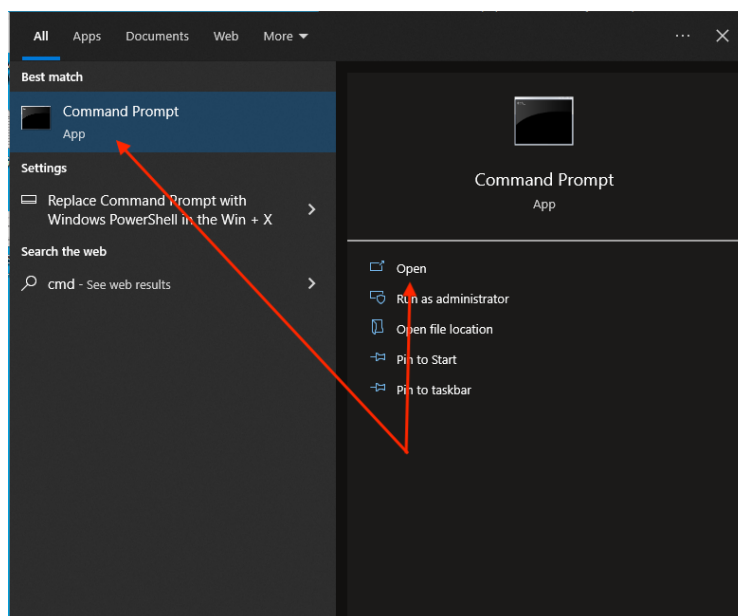
*A previous lab had you create a folder **student@example.com-keys** that held a backup copy of the revocation certificate (`revoke.asc`), your public key file (`public-key.asc`), and your private key file (`private-key.asc`). If the contents of this folder or the key ring have been removed, you will need to follow the steps outlined in **Appendix B** to recreate the folder and restore the files listed above from the backup location you were asked to create in the previous lab.*

Task 1: The Command-Line Interface (CLI)

Like the previous lab, this entire lab will be done in the CLI. This is done for two (2) reasons; first, there may not always be a Graphical User Interface (GUI) available when using these tools, and second, there are huge differences between the Windows, macOS, and Linux GUI versions of GPG, however, the command-line interface remains consistent across these platforms.

The Windows Command Prompt

1. From “Windows Search,” located next to the “Start” menu, type “cmd.”
2. Click the “Open” icon from the “Command Prompt” menu.



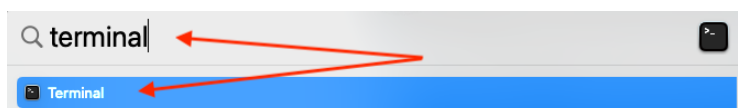
3. When the Command Prompt opens, you will be at the **C: \Users \username>** prompt.
4. Change into your **student@example.com-keys** folder by typing “cd student@example.com-keys,” and your command prompt will change to your directory name.

```
C:\Users\username> cd student@example.com-keys
C:\Users\username\student@example.com-keys>
```

NOTE: In these examples, ensure that you replace **student@example.com** with your e-mail address. If this folder does not exist, please refer to **Appendix B**.

🍏 The macOS Terminal

1. From “Spotlight Search,” located in the macOS menu bar, type “terminal”.
2. Click the “Terminal” icon.



3. When the terminal opens, you will be at the **computername:~ username\$** prompt.²
4. Change into your **student@example.com-keys** folder by typing “cd student@example.com-keys,” and your command prompt will change to your directory name.

```
computername:~ username$ cd student@example.com-keys
computername:student@example.com username$
```

NOTE: In these examples, ensure that you replace **student@example.com** with your e-mail address. If this folder does not exist, please refer to **Appendix B**.

² The command prompt may look different depending on the type of command-line shell your macOS is using. If you are using the zsh shell, your command prompt will look like this: **username@computername ~ %**

Task 2: Validating the Lab Environment

You need to validate the lab environment first by ensuring you have GPG installed and that it is a relevant version suitable for completing this lab's tasks and second by ensuring that our public/private key pair is available.

Part I: GPG Version

Check if GPG is installed on your laptop or desktop computer using the command `gpg --version`. This command should return a version for **GnuPG** that is higher than or equal to version 2.2.x, with a **libcrypt** version higher than or equal to 1.8.x.

1. Using the **gpg** command, you will list all keys in your key chain.

```
gpg --version
```

Use the following options with the **gpg** command to list the public and private keys in our key chain.

--version

This atom allows us to show the current version of GPG and the encryption library that is installed.

```
gpg --version
```

```
gpg (GnuPG/MacGPG2) 2.2.34
libgcrypt 1.8.9
Copyright (C) 2022 g10 Code GmbH
License GNU GPL-3.0-or-later <https://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Home: /Users/username/.gnupg
Supported algorithms:
Pubkey: RSA, ELG, DSA, ECDH, ECDSA, EDDSA
Cipher: IDEA, 3DES, CAST5, BLOWFISH, AES, AES192, AES256, TWOFISH,
        CAMELLIA128, CAMELLIA192, CAMELLIA256
Hash: SHA1, RIPEMD160, SHA256, SHA384, SHA512, SHA224
Compression: Uncompressed, ZIP, ZLIB, BZIP2
```

NOTE: If the `gpg --version` command does not return a value or an error message is displayed, please reinstall **GnuPG**.

Part II: Checking Keys

You will ensure that your public/private key pair from the ***Sending Encrypted Messages*** lab is still available in your gpg key chain.

- Using the **gpg** command, you will list all public and private keys in your key chain.

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

Use the following options with the **gpg** command to list the public and private keys in your key chain.

--list-keys

This atom allows you to list ALL public keys in your key chain.

--list-secret-keys

This atom allows you to list ALL private keys in your key chain.

```
gpg --list-keys
/Users/labstudent/.gnupg/pubring.kbx
-----
pub   rsa4096 2022-09-27 [SC]
      C56EA6745BF05DF4313A29DD3B8F1D139D70141E
uid   [ultimate] Lab Student <student@example.com>
sub   rsa4096 2022-09-27 [E]

pub   rsa4096 2022-01-30 [SC]
      38752E510AEFF60B2CA4A5E7007A2EFE19C015A2
uid   [ unknown] Matthew R. Kisow <mkisow@ccac.edu>
sub   rsa4096 2022-01-30 [E]

gpg --list-secret-keys
/Users/labstudent/.gnupg/pubring.kbx
-----
pub   rsa4096 2022-09-27 [SC]
      C56EA6745BF05DF4313A29DD3B8F1D139D70141E
uid   [ultimate] Lab Student <student@example.com>
sub   rsa4096 2022-09-27 [E]
```

*To ensure that your public/private key pair already exists, scrutinize the output from these two commands. If your keychain does not contain either your public or private key, follow the instructions located in **Appendix B** to recover them.*

Task 3: Establishing a Web-of-Trust (Instructors Public Key)

In a decentralized trust model, a Web-of-Trust (WoT) is a concept used to establish **authenticity** between a public key and the public key owner. In this section, you will establish a WoT by signing, exporting, and sending your instructor's signed public key to them for publishing; then, you will do the same with a peer's public key.

1. Using the **gpg** command, you will again list all the keys in your public key chain.

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

Use the following option with the **gpg** command to list the public keys in your key chain.

```
--list-keys
```

```
--list-keys
```

This atom allows you to list ALL public keys in your key chain.

```
gpg --list-keys
/Users/labstudent/.gnupg/pubring.kbx
-----
pub  rsa4096 2022-09-27 [SC]
    C56EA6745BF05DF4313A29DD3B8F1D139D70141E
uid  [ultimate] Lab Student <student@example.com>
sub  rsa4096 2022-09-27 [E]

pub  rsa4096 2022-01-30 [SC]
    38752E510AEFF60B2CA4A5E7007A2EFE19C015A2
uid  [ unknown] Matthew R. Kisow <mkisow@ccac.edu>
sub  rsa4096 2022-01-30 [E]
```

NOTE: The selections shown in red are the fingerprints of the public keys in your key chain. You will use these fingerprints in subsequent steps to identify the different keys in your key chain.

2. Using the **gpg** command, identify your instructor's key, then list all of the signatures associated with that key using the fingerprint shown in the previous command. The signatures will be shown as RSA IDs.

```
gpg --list-sigs <fingerprint>
```

Use the following option with the **gpg** command to list all signatures with your instructor's public key.

```
--list-sigs <fingerprint>
```

This atom allows us to list the signatures associated with the fingerprint.

```
gpg --list-sigs 38752E510AEFF60B2CA4A5E7007A2EFE19C015A2
pub  rsa4096 2022-01-30 [SC]
    38752E510AEFF60B2CA4A5E7007A2EFE19C015A2
uid  [ unknown] Matthew R. Kisow <mkisow@ccac.edu>
sig 3      007A2EFE19C015A2 2022-01-30  Matthew R. Kisow <mkisow@ccac.edu>
sub  rsa4096 2022-01-30 [E]
sig      007A2EFE19C015A2 2022-01-30  Matthew R. Kisow <mkisow@ccac.edu>
```

The selections in red are the RSA IDs associated with the public key you scrutinize. The RSA ID is the last 16-digits of a key's fingerprint. You will use these RSA IDs in subsequent steps to identify your instructor's and peer's public keys.

NOTE: In a WoT, a public key may be signed by multiple people. Those signatures will be identified by their public key RSA ID.

- Using the **gpg** command, you will check the signatures of your instructor's public key using the RSA ID shown in the previous command.

```
gpg --check-sigs <RSA ID>
```

Use the following option with the **gpg** command to validate all of the signatures associated with a public key.

[--check-sigs](#)

Validate the signatures associated with a public key.

```
gpg --check-sigs 007A2EFE19C015A2

pub  rsa4096 2022-01-30 [SC]
     38752E510AEFF60B2CA4A5E7007A2EFE19C015A2
uid  [unknown] Matthew R. Kisow <mkisow@ccac.edu>
sig!3      007A2EFE19C015A2 2022-01-30  Matthew R. Kisow <mkisow@ccac.edu>
sub  rsa4096 2022-01-30 [E]
sig!      007A2EFE19C015A2 2022-01-30  Matthew R. Kisow mkisow@ccac.edu

gpg: 2 good signatures
```

4. Using the **gpg** command, you will now sign and certify your instructor's public key.

```
gpg --ask-cert-level --sign-key <RSA ID>
```

Use the following two options with the **gpg** command when signing a peer's public key.

[--ask-cert-level](#)

This atom allows us to list and scrutinize ALL of the signatures of a public key.

[--sign-key](#) <RSA ID>

This atom allows us to sign the public key with your private key using the RSA ID of the key you will sign.

```
gpg --ask-cert-level --sign-key 007A2EFE19C015A2

pub  rsa4096/007A2EFE19C015A2
     created: 2022-01-30  expires: never      usage: SC
     trust: unknown      validity: unknown
sub  rsa4096/007A2EFE19C015A2
     created: 2022-01-30  expires: never      usage: E
[ unknown] (1). Matthew R. Kisow <mkisow@ccac.edu>

gpg: using "4986F621BF0071C3DB5B6F81183F002AD126B3D3" as default secret key for
signing

pub  rsa4096/007A2EFE19C015A2
     created: 2022-01-30  expires: never      usage: SC
     trust: unknown      validity: unknown
Primary key fingerprint: 3875 2E51 0AEF F60B 2CA4  A5E7 007A 2EFE 19C0 15A2

    Matthew R. Kisow <mkisow@ccac.edu>

How carefully have you verified the key you are about to sign actually belongs
to the person named above?  If you don't know what to answer, enter "0".

(0) I will not answer. (default)
(1) I have not checked at all.
(2) I have done casual checking.
(3) I have done very careful checking.

Your selection? (enter '?' for more information): 3
Are you sure that you want to sign this key with your
key "Lab Student <student@example.com> (183F002AD126B3D3)"

I have checked this key very carefully.

Really sign? (y/N) y
```

NOTE: When signing someone's public key, best practice dictates that you **ONLY** sign it if you know that person and can identify a relationship between them and their public key.

5.  Using the **gpg** command, you will *again* list the signatures of your instructor's public key.

This time take notice that the RSA ID of your public key is now listed because you signed it.

```
gpg --list-sigs <RSA ID>
```

We use the following option with the **gpg** command to list all of the signatures associated with a public key.

[--list-sigs](#)

This atom will list ALL of the signatures of a public key.

```
gpg --list-sigs 007A2EFE19C015A2

pub  rsa4096 2022-03-26 [SC]
    38752E510AEFF60B2CA4A5E7007A2EFE19C015A2
uid  [ full ] Matthew R. Kisow <mkisow@ccac.edu>
sig! 007A2EFE19C015A2 2022-06-12 Matthew R. Kisow <mkisow@ccac.edu>
sig!3 3B8F1D139D70141E 2022-09-27 Lab Student <student@example.com>
sub  rsa4096 2022-06-12 [E]
sig! 007A2EFE19C015A2 2022-06-12 Matthew R. Kisow <mkisow@ccac.edu>

gpg: 3 good signatures
```

6. Using the **gpg** command, you will export, sign, and encrypt your instructor's signed public key.

```
gpg --armor --output <signed public key>.asc --export <fingerprint>
gpg -a -o <encrypted filename>.gpg -s -e -r <email address> <signed public key>.asc
```

Use the following options with the **gpg** command to export, then sign and encrypt the exported public key.

NOTE: In this step, we will use both the long and short atoms of the **gpg** command to practice using both.

--armor

-a

This atom instructs gpg to export the signed public key in a uuencoded ASCII-armored format.

--output <signed public key>

-o <signed public key>

This atom instructs gpg to save the signed public key with the filename specified.

--export <e-mail address>

This atom instructs gpg to export the signed public key for the e-mail address specified.

--sign

-s

This atom instructs gpg to apply a digital signature to the signed public key to ensure it has not been tampered with in transport.

--encrypt

-e

This atom instructs gpg to encrypt the signed public key with the recipients public key.

--recipient <email address>

-r <email address>

This atom instructs gpg to encrypt the signed public key with the public key of the intended recipient.

```
gpg --armor --output mkisow_at_ccac_edu.asc --export mkisow@ccac.edu
gpg -a -o mkisow_at_ccac_edu.asc.gpg -s -e -r mkisow@ccac.edu mkisow_at_ccac_edu.asc
```

At this point, you could easily upload your signature for this public key back to the key server from where you imported it; however, best practice and netiquette dictate differently.

To ensure that this public key is indeed your instructor's public key, you need to export this signed public key, encrypt it with your instructor's public key, sign it with your private key then, and e-Mail it to your instructor so they can upload it.

This practice not only ensures that they have the private keys to decrypt your signature to their public key, but it also satisfies netiquette by offering them the opportunity to accept and publish this signed public key.

7. 📷 Attach the ***mkisow_at_ccac_edu.asc.gpg*** file you created in the previous step to an email and send it to your instructor.

*After your instructor has notified you that your signature on their public key has been accepted and uploaded, you may proceed to the next step. You will **NOT** be penalized if your instructor gets back to you late.*

8. 📷 Using the **gpg** command, download (refresh) your instructor's public key, then verify the signatures applied to your instructor's public key.

```
gpg --keyserver <key server> --recv-keys <RSA ID>
gpg --check-sigs <RSA ID>
```

Use the following options with the **gpg** command to download, then verify the signatures applied to your instructor's public key.

[--keyserver](#) <keyserver>

This atom instructs gpg to re-download your instructor's public key from the public key server specified.

[--receive-keys](#) <RSA ID>

This atom instructs gpg to download the key associated with the RSA ID specified.

[--check-sigs](#)


This atom instructs gpg to validate the signatures associated with a public key.

```
gpg --keyserver htps://keys.openpgp.org --receive-keys 007A2EFE19C015A2

gpg: key 007A2EFE19C015A2: "Matthew R. Kisow <mkisow@ccac.edu>" updated
gpg: Total number processed: 1
gpg:                updated: 1

gpg --check-sigs 007A2EFE19C015A2

pub  rsa4096 2022-01-30 [SC]
    38752E510AEFF60B2CA4A5E7007A2EFE19C015A2
uid          [ultimate] Matthew R. Kisow <mkisow@ccac.edu>
sig!3       007A2EFE19C015A2 2022-01-30  Matthew R. Kisow <mkisow@ccac.edu>
sig 3       3B8F1D139D70141E 2022-09-27  Lab Student <student@example.com>
sub  rsa4096 2022-01-30 [E]
sig!       007A2EFE19C015A2 2022-01-30  Matthew R. Kisow <mkisow@ccac.edu>
```

9.  From the command prompt, set a trust level³ for your instructor's public key by typing:

```
gpg --edit-key <RSA ID>
```

Use the following option with the **gpg** command to edit the trust level of your instructor's or peer's public key.

`--edit-key` <RSA ID>

This atom instructs gpg to edit the public key specified by the RSA ID.

```
gpg --edit-key 007A2EFE19C015A2

gpg (GnuPG/MacGPG2) 2.2.34; Copyright (C) 2022 g10 Code GmbH
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

sec  rsa4096/007A2EFE19C015A2
    created: 2022-01-30  expires: never           usage: SC
    trust: unknown      validity: full
sub  rsa4096/110AF8C0A1276394
    created: 2022-01-30  expires: never           usage: E
[ unknown] (1). Matthew R. Kisow <mkisow@ccac.edu>

gpg> trust
sec  rsa4096/007A2EFE19C015A2
    created: 2022-01-30  expires: never           usage: SC
    trust: unknown      validity: full
sub  rsa4096/110AF8C0A1276394
    created: 2022-01-30  expires: never           usage: E
[ unknown] (1). Matthew R. Kisow <mkisow@ccac.edu>

Please decide how far you trust this user to correctly verify other users' keys
(by looking at passports, checking fingerprints from different sources, etc.)

 1 = I don't know or won't say
 2 = I do NOT trust
 3 = I trust marginally
 4 = I trust fully
 5 = I trust ultimately
 m = back to the main menu

Your decision? 4

sec  rsa4096/007A2EFE19C015A2
    created: 2022-01-30  expires: never           usage: SC
    trust: full          validity: full
sub  rsa4096/110AF8C0A1276394
    created: 2022-01-30  expires: never           usage: E
[ unknown] (1). Matthew R. Kisow <mkisow@ccac.edu>
Please note that the shown key validity is not necessarily correct
unless you restart the program.

gpg> quit
```


You are under no obligation to set your instructor's public key to full trust. Set this trust level according to your comfort level. You will not be penalized!

³The different trust levels and their meanings are defined in the GPG manual page located in [Appendix A](#). It is best practice to **ONLY** set your personal public/private key pair to ultimate trust. All other keys should be set according to how well you know the individual.

Task 4: Uploading Your Instructor Signed Public Key

Once you receive your signed public key from your instructor, you will decrypt, verify and publish it to two key servers, *keys.openpgp.org* and *pgp.mit.edu*.

1. Copy the *student_at_example_com.asc.gpg* file from your email to your *student@example.com-keys* folder.

2.  Using the **gpg** command, decrypt your signed public key.

```
gpg --output <signed public key>.asc --decrypt <encrypted message>.gpg
```

Use the following options with the **gpg** command to decrypt your signed public key.


--output <signed public key>

This atom instructs gpg to save the signed public key with the filename specified.

--decrypt

This atom instructs gpg to decrypt the file encrypted with your public key.

```
gpg --output student_at_example_com.asc --decrypt student_at_example_com.asc.gpg
```

3.  Using the **gpg** command, import the public key that your instructor or peer signed and sent you, then verify the signature was imported by listing the signatures.

```
gpg --import <signed public key>
gpg --check-sigs <RSA ID>
```

Use the following options with the **gpg** command to download, then verify the signatures applied to your instructor's public key.

--import <signed public key>

This atom instructs gpg to import the public key signed by your instructor.

--check-sigs <RSA ID>


This atom instructs gpg to verify that your instructors signature has been applied to your public key.

```
gpg --import student_at_example_com.asc

gpg: key 3B8F1D139D70141E: "Lab Student <student@example.com>" updated
gpg: Total number processed: 1
gpg:                updated: 1

gpg --check-sigs 3B8F1D139D70141E

pub   rsa4096 2022-09-27 [SC]
      C56EA6745BF05DF4313A29DD3B8F1D139D70141E
uid   [ultimate] Lab Student <student@example.com>
sig!3   3B8F1D139D70141E 2022-09-27  Lab Student <student@example.com>
sig 3    007A2EFE19C015A2 2022-01-30  Matthew R. Kisow <mkisow@ccac.edu>
sub   rsa4096 2022-09-27 [E]
sig!     3B8F1D139D70141E 2022-09-27  Lab Student <student@example.com>
```

4.  Using the **gpg** command, you will upload your public key to two key servers.

```
gpg --keyserver <keyserver> --send-key <fingerprint>
```

You use the following two options with the **gpg** command to upload your public key to a key server.

[--keyserver](#) <keyserver>

This atom instructs gpg to upload your public key public to the key server specified.

[--send-key](#) <fingerprint>


This atom instructs gpg to send the public key for the fingerprint specified.

```
gpg --keyserver hkps://keys.openpgp.org --send-key C56EA6745BF05DF4313A29DD3B8F1D139D70141E
gpg: sending key 3B8F1D139D70141E to hkps://keys.openpgp.org

gpg --keyserver hkps://pgp.mit.edu --send-key C56EA6745BF05DF4313A29DD3B8F1D139D70141E
gpg: sending key 3B8F1D139D70141E to hkps://pgp.mit.edu
```

Task 5: Establishing a Web-of-Trust (Peers Public Key)

Using the “Lab 2” discussion board, communicate with your peers. Get to know them and share the information necessary to create a WoT with at least one of your peers.

1. Once you have established a rapport with one of your peers, obtain their email address, search for and download their public key from one of the two key servers used in this lab.
2.  Using the **gpg** command, you will search for and import your peers's public key from the key server using their e-mail address.

```
gpg --keyserver <keyserver> --search <email address>
```

Use the following two options with the **gpg** command to search for and retrieve your instructor's public key from the key server.

--keyserver <keyserver>

This atom instructs gpg to search the keyserver specified for your instructor's public key.

--search <email address>

This atom instructs gpg to search the keyserver specified for the email address associated with your instructors public key and download it.

```
gpg --keyserver htps://keys.openpgp.org --search peer@example.com
gpg: data source: https://keys.openpgp.org:443
(1)  Lab Peer <peer@example.com>
      4096 bit RSA key 007A2C015A2EFE19, created: 2022-01-30
Keys 1-1 of 1 for "peer@example.com". Enter number(s), N)ext, or Q)uit > 1
gpg: key 007A2C015A2EFE19: public key " Lab Peer peer@example.com imported
gpg: Total number processed: 1
gpg: imported: 1
```

3. Using the **gpg** command, list all the keys in your public key chain.

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

Use the following option with the **gpg** command to list the public keys in your key chain.

[--list-keys](#)

This atom allows you to list ALL public keys in your key chain.

```
gpg --list-keys

/Users/labstudent/.gnupg/pubring.kbx
-----
pub   rsa4096 2022-09-27 [SC]
      C56EA6745BF05DF4313A29DD3B8F1D139D70141E
uid   [ultimate] Lab Student <student@example.com>
sub   rsa4096 2022-09-27 [E]

pub   rsa4096 2022-01-30 [SC]
      38752E510AEFF60B2CA4A5E7007A2EFE19C015A2
uid   [ full] Matthew R. Kisow <mkisow@ccac.edu>
sub   rsa4096 2022-01-30 [E]

pub   rsa4096 2022-09-26 [SC]
      3AEFF8760B2C52E50A41A5E7007A2C015A2EFE19
uid   [ unknown] Lab Peer <peer@example.com>
sub   rsa4096 2022-09-26 [E]
```

NOTE: The selections shown in red are the fingerprints of the public keys in your key chain. You will use these fingerprints in subsequent steps to identify the different keys in your key chain.

- Using the **gpg** command, identify your peer's key, then list all of the signatures associated with that key. The signatures will be shown as RSA IDs.

```
gpg --list-sigs <fingerprint>
```

Use the following option with the **gpg** command to list all signatures with your peer's public key.

```
--list-sigs <fingerprint>
```

This atom allows us to list the signatures associated with the fingerprint.

```
gpg --list-sigs 3AEFF8760B2C52E50A41A5E7007A2C015A2EFE19

pub  rsa4096 2022-09-26 [SC]
    3AEFF8760B2C52E50A41A5E7007A2C015A2EFE19
uid  [ unknown] Lab Peer <peer@example.com>
sig   007A2C015A2EFE19 2022-09-26 Lab Peer <peer@example.com>
sig   007A2EFE19C015A2 2022-06-12 Matthew R. Kisow <mkisow@ccac.edu>
sub   rsa4096 2022-09-26 [E]
sig   007A2C015A2EFE19 2022-09-26 Lab Peer <peer@example.com>
```

The selections in red are the RSA IDs associated with the public key you scrutinize. The RSA ID is the last 16-digits of a key's fingerprint. You will use these RSA IDs in subsequent steps to identify your peer's public keys.

NOTE: In a WoT, a public key may be signed by multiple people. Those signatures will be identified by their public key RSA ID.

- Using the **gpg** command, you will check the signatures of your peer's public key using the RSA ID shown in the previous command.

```
gpg --check-sigs <RSA ID>
```

Use the following option with the **gpg** command to validate all of the signatures associated with a public key.

```
--check-sigs
```

Validate the signatures associated with a public key.

```
gpg --check-sigs 007A2C015A2EFE19

pub  rsa4096 2022-09-26 [SC]
    3AEFF8760B2C52E50A41A5E7007A2C015A2EFE19
uid  [ unknown] Lab Peer <peer@example.com>
sig!  007A2C015A2EFE19 2022-09-26 Lab Peer <peer@example.com>
sig!  007A2EFE19C015A2 2022-06-12 Matthew R. Kisow <mkisow@ccac.edu>
sub   rsa4096 2022-09-26 [E]
sig!  007A2C015A2EFE19 2022-09-26 Lab Peer <peer@example.com>

gpg: 3 good signatures
```


6. Using the **gpg** command, you will now sign and certify your peer's public key.

```
gpg --ask-cert-level --sign-key <RSA ID>
```

Use the following two options with the **gpg** command when signing a peer's public key.

[--ask-cert-level](#)

This atom allows us to list and scrutinize ALL of the signatures of a public key.

[--sign-key](#) <RSA ID>

This atom allows us to sign the public key with your private key using the RSA ID of the key you will sign.

```
gpg --ask-cert-level --sign-key 007A2C015A2EFE19

pub  rsa4096/007A2C015A2EFE19
    created: 2022-09-26  expires: never       usage: SC
    trust: unknown      validity: unknown
sub  rsa4096/007A2EFE19C015A2
    created: 2022-09-26  expires: never       usage: E
[ unknown] (1). Lab Peer <peer@example.com>

gpg: using "4986F621BF0071C3DB5B6F81183F002AD126B3D3" as default secret key for
signing

pub  rsa4096/007A2C015A2EFE19
    created: 2022-09-26  expires: never       usage: SC
    trust: unknown      validity: unknown
Primary key fingerprint: 3AEF F876 0B2C 52E5 0A41 A5E7 007A 2C01 5A2E FE19

    Lab Peer <peer@example.com>

How carefully have you verified the key you are about to sign actually belongs
to the person named above?  If you don't know what to answer, enter "0".

(0) I will not answer. (default)
(1) I have not checked at all.
(2) I have done casual checking.
(3) I have done very careful checking.

Your selection? (enter '?' for more information): 3
Are you sure that you want to sign this key with your
key "Lab Student <student@example.com> (183F002AD126B3D3)"

I have checked this key very carefully.

Really sign? (y/N) y
```

NOTE: When signing someone's public key, best practice dictates that you **ONLY** sign it if you know that person and can identify a relationship between them and their public key.

7.  Using the **gpg** command, you will *again* list the signatures of your instructor's public key.

This time take notice that the RSA ID of your public key is now listed because you signed it.

```
gpg --list-sigs <RSA ID>
```

We use the following option with the **gpg** command to list all of the signatures associated with a public key.

[--list-sigs](#)

This atom will list ALL of the signatures of a public key.

```
gpg --check-sigs 007A2C015A2EFE19

pub  rsa4096 2022-09-26 [SC]
    3AEFF8760B2C52E50A41A5E7007A2C015A2EFE19
uid  [ full. ] Lab Peer <peer@example.com>
sig!          007A2C015A2EFE19 2022-09-26  Lab Peer <peer@example.com>
sig!          007A2EFE19C015A2 2022-06-12  Matthew R. Kisow <mkisow@ccac.edu>
sig!3        183F002AD126B3D3 2022-09-27  Lab Student <student@example.com>
sub  rsa4096 2022-09-26 [E]
sig!          007A2C015A2EFE19 2022-09-26  Lab Peer <peer@example.com>

gpg: 4 good signatures
```

8. Using the **gpg** command, you will export, sign, and encrypt your instructor's signed public key.

```
gpg --armor --output <signed public key>.asc --export <fingerprint>
gpg -a -o <encrypted filename>.gpg -s -e -r <email address> <signed public key>.asc
```

Use the following options with the **gpg** command to export, then sign and encrypt the exported public key.

NOTE: In this step, we will use both the long and short atoms of the **gpg** command to practice using both.

--armor

-a

This atom instructs gpg to export the signed public key in a uuencoded ASCII-armored format.

--output <signed public key>

-o <signed public key>

This atom instructs gpg to save the signed public key with the filename specified.

--export <e-mail address>

This atom instructs gpg to export the signed public key for the e-mail address specified.

--sign

-s

This atom instructs gpg to apply a digital signature to the signed public key to ensure it has not been tampered with in transport.

--encrypt

-e

This atom instructs gpg to encrypt the signed public key with the recipients public key.

--recipient <email address>

-r <email address>

This atom instructs gpg to encrypt the signed public key with the public key of the intended recipient.

```
gpg --armor --output peer_at_example_com.asc --export peer@example.com
gpg -a -o peer_at_example_com.asc.gpg -s -e -r peer@example.com peer_at_example_com.asc
```

At this point, you could easily upload your signature for this public key back to the key server from where you imported it; however, best practice and netiquette dictate differently.

To ensure that this public key is indeed your peer's public key, you need to export this signed public key, encrypt it with your peer's public key, sign it with your private key then, and e-Mail it to your instructor so they can upload it.

This practice not only ensures that they have the private keys to decrypt your signature to their public key, but it also satisfies netiquette by offering them the opportunity to accept and publish this signed public key.

9. 📷 Attach the **peer_at_example_com.asc.gpg** file you created in the previous step to an email and send it to your peer.

After your peer has notified you that your signature on their public key has been accepted and uploaded, you may proceed to the next step.

10. 📷 Using the **gpg** command, download (refresh) your peer's public key, then verify the signatures applied to your instructor's public key.

```
gpg --keyserver <key server> --recv-keys <RSA ID>
gpg --check-sigs <RSA ID>
```

Use the following options with the **gpg** command to download, then verify the signatures applied to your instructor's public key.

[--keyserver](#) <keyserver>

This atom instructs gpg to re-download your instructor's public key from the public key server specified.

[--receive-keys](#) <RSA ID>

This atom instructs gpg to download the key associated with the RSA ID specified.

[--check-sigs](#)

This atom instructs gpg to validate the signatures associated with a public key.

```
gpg --keyserver hkps://keys.openpgp.org --receive-keys 007A2C015A2EFE19

gpg: key 007A2C015A2EFE19: "Lab Peer <peer@example.com>" updated
gpg: Total number processed: 1
gpg:                updated: 1

gpg --check-sigs 007A2C015A2EFE19

pub   rsa4096 2022-09-26 [SC]
      3AEFF8760B2C52E50A41A5E7007A2C015A2EFE19
uid           [ full. ] Lab Peer <peer@example.com>
sig!         007A2C015A2EFE19 2022-09-26  Lab Peer <peer@example.com>
sig!         007A2EFE19C015A2 2022-06-12  Matthew R. Kisow <mkisow@ccac.edu>
sig!3        183F002AD126B3D3 2022-09-27  Lab Student <student@example.com>
sub   rsa4096 2022-09-26 [E]
sig!         007A2C015A2EFE19 2022-09-26  Lab Peer <peer@example.com>

gpg: 4 good signatures
```

11. 📷 From the command prompt, set a trust level⁴ for your peer's public key by typing:

```
gpg --edit-key <RSA ID>
```

Use the following option with the **gpg** command to edit the trust level of your instructor's or peer's public key.

`--edit-key` <RSA ID>

This atom instructs gpg to edit the public key specified by the RSA ID.

```
gpg --edit-key 007A2C015A2EFE19

gpg (GnuPG/MacGPG2) 2.2.34; Copyright (C) 2022 g10 Code GmbH
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

sec  rsa4096/007A2C015A2EFE19
    created: 2022-09-26  expires: never           usage: SC
    trust: unknown      validity: full
sub  rsa4096/110AF8C0A1276394
    created: 2022-09-26  expires: never           usage: E
[ unknown] (1). Lab Peer <peer@example.com>

gpg> trust
sec  rsa4096/007A2C015A2EFE19
    created: 2022-09-26  expires: never           usage: SC
    trust: unknown      validity: full
sub  rsa4096/110AF8C0A1276394
    created: 2022-09-26  expires: never           usage: E
[ unknown] (1). Lab Peer <peer@example.com>

Please decide how far you trust this user to correctly verify other users' keys
(by looking at passports, checking fingerprints from different sources, etc.)

  1 = I don't know or won't say
  2 = I do NOT trust
  3 = I trust marginally
  4 = I trust fully
  5 = I trust ultimately
  m = back to the main menu

Your decision? 4

sec  rsa4096/007A2C015A2EFE19
    created: 2022-09-26  expires: never           usage: SC
    trust: full          validity: full
sub  rsa4096/110AF8C0A1276394
    created: 2022-09-26  expires: never           usage: E
[ unknown] (1). Lab Peer <peer@example.com>
Please note that the shown key validity is not necessarily correct
unless you restart the program.

gpg> quit
```


You are under no obligation to set your peer's public key to full trust. Set this trust level according to your comfort level. You will not be penalized!

⁴The different trust levels and their meanings are defined in the GPG manual page located in [Appendix A](#). It is best practice to ONLY set your personal public/private key pair to ultimate trust. All other keys should be set according to how well you know the individual.

Task 6: Uploading Your Peer Signed Public Key

Once you receive your signed public key from your peer, you will decrypt, verify and publish it to two key servers, *keys.openpgp.org* and *pgp.mit.edu*.

4. Copy the *student_at_example_com.asc.gpg* file from your email to your *student@example.com-keys* folder.

5.  Using the **gpg** command, decrypt your signed public key.

```
gpg --output <signed public key>.asc --decrypt <encrypted message>.gpg
```

Use the following options with the **gpg** command to decrypt your signed public key.


[--output](#) <signed public key>

This atom instructs gpg to save the signed public key with the filename specified.

[--decrypt](#)

This atom instructs gpg to decrypt the file encrypted with your public key.

```
gpg --output student_at_example_com.asc --decrypt student_at_example_com.asc.gpg
```

6.  Using the **gpg** command, import the public key that your instructor or peer signed and sent you, then verify the signature was imported by listing the signatures.

```
gpg --import <signed public key>
gpg --check-sigs <RSA ID>
```

Use the following options with the **gpg** command to download, then verify the signatures applied to your instructor's public key.

[--import](#) <signed public key>

This atom instructs gpg to import the public key signed by your instructor.

[--check-sigs](#) <RSA ID>


This atom instructs gpg to verify that your instructors signature has been applied to your public key.

```
gpg --import student_at_example_com.asc

gpg: key 3B8F1D139D70141E: "Lab Student <student@example.com>" updated
gpg: Total number processed: 1
gpg:          updated: 1

gpg --check-sigs 007A2C015A2EFE19

pub   rsa4096 2022-09-26 [SC]
      3AEFF8760B2C52E50A41A5E7007A2C015A2EFE19
uid   [ full. ] Lab Peer <peer@example.com>
sig!   007A2C015A2EFE19 2022-09-26  Lab Peer <peer@example.com>
sig!   007A2EFE19C015A2 2022-06-12  Matthew R. Kisow <mkisow@ccac.edu>
sig!3   183F002AD126B3D3 2022-09-27  Lab Student <student@example.com>
sub    rsa4096 2022-09-26 [E]
sig!   007A2C015A2EFE19 2022-09-26  Lab Peer <peer@example.com>
```

5.  Using the **gpg** command, you will upload your public key to two key servers.

```
gpg --keyserver <keyserver> --send-key <fingerprint>
```

You use the following two options with the **gpg** command to upload your public key to a key server.

[--keyserver](#) <keyserver>

This atom instructs gpg to upload your public key public to the key server specified.

[--send-key](#) <fingerprint>

This atom instructs gpg to send the public key for the fingerprint specified.


```
gpg --keyserver hkps://keys.openpgp.org --send-key C56EA6745BF05DF4313A29DD3B8F1D139D70141E
gpg: sending key 3B8F1D139D70141E to hkps://keys.openpgp.org

gpg --keyserver hkps://pgp.mit.edu --send-key C56EA6745BF05DF4313A29DD3B8F1D139D70141E
gpg: sending key 3B8F1D139D70141E to hkps://pgp.mit.edu
```

Submission

You must submit a detailed lab report with screenshots describing what you have done and observed. You will also need to provide a narrative explaining your observations, particularly anything you found interesting or surprising.

At a minimum, your lab report must include the following:

- A cover page.
- A summary paragraph that describes your experience using PKI to create a WoT.
- A narrative of your observations notes any interesting or surprising findings.
- Screenshots (included in-line with your report) from throughout the lab as indicated by the  icon.
- A copy of the decrypted e-mail that your instructor sent you.

This report should be written in Microsoft Word or its equivalent. The title page should include the name of the lab, your name, class number and section, and the date.

Using the **gpg** command, you will encrypt the lab report with your instructor's public key and email it to them.

```
gpg --armor --output <encrypted filename>.gpg --encrypt --recipient <recipient email>  
<lab report>
```

```
gpg --armor --output "Lab Student - Lab 2 Report.gpg" --encrypt --recipient  
mkisow@ccac.edu "Lab Student - Lab Report.docx"
```

Send your encrypted lab report in an e-mail to your instructor and post an unencrypted version to the assignment in BlackBoard.

Rubric

The rubric or check sheet (below) shows what I am looking for in your lab report; if you fail to include any information, you will lose points on your submission.

X	POINTS	REQUIREMENT OR LEARNING OBJECTIVE
	5	Cover Page, (Name, Class-Section and Date).
	20	A summary paragraph that describes students' experience using PKI.
	20	A narrative of the students' observations and findings.
	15	Screenshots, taken throughout the lab and included in the report.
	10	Your signed document and digital signature file.
	5	The lab report was written in Microsoft Word or an equivalent format.
	5	The lab report title page has all requested information.
	10	There are no spelling or grammatical issues.
	10	The lab report was encrypted, sent to the instructor in an e-mail, and published to BlackBoard in an unencrypted format.

APPENDIX A

GPG Command Line Reference

NAME

gpg - OpenPGP encryption and signing tool

SYNOPSIS

gpg [--homedir dir] [--options file] [options] command [args]

DESCRIPTION

gpg is the OpenPGP part of the GNU Privacy Guard (GnuPG). It is a tool to provide digital encryption and signing services using the OpenPGP standard. gpg features complete key management and all the bells and whistles you would expect from a full OpenPGP implementation.

There are two main versions of GnuPG: GnuPG 1.x and GnuPG 2.x. GnuPG 2.x supports modern encryption algorithms and thus should be preferred over GnuPG 1.x. You only need to use GnuPG 1.x if your platform doesn't support GnuPG 2.x, or you need support for some features that GnuPG 2.x has deprecated, e.g., decrypting data created with PGP-2 keys.

If you are looking for version 1 of GnuPG, you may find that version installed under the name gpg1.

RETURN VALUE

The program returns 0 if there are no severe errors, 1 if at least a signature was bad, and other error codes for fatal errors.

Note: that signature verification requires exact knowledge of what has been signed and by whom it has been signed. Using only the return code is thus not an appropriate way to verify a signature by a script. Either make proper use of the status codes or use the gpgv tool which has been designed to make signature verification easy for scripts.

WARNINGS

Use a good password for your user account and make sure that all security issues are always fixed on your machine. Also employ diligent physical protection to your machine. Consider to use a good passphrase as a last resort protection to your secret key in the case your machine gets stolen. It is important that your secret key is never leaked. Using an easy to carry around token or smartcard with the secret key is often advisable.

If you are going to verify detached signatures, make sure that the program knows about it; either give both filenames on the command line or use '-' to specify STDIN.

For scripted or other unattended use of gpg make sure to use the machine-parseable interface and not the default interface which is intended for direct use by humans. The machine-parseable interface provides a stable and well documented API independent of the locale or future changes of gpg. To enable this interface use the options `--with-colons` and `--status-fd`. For certain operations the option `--command-fd` may come handy too. See this man page and the file 'DETAILS' for the specification of the interface.

Note: that the GnuPG "info" pages as well as the PDF version of the GnuPG manual features a chapter on unattended use of GnuPG. As an alternative the library GPGME can be used as a high-level abstraction on top of that interface.

INTEROPERABILITY

GnuPG tries to be a very flexible implementation of the OpenPGP standard. In particular, GnuPG implements many of the optional parts of the standard, such as the SHA-512 hash, and the ZLIB and BZIP2 compression algorithms. It is important to be aware that not all OpenPGP programs implement these optional algorithms and that by forcing their use via the `--cipher-algo`, `--digest-algo`, `--cert-digest-algo`, or `--compress-algo` options in GnuPG, it is possible to create a perfectly valid OpenPGP message, but one that cannot be read by the intended recipient.

There are dozens of variations of OpenPGP programs available, and each supports a slightly different subset of these optional algorithms. For example, until recently, no (unhacked) version of PGP supported the BLOWFISH cipher algorithm. A message using BLOWFISH simply could not be read by a PGP user. By default, GnuPG uses the standard OpenPGP preferences system that will always do the right thing and create messages that are usable by all recipients, regardless of which OpenPGP program they use. Only override this safe default if you really know what you are doing.

If you absolutely must override the safe default, or if the preferences on a given key are invalid for some reason, you are far better off using the `--pgp6`, `--pgp7`, or `--pgp8` options. These options are safe as they do not force any particular algorithms in violation of OpenPGP, but rather reduce the available algorithms to a "PGP-safe" list.

COMMANDS

Commands are not distinguished from options except for the fact that only one command is allowed. Generally speaking, irrelevant options are silently ignored, and may not be checked for correctness.

gpg may be run with no commands. In this case it will print a warning perform a reasonable action depending on the type of file it is given as input (an encrypted message is decrypted, a signature is verified, a file containing keys is listed, etc.).

If you run into any problems, please add the option `--verbose` to the invocation to see more diagnostics.

Commands Not Specific to the Function

--version

Print the program version and licensing information.

Note: that you cannot abbreviate this command.

--help**-h**

Print a usage message summarizing the most useful command-line options.

Note: that you cannot arbitrarily abbreviate this command (though you can use its short form `-h`).

--warranty

Print warranty information.

--dump-options

Print a list of all available options and commands.

Note: that you cannot abbreviate this command.

Commands to Select the Type of Operation

--sign

-s

Sign a message. This command may be combined with `--encrypt` (to sign and encrypt a message), `--symmetric` (to sign and symmetrically encrypt a message), or both `--encrypt` and `--symmetric` (to sign and encrypt a message that can be decrypted using a secret key or a passphrase). The signing key is chosen by default or can be set explicitly using the `--local-user` and `--default-key` options.

--clear-sign

--clearsign

Make a cleartext signature. The content in a cleartext signature is readable without any special software. OpenPGP software is only needed to verify the signature. cleartext signatures may modify end-of-line whitespace for platform independence and are not intended to be reversible. The signing key is chosen by default or can be set explicitly using the `--local-user` and `--default-key` options.

--detach-sign

-b

Make a detached signature.

--encrypt

-e

Encrypt data to one or more public keys. This command may be combined with `--sign` (to sign and encrypt a message), `--symmetric` (to encrypt a message that can be decrypted using a secret key or a passphrase), or `--sign` and `--symmetric` together (for a signed message that can be decrypted using a secret key or a passphrase). `--recipient` and related options specify which public keys to use for encryption.

--symmetric

-c

Encrypt with a symmetric cipher using a passphrase. The default symmetric cipher used is AES-128, but may be chosen with the `--cipher-algo` option. This command may be combined with `--sign` (for a signed and symmetrically encrypted message), `--encrypt` (for a message that may be decrypted via a secret key or a passphrase), or `--sign` and `--encrypt` together (for a signed message that may be decrypted via a secret key or a passphrase). `gpg` caches the passphrase used for symmetric encryption so that a decrypt operation may not require that the user needs to enter the passphrase. The option `--no-symkey-cache` can be used to disable this feature.

--store

Store only (make a simple literal data packet).

--decrypt

-d

Decrypt the file given on the command line (or STDIN if no file is specified) and write it to STDOUT (or the file specified with `--output`). If the decrypted file is signed, the signature is also verified. This command differs from the default operation, as it never writes to the filename which is included in the file and it rejects files that don't begin with an encrypted message.

--verify

Assume that the first argument is a signed file and verify it without generating any output. With no arguments, the signature packet is read from STDIN. If only one argument is given, the specified file is expected to include a complete signature.

With more than one argument, the first argument should specify a file with a detached signature and the remaining files should contain the signed data. To read the signed data from STDIN, use '-' as the second filename. For security reasons, a detached signature will not read the signed material from STDIN if not explicitly specified.

Note: If the option `--batch` is not used, `gpg` may assume that a single argument is a file with a detached signature, and it will try to find a matching data file by stripping certain suffixes. Using this historical feature to verify a detached signature is strongly discouraged; you should always specify the data file explicitly.

Note: When verifying a cleartext signature, `gpg` verifies only what makes up the cleartext signed data and not any extra data outside of the cleartext signature or the header lines directly following the dash marker line. The option `--output` may be used to write out the actual signed data, but there are other pitfalls with this format as well. It is suggested to avoid cleartext signatures in favor of detached signatures.

Note: Sometimes the use of the `gpgv` tool is easier than using the full-fledged `gpg` with this option. `gpgv` is designed to compare signed data against a list of trusted keys and returns with success only for a good signature. It has its own manual page.

--multifile

This modifies certain other commands to accept multiple files for processing on the command line or read from STDIN with each filename on a separate line. This allows for many files to be processed at once. `--multifile` may currently be used along with `--verify`, `--encrypt`, and `--decrypt`.

Note: that `--multifile --verify` may not be used with detached signatures.

--verify-files

Identical to `--multifile --verify`.

--encrypt-files

Identical to `--multifile --encrypt`.

--decrypt-files

Identical to `--multifile --decrypt`.

--list-keys**-k****--list-public-keys**

List the specified keys. If no keys are specified, then all keys from the configured public keyrings are listed.

Never use the output of this command in scripts or other programs. The output is intended only for humans and its format is likely to change. The `--with-colons` option emits the output in a stable, machine-parseable format, which is intended for use by scripts and other programs.

--list-secret-keys**-K**

List the specified secret keys. If no keys are specified, then all known secret keys are listed. A # after the initial tags sec or ssb means that the secret key or subkey is currently not usable. We also say that this key has been taken offline (for example, a primary key can be taken offline by exporting the key using the command `--export-secret-subkeys`). A > after these tags indicate that the key is stored on a smartcard. See also `--list-keys`.

--check-signatures**--check-sigs**

Same as `--list-keys`, but the key signatures are verified and listed too.

Note: that for performance reasons the revocation status of a signing key is not shown. This command has the same effect as using `--list-keys` with `--with-sig-check`.

The status of the verification is indicated by a flag directly following the "sig" tag (and thus before the flags described below. A "!" indicates that the signature has been successfully verified, a "-" denotes a bad signature and a "%" is used if an error occurred while checking the signature (e.g. a non-supported algorithm). Signatures where the public key is not available are not listed; to see their keyids the command `--list-sigs` can be used.

For each signature listed, there are several flags in between the signature status flag and keyid. These flags give additional information about each key signature. From left to right, they are the numbers 1-3 for certificate check level (see `--ask-cert-level`), "L" for a local or non-exportable signature (see `--lsign-key`), "R" for a nonRevocable signature (see the `--edit-key` command "nrsign"), "P" for a signature that contains a policy URL (see `--cert-policy-url`), "N" for a signature that contains a notation (see `--cert-notation`), "X" for an expired signature (see `--ask-cert-expire`), and the numbers 1-9 or "T" for 10 and above to indicate trust signature levels (see the `--edit-key` command "tsign").

--locate-keys**--locate-external-keys**

Locate the keys given as arguments. This command basically uses the same algorithm as used when locating keys for encryption and may thus be used to see what keys gpg might use. In particular external methods as defined by `--auto-key-locate` are used to locate a key if the arguments contain a valid mail addresses. Only public keys are listed.

The variant `--locate-external-keys` does not consider a locally existing key and can thus be used to force the refresh of a key via the defined external methods. If a fingerprint is given and the methods defined by `--auto-key-locate` define LDAP servers, the key is fetched from these resources; defined non-LDAP key servers are skipped.

--show-keys

This command takes OpenPGP keys as input and prints information about them in the same way the command `--list-keys` does for locally stored key. In addition the list options `show-unusable-uids`, `show-unusable-subkeys`, `show-notations` and `show-policy-urls` are also enabled. As usual for automated processing, this command should be combined with the option `--with-colons`.

--fingerprint

List all keys (or the specified ones) along with their fingerprints. This is the same output as `--list-keys` but with the additional output of a line with the fingerprint. May also be combined with `--check-signatures`. If this command is given twice, the fingerprints of all secondary keys are listed too. This command also forces pretty printing of fingerprints if the keyid format has been set to "none".

--list-packets

List only the sequence of packets. This command is only useful for debugging. When used with option `--verbose` the actual MPI values are dumped and not only their lengths.

Note: that the output of this command may change with new releases.

--edit-card**--card-edit**

Present a menu to work with a smartcard. The subcommand "help" provides an overview on available commands. For a detailed description, please see the Card HOWTO at <https://gnupg.org/documentation/howtos.html#GnuPG-cardHOWTO>.

--card-status

Show the content of the smart card.

--change-pin

Present a menu to allow changing the PIN of a smartcard. This functionality is also available as the subcommand "passwd" with the `--edit-card` command.

--delete-keys name

Remove key from the public keyring. In batch mode either `--yes` is required or the key must be specified by fingerprint. This is a safeguard against accidental deletion of multiple keys. If the exclamation mark syntax is used with the fingerprint of a subkey only that subkey is deleted; if the exclamation mark is used with the fingerprint of the primary key the entire public key is deleted.

--delete-secret-keys name

Remove key from the secret keyring. In batch mode the key must be specified by fingerprint. The option `--yes` can be used to advise gpg-agent not to request a confirmation. This extra pre-caution is done because gpg can't be sure that the secret key (as controlled by gpg-agent) is only used for the given OpenPGP public key. If the exclamation mark syntax is used with the fingerprint of a subkey only the secret part of that subkey is deleted; if the exclamation mark is used with the fingerprint of the primary key only the secret part of the primary key is deleted.

--delete-secret-and-public-key name

Same as `--delete-key`, but if a secret key exists, it will be removed first. In batch mode the key must be specified by fingerprint. The option `--yes` can be used to advise gpg-agent not to request a confirmation.

--export

Either export all keys from all keyrings (default keyring and those registered via option `--keyring`), or if at least one name is given, those of the given name. The exported keys are written to STDOUT or to the file given with option `--output`. Use together with `--armor` to mail those keys.

--send-keys keyIDs

Similar to `--export` but sends the keys to a keyserver. Fingerprints may be used instead of key IDs. Don't send your complete keyring to a keyserver --- select only those keys which are new or changed by you. If no keyIDs are given, gpg does nothing.

Take Care: Keyserver are by design write only systems and thus it is not possible to ever delete keys once they have been send to a keyserver.

--export-secret-keys**--export-secret-subkeys**

Same as `--export`, but exports the secret keys instead. The exported keys are written to STDOUT or to the file given with option `--output`. This command is often used along with the option `--armor` to allow for easy printing of the key for paper backup; however the external tool paperkey does a better job of creating backups on paper.

Note: that exporting a secret key can be a security risk if the exported keys are sent over an insecure channel.

The second form of the command has the special property to render the secret part of the primary key useless; this is a GNU extension to OpenPGP and other implementations cannot be expected to successfully import such a key. Its intended use is in generating a full key with an additional signing subkey on a dedicated machine. This command then exports the key without the primary key to the main machine.

GnuPG may ask you to enter the passphrase for the key. This is required, because the internal protection method of the secret key is different from the one specified by the OpenPGP protocol.

--export-ssh-key

This command is used to export a key in the OpenSSH public key format. It requires the specification of one key by the usual means and exports the latest valid subkey which has an authentication capability to STDOUT or to the file given with option `--output`. That output can directly be added to ssh's "authorized_key" file.

By specifying the key to export using a key ID or a fingerprint suffixed with an exclamation mark (!), a specific subkey or the primary key can be exported. This does not even require that the key has the authentication capability flag set.

--import**--fast-import**

Import/merge keys. This adds the given keys to the keyring. The fast version is currently just a synonym.

There are a few other options which control how this command works. Most notable here is the `--import-options merge-only` option which does not insert new keys but does only the merging of new signatures, user-IDs and subkeys.

--receive-keys keyIDs**--recv-keys keyIDs**

Import the keys with the given keyIDs from a keyserver.

--refresh-keys

Request updates from a keyserver for keys that already exist on the local keyring. This is useful for updating a key with the latest signatures, user IDs, etc. Calling this with no arguments will refresh the entire keyring.

--search-keys names

Search the keyserver for the given names. Multiple names given here will be joined together to create the search string for the keyserver.

Note: that keyservers search for names in a different and simpler way than gpg does. The best choice is to use a mail address. Due to data privacy reasons keyservers may even not even allow searching by user id or mail address and thus may only return results when being used with the `--recv-key` command to search by key fingerprint or keyid.

--fetch-keys URIs

Retrieve keys located at the specified URIs.

Note: that different installations of GnuPG may support different protocols (HTTP, FTP, LDAP, etc.). When using HTTPS the system provided root certificates are used by this command.

--update-trustdb

Do trust database maintenance. This command iterates over all keys and builds the Web of Trust. This is an interactive command because it may have to ask for the "ownertrust" values for keys. The user has to give an estimation of how far she trusts the owner of the displayed key to correctly certify (sign) other keys. GnuPG only asks for the ownertrust value if it has not yet been assigned to a key. Using the `--edit-key` menu, the assigned value can be changed at any time.

--check-trustdb

Do trust database maintenance without user interaction. From time to time the trust database must be updated so that expired keys or signatures and the resulting changes in the Web of Trust can be tracked. Normally, GnuPG will calculate when this is required and do it automatically unless `--no-auto-check-trustdb` is set. This command can be used to force a trust database check at any time. The processing is identical to that of `--update-trustdb` but it skips keys with a not yet defined "ownertrust".

For use with cron jobs, this command can be used together with `--batch` in which case the trust database check is done only if a check is needed. To force a run even in batch mode add the option `--yes`.

--export-ownertrust

Send the ownertrust values to STDOUT. This is useful for backup purposes as these values are the only ones which can't be re-created from a corrupted trustdb.

Example: `gpg --export-ownertrust > otrust.txt`

--import-ownertrust

Update the trustdb with the ownertrust values stored in files (or STDIN if not given); existing values will be overwritten. In case of a severely damaged trustdb and if you have a recent backup of the ownertrust values (e.g. in the file 'otrust.txt'), you may re-create the trustdb using these commands:

```
cd ~/.gnupg
rm trustdb.gpg
gpg --import-ownertrust < otrust.txt
```


--rebuild-keydb-caches

When updating from version 1.0.6 to 1.0.7 this command should be used to create signature caches in the keyring. It might be handy in other situations too.

--print-md algo**--print-mds**

Print message digest of algorithm algo for all given files or STDIN. With the second form (or a deprecated "*" for algo) digests for all available algorithms are printed.

--gen-random 0|1|2 count

Emit count random bytes of the given quality level 0, 1 or 2. If count is not given or zero, an endless sequence of random bytes will be emitted. If used with --armor the output will be base64 encoded. PLEASE, don't use this command unless you know what you are doing; it may remove precious entropy from the system!

--gen-prime mode bits

Use the source, Luke :-). The output format is subject to change with ant release.

--enarmor**--dearmor**

Pack or unpack an arbitrary input into/from an OpenPGP ASCII armor. This is a GnuPG extension to OpenPGP and in general not very useful.

--tofu-policy {auto|good|unknown|bad|ask} keys

Set the TOFU policy for all the bindings associated with the specified keys. For more information about the meaning of the policies, see: [trust-model-tofu]. The keys may be specified either by their fingerprint (preferred) or their keyid.

How to Manage your Keys

This section explains the main commands for key management.

--quick-generate-key user-id [algo [usage [expire]]]**--quick-gen-key**

This is a simple command to generate a standard key with one user id. In contrast to --generate-key the key is generated directly without the need to answer a bunch of prompts. Unless the option --yes is given, the key creation will be canceled if the given user id already exists in the keyring.

If invoked directly on the console without any special options an answer to a "Continue?" style confirmation prompt is required. In case the user id already exists in the keyring a second prompt to force the creation of the key will show up.

If algo or usage are given, only the primary key is created and no prompts are shown. To specify an expiration date but still create a primary and subkey use "default" or "future-default" for algo and "default" for usage. For a description of these optional arguments see the command --quick-add-key. The usage accepts also the value "cert" which can be used to create a certification only primary key; the default is to create certification and signing key.

The expire argument can be used to specify an expiration date for the key. Several formats are supported; commonly the ISO formats "YYYY-MM-DD" or "YYYYMMDDThhmmss" are used. To make the key expire in N seconds, N days, N weeks, N months, or N years use "seconds=N", "Nd", "Nw", "Nm", or "Ny" respectively. Not specifying a value, or using "-" results in a key expiring in a reasonable default interval. The values "never", "none" can be used for no expiration date.

If this command is used with --batch, --pinentry-mode has been set to loopback, and one of the passphrase options (--passphrase, --passphrase-fd, or --passphrase-file) is used, the supplied passphrase is used for the new key and the agent does not ask for it. To create a key without any protection --passphrase "" may be used.

To create an OpenPGP key from the keys available on the currently inserted smartcard, the special string "card" can be used for algo. If the card features an encryption and a signing key, gpg will figure them out and creates an OpenPGP key consisting of the usual primary key and one subkey. This works only with certain smartcards.

Note: that the interactive --full-gen-key command allows to do the same but with greater flexibility in the selection of the smartcard keys.

Note: that it is possible to create a primary key and a subkey using non-default algorithms by using "default" and changing the default parameters using the option --default-new-key-algo.

--quick-set-expire fpr expire [*|subfprs]

With two arguments given, directly set the expiration time of the primary key identified by fpr to expire. To remove the expiration time 0 can be used. With three arguments and the third given as an asterisk, the expiration time of all non-revoked and not yet expired subkeys are set to expire. With more than two arguments and a list of fingerprints given for subfprs, all non-revoked subkeys matching these fingerprints are set to expire.

--quick-add-key fpr [algo [usage [expire]]]

Directly add a subkey to the key identified by the fingerprint fpr. Without the optional arguments an encryption subkey is added. If any of the arguments are given a more specific subkey is added.

algo may be any of the supported algorithms or curve names given in the format as used by key listings. To use the default algorithm the string "default" or "-" can be used. Supported algorithms are "rsa", "dsa", "elg", "ed25519", "cv25519", and other ECC curves. For example the string "rsa" adds an RSA key with the default key length; a string "rsa4096" requests that the key length is 4096 bits. The string "future-default" is an alias for the algorithm which will likely be used as default algorithm in future versions of gpg. To list the supported ECC curves the command gpg --with-colons --list-config curve can be used.

Depending on the given algo the subkey may either be an encryption subkey or a signing subkey. If an algorithm is capable of signing and encryption and such a subkey is desired, a usage string must be given. This string is either "default" or "-" to keep the default or a comma delimited list (or space delimited list) of keywords: "sign" for a signing subkey, "auth" for an authentication subkey, and "encr" for an encryption subkey ("encrypt" can be used as alias for "encr"). The valid combinations depend on the algorithm.

The expire argument can be used to specify an expiration date for the key. Several formats are supported; commonly the ISO formats "YYYY-MM-DD" or "YYYYMMDDThhmmss" are used. To make the key expire in N seconds, N days, N weeks, N months, or N years use "seconds=N", "Nd", "Nw", "Nm", or "Ny" respectively. Not specifying a value, or using "-" results in a key expiring in a reasonable default interval. The values "never", "none" can be used for no expiration date.

--generate-key**--gen-key**

Generate a new key pair using the current default parameters. This is the standard command to create a new key. In addition to the key a revocation certificate is created and stored in the "openpgp-revocs.d" directory below the GnuPG home directory.

--full-generate-key**--full-gen-key**

Generate a new key pair with dialogs for all options. This is an extended version of --generate-key.

There is also a feature which allows you to create keys in batch mode. See the manual section "Unattended key generation" on how to use this.

--generate-revocation name**--gen-revoke name**

Generate a revocation certificate for the complete key. To only revoke a subkey or a key signature, use the --edit command.

This command merely creates the revocation certificate so that it can be used to revoke the key if that is ever needed. To actually revoke a key the created revocation certificate needs to be merged with the key to revoke. This is done by importing the revocation certificate using the --import command. Then the revoked key needs to be published, which is best done by sending the key to a keyserver (command --send-key) and by exporting (--export) it to a file which is then sent to frequent communication partners.

--generate-designated-revocation name**--desig-revoke name**

Generate a designated revocation certificate for a key. This allows a user (with the permission of the keyholder) to revoke someone else's key.

--edit-key

Present a menu which enables you to do most of the key management related tasks. It expects the specification of a key on the command line.

uid n

Toggle selection of user ID or photographic user ID with index n. Use * to select all and 0 to deselect all.

key n

Toggle selection of subkey with index n or key ID n. Use * to select all and 0 to deselect all.

sign

Make a signature on key of user name. If the key is not yet signed by the default user (or the users given with -u), the program displays the information of the key again, together with its fingerprint and asks whether it should be signed. This question is repeated for all users specified with -u.

lsign

Same as "sign" but the signature is marked as non-exportable and will therefore never be used by others. This may be used to make keys valid only in the local environment.

nrsign

Same as "sign" but the signature is marked as non-revocable and can therefore never be revoked.

tsign

Make a trust signature. This is a signature that combines the notions of certification (like a regular signature), and trust (like the "trust" command). It is generally only useful in distinct communities or groups. For more information please read the sections "Trust Signature" and "Regular Expression" in RFC-4880.

Note: that "l" (for local / non-exportable), "nr" (for non-revocable, and "t" (for trust) may be freely mixed and prefixed to "sign" to create a signature of any type desired.

If the option --only-sign-text-ids is specified, then any non-text based user ids (e.g., photo IDs) will not be selected for signing.

delsig

Delete a signature.

Note: that it is not possible to retract a signature, once it has been send to the public (i.e. to a keyserver). In that case you better use revsig.

revsig

Revoke a signature. For every signature which has been generated by one of the secret keys, GnuPG asks whether a revocation certificate should be generated.

check

Check the signatures on all selected user IDs. With the extra option selfsig only self-signatures are shown.

adduid

Create an additional user ID.

addphoto

Create a photographic user ID. This will prompt for a JPEG file that will be embedded into the user ID.

Note: that a very large JPEG will make for a very large key.

Note: that some programs will display your JPEG unchanged (GnuPG), and some programs will scale it to fit in a dialog box (PGP).

showphoto

Display the selected photographic user ID.

deluid

Delete a user ID or photographic user ID.

Note: that it is not possible to retract a user id, once it has been send to the public (i.e. to a keyserver). In that case you better use revuid.

revuid

Revoke a user ID or photographic user ID.

primary

Flag the current user id as the primary one, removes the primary user id flag from all other user ids and sets the timestamp of all affected self-signatures one second ahead.

Note: that setting a photo user ID as primary makes it primary over other photo user IDs, and setting a regular user ID as primary makes it primary over other regular user IDs.

keyserver

Set a preferred keyserver for the specified user ID(s). This allows other users to know where you prefer they get your key from. See --keyserver-options honor-keyserver-url for more on how this works. Setting a value of "none" removes an existing preferred keyserver.

notation

Set a name=value notation for the specified user ID(s). See --cert-notation for more on how this works. Setting a value of "none" removes all notations, setting a notation prefixed with a minus sign (-) removes that notation, and setting a notation name (without the =value) prefixed with a minus sign removes all notations with that name.

pref

List preferences from the selected user ID. This shows the actual preferences, without including any implied preferences.

showpref

More verbose preferences listing for the selected user ID. This shows the preferences in effect by including the implied preferences of 3DES (cipher), SHA-1 (digest), and Uncompressed (compression) if they are not already included in the preference list. In addition, the preferred keyserver and signature notations (if any) are shown.

setpref <string>

Set the list of user ID preferences to string for all (or just the selected) user IDs. Calling setpref with no arguments sets the preference list to the default (either built-in or set via --default-preference-list), and calling setpref with "none" as the argument sets an empty preference list. Use gpg --version to get a list of available algorithms.

Note: that while you can change the preferences on an attribute user ID (aka "photo ID"), GnuPG does not select keys via attribute user IDs so these preferences will not be used by GnuPG.

When setting preferences, you should list the algorithms in the order which you'd like to see them used by someone else when encrypting a message to your key. If you don't include 3DES, it will be automatically added at the end.

Note: that there are many factors that go into choosing an algorithm (for example, your key may not be the only recipient), and so the remote OpenPGP application being used to send to you may or may not follow your exact chosen order for a given message. It will, however, only choose an algorithm that is present on the preference list of every recipient key. See also the INTEROPERABILITY WITH OTHER OPENPGP PROGRAMS section below.

addkey

Add a subkey to this key.

addcardkey

Generate a subkey on a card and add it to this key.

keytocard

Transfer the selected secret subkey (or the primary key if no subkey has been selected) to a smartcard. The secret key in the keyring will be replaced by a stub if the key could be stored successfully on the card and you use the save command later. Only certain key types may be transferred to the card. A sub menu allows you to select on what card to store the key.

Note: that it is not possible to get that key back from the card - if the card gets broken your secret key will be lost unless you have a backup somewhere.

bkuptocard

file Restore the given file to a card. This command may be used to restore a backup key (as generated during card initialization) to a new card. In almost all cases this will be the encryption key. You should use this command only with the corresponding public key and make sure that the file given as argument is indeed the backup to restore. You should then select 2 to restore as encryption key. You will first be asked to enter the passphrase of the backup key and then for the Admin PIN of the card.

delkey

Remove a subkey (secondary key).

Note: that it is not possible to retract a subkey, once it has been send to the public (i.e. to a keyserver). In that case you better use revkey.

Note: that this only deletes the public part of a key.

revkey

Revoke a subkey.

expire

Change the key or subkey expiration time. If a subkey is selected, the expiration time of this subkey will be changed. With no selection, the key expiration of the primary key is changed.

trust

Change the owner trust value for the key. This updates the trust-db immediately and no save is required.

disable

enable Disable or enable an entire key. A disabled key cannot normally be used for encryption.

addrevoker

Add a designated revoker to the key. This takes one optional argument: "sensitive". If a designated revoker is marked as sensitive, it will not be exported by default (see export-options).

passwd

Change the passphrase of the secret key.

toggle

This is dummy command which exists only for backward compatibility.

clean

Compact (by removing all signatures except the selfsig) any user ID that is no longer usable (e.g. revoked, or expired). Then, remove any signatures that are not usable by the trust calculations. Specifically, this removes any signature that does not validate, any signature that is superseded by a later signature, revoked signatures, and signatures issued by keys that are not present on the keyring.

minimize

Make the key as small as possible. This removes all signatures from each user ID except for the most recent self-signature.

change-usage

Change the usage flags (capabilities) of the primary key or of subkeys. These usage flags (e.g. Certify, Sign, Authenticate, Encrypt) are set during key creation. Sometimes it is useful to have the opportunity to change them (for example to add Authenticate) after they have been created. Please take care when doing this; the allowed usage flags depend on the key algorithm.

cross-certify

Add cross-certification signatures to signing subkeys that may not currently have them. Cross-certification signatures protect against a subtle attack against signing subkeys. See `--require-cross-certification`. All new keys generated have this signature by default, so this command is only useful to bring older keys up to date.

save

Save all changes to the keyring and quit.

quit

Quit the program without updating the keyring.

The listing shows you the key with its secondary keys and all user IDs. The primary user ID is indicated by a dot, and selected keys or user IDs are indicated by an asterisk. The trust value is displayed with the primary key: "trust" is the assigned owner trust and "validity" is the calculated validity of the key. Validity values are also displayed for all user IDs. For possible values of trust, see: [trust-values].

--sign-key name

Signs a public key with your secret key. This is a shortcut version of the subcommand "sign" from `--edit`.

--lsign-key name

Signs a public key with your secret key but marks it as non-exportable. This is a shortcut version of the subcommand "lsign" from `--edit-key`.

--quick-sign-key fpr [names]**--quick-lsign-key fpr [names]**

Directly sign a key from the passphrase without any further user interaction. The fpr must be the verified primary fingerprint of a key in the local keyring. If no names are given, all useful user ids are signed; with given [names] only useful user ids matching one of these names are signed. By default, or if a name is prefixed with a '*', a case insensitive substring match is used. If a name is prefixed with a '=' a case sensitive exact match is done.

The command `--quick-lsign-key` marks the signatures as non-exportable. If such a non-exportable signature already exists the `--quick-sign-key` turns it into a exportable signature. If you need to update an existing signature, for example to add or change notation data, you need to use the option `--force-sign-key`.

This command uses reasonable defaults and thus does not provide the full flexibility of the "sign" subcommand from `--edit-key`. Its intended use is to help unattended key signing by utilizing a list of verified fingerprints.

--quick-add-uid user-id new-user-id

This command adds a new user id to an existing key. In contrast to the interactive subcommand `adduid` of `--edit-key` the `new-user-id` is added verbatim with only leading and trailing white space removed, it is expected to be UTF-8 encoded, and no checks on its form are applied.

--quick-revoke-uid user-id user-id-to-revoke

This command revokes a user ID on an existing key. It cannot be used to revoke the last user ID on key (some non-revoked user ID must remain), with revocation reason "User ID is no longer valid". If you want to specify a different revocation reason, or to supply supplementary revocation text, you should use the interactive sub-command `revuid` of `--edit-key`.

--quick-revoke-sig fpr signing-fpr [names]

This command revokes the key signatures made by `signing-fpr` from the key specified by the fingerprint fpr. With names given only the signatures on user ids of the key matching any of the given names are affected (see `--quick-sign-key`). If a revocation already exists a notice is printed instead of creating a new revocation; no error is returned in this case.

Note: that key signature revocations may be superseded by a newer key signature and in turn again revoked.

--quick-set-primary-uid user-id primary-user-id

This command sets or updates the primary user ID flag on an existing key. `user-id` specifies the key and `primary-user-id` the user ID which shall be flagged as the primary user ID. The primary user ID flag is removed from all other user ids and the timestamp of all affected self-signatures is set one second ahead.

--change-passphrase user-id**--passwd user-id**

Change the passphrase of the secret key belonging to the certificate specified as user-id. This is a shortcut for the sub-command passwd of the edit key menu. When using together with the option **--dry-run** this will not actually change the passphrase but check that the current passphrase is correct.

OPTIONS

gpg features a bunch of options to control the exact behaviour and to change the default configuration.

Long options can be put in an options file (default `~/gnupg/gpg.conf`). Short option names will not work – for example, `armor` is a valid option for the options file, while `a` is not. Do not write the 2 dashes, but simply the name of the option and any required arguments. Lines with a hash (`#`) as the first non-white-space character are ignored. Commands may be put in this file too, but that is not generally useful as the command will execute automatically with every execution of gpg.

Please remember that option parsing stops as soon as a non-option is encountered, you can explicitly stop parsing by using the special option **--**.

How to Change the Configuration

These options are used to change the configuration and most of them are usually found in the option file.

--default-key name

Use name as the default key to sign with. If this option is not used, the default key is the first key found in the secret keyring.

Note: that `-u` or **--local-user** overrides this option. This option may be given multiple times. In this case, the last key for which a secret key is available is used. If there is no secret key available for any of the specified values, GnuPG will not emit an error message but continue as if this option wasn't given.

--default-recipient name

Use name as default recipient if option **--recipient** is not used and don't ask if this is a valid one. name must be non-empty.

--default-recipient-self

Use the default key as default recipient if option **--recipient** is not used and don't ask if this is a valid one. The default key is the first one from the secret keyring or the one set with **--default-key**.

--no-default-recipient

Reset **--default-recipient** and **--default-recipient-self**. Should not be used in an option file.

--verbose**-v**

Give more information during processing. If used twice, the input data is listed in detail.

--no-verbose

Reset verbose level to 0. Should not be used in an option file.

--quiet**-q**

Try to be as quiet as possible. Should not be used in an option file.

--batch**--no-batch**

Use batch mode. Never ask, do not allow interactive commands. **--no-batch** disables this option.

Note: that even with a filename given on the command line, gpg might still need to read from STDIN (in particular if gpg figures that the input is a detached signature and no data file has been specified). Thus if you do not want to feed data via STDIN, you should connect STDIN to `/dev/null`.

It is highly recommended to use this option along with the options **--status-fd** and **--with-colons** for any unattended use of gpg. Should not be used in an option file.

--no-tty

Make sure that the TTY (terminal) is never used for any output. This option is needed in some cases because GnuPG sometimes prints warnings to the TTY even if `--batch` is used.

--yes

Assume "yes" on most questions. Should not be used in an option file.

--no

Assume "no" on most questions. Should not be used in an option file.

--list-options parameters

This is a space or comma delimited string that gives options used when listing keys and signatures (that is, `--list-keys`, `--check-signatures`, `--list-public-keys`, `--list-secret-keys`, and the `--edit-key` functions). Options can be prepended with a `no-` (after the two dashes) to give the opposite meaning. The options are:

show-photos

Causes `--list-keys`, `--check-signatures`, `--list-public-keys`, and `--list-secret-keys` to display any photo IDs attached to the key. Defaults to no. See also `--photo-viewer`. Does not work with `--with-colons`: see `--attribute-fd` for the appropriate way to get photo data for scripts and other frontends.

show-usage

Show usage information for keys and subkeys in the standard key listing. This is a list of letters indicating the allowed usage for a key (E=encryption, S=signing, C=certification, A=authentication). Defaults to yes.

show-policy-urls

Show policy URLs in the `--check-signatures` listings. Defaults to no.

show-notations***show-std-notations******show-user-notations***

Show all, IETF standard, or user-defined signature notations in the `--check-signatures` listings. Defaults to no.

show-keyserver-urls

Show any preferred keyserver URL in the `--check-signatures` listings. Defaults to no.

show-uid-validity

Display the calculated validity of user IDs during key listings. Defaults to yes.

show-unusable-uids

Show revoked and expired user IDs in key listings. Defaults to no.

show-unusable-subkeys

Show revoked and expired subkeys in key listings. Defaults to no.

show-keyring

Display the keyring name at the head of key listings to show which keyring a given key resides on. Defaults to no.

show-sig-expire

Show signature expiration dates (if any) during `--check-signatures` listings. Defaults to no.

show-sig-subpackets

Include signature subpackets in the key listing. This option can take an optional argument list of the subpackets to list. If no argument is passed, list all subpackets. Defaults to no. This option is only meaningful when using `--with-colons` along with `--check-signatures`.

show-only-fpr-mbox

For each user-id which has a valid mail address print only the fingerprint followed by the mail address.

--verify-options parameters

This is a space or comma delimited string that gives options used when verifying signatures. Options can be prepended with a "no-" to give the opposite meaning. The options are:

show-photos

Display any photo IDs present on the key that issued the signature. Defaults to no. See also --photo-viewer.

show-policy-urls

Show policy URLs in the signature being verified. Defaults to yes.

show-notations**show-std-notations****show-user-notations**

Show all, IETF standard, or user-defined signature notations in the signature being verified. Defaults to IETF standard.

show-keyserver-urls

Show any preferred keyserver URL in the signature being verified. Defaults to yes.

show-uid-validity

Display the calculated validity of the user IDs on the key that issued the signature. Defaults to yes.

show-unusable-uids

Show revoked and expired user IDs during signature verification. Defaults to no.

show-primary-uid-only

Show only the primary user ID during signature verification. That is all the AKA lines as well as photo IDs are not shown with the signature verification status.

pka-lookups

Enable PKA lookups to verify sender addresses.

Note: that PKA is based on DNS, and so enabling this option may disclose information on when and what signatures are verified or to whom data is encrypted. This is similar to the "web bug" described for the --auto-key-retrieve option.

pka-trust-increase

Raise the trust in a signature to full if the signature passes PKA validation. This option is only meaningful if pka-lookups is set.

--enable-large-rsa**--disable-large-rsa**

With --generate-key and --batch, enable the creation of RSA secret keys as large as 8192 bit.

Note: 8192 bit is more than is generally recommended. These large keys don't significantly improve security, but they are more expensive to use, and their signatures and certifications are larger. This option is only available if the binary was build with large-secmem support.

--enable-dsa2**--disable-dsa2**

Enable hash truncation for all DSA keys even for old DSA Keys up to 1024 bit. This is also the default with --openpgp.

Note: that older versions of GnuPG also required this flag to allow the generation of DSA larger than 1024 bit.

--photo-viewer string

This is the command line that should be run to view a photo ID. "%i" will be expanded to a filename containing the photo. "%I" does the same, except the file will not be deleted once the viewer exits. Other flags are "%k" for the key ID, "%K" for the long key ID, "%f" for the key fingerprint, "%t" for the extension of the image type (e.g. "jpg"), "%T" for the MIME type of the image (e.g. "image/jpeg"), "%v" for the single-character calculated validity of the image being viewed (e.g. "f"), "%V" for the calculated validity as a string (e.g. "full"), "%U" for a base32 encoded hash of the user ID, and "%%" for an actual percent sign. If neither %i or %I are present, then the photo will be supplied to the viewer on standard input.

On Unix the default viewer is `xloadimage -fork -quiet -title "KeyID 0x%k" STDIN` with a fallback to `display -title "KeyID 0x%k" %i` and finally to `xdg-open %i`. On Windows `!ShellExecute 400 %i` is used; here the command is a meta command to use that API call followed by a wait time in milliseconds which is used to give the viewer time to read the temporary image file before `gpg` deletes it again.

Note: that if your image viewer program is not secure, then executing it from `gpg` does not make it secure.

--exec-path string

Sets a list of directories to search for photo viewers. If not provided photo viewers use the `PATH` environment variable.

--keyring file

Add file to the current list of keyrings. If file begins with a tilde and a slash, these are replaced by the `$HOME` directory. If the filename does not contain a slash, it is assumed to be in the GnuPG home directory ("`~/.gnupg`" unless `--homedir` or `$GNUPGHOME` is used).

Note: that this adds a keyring to the current list. If the intent is to use the specified keyring alone, use `--keyring` along with `--no-default-keyring`.

If the option `--no-keyring` has been used no keyrings will be used at all.

--primary-keyring file

This is a variant of `--keyring` and designates file as the primary public keyring. This means that newly imported keys (via `--import` or `keyserver --recv-from`) will go to this keyring.

--secret-keyring file

This is an obsolete option and ignored. All secret keys are stored in the "`private-keys-v1.d`" directory below the GnuPG home directory.

--trustdb-name file

Use file instead of the default trustdb. If file begins with a tilde and a slash, these are replaced by the `$HOME` directory. If the filename does not contain a slash, it is assumed to be in the GnuPG home directory ("`~/.gnupg`" if `--homedir` or `$GNUPGHOME` is not used).

--homedir dir

Set the name of the home directory to `dir`. If this option is not used, the home directory defaults to "`~/.gnupg`". It is only recognized when given on the command line. It also overrides any home directory stated through the environment variable "`GNUPGHOME`" or (on Windows systems) by means of the Registry entry `HKCU\Software\GNU\GnuPG:HomeDir`.

On Windows systems it is possible to install GnuPG as a portable application. In this case only this command line option is considered, all other ways to set a home directory are ignored.

To install GnuPG as a portable application under Windows, create an empty file named "`gpgconf.ctl`" in the same directory as the tool "`gpgconf.exe`". The root of the installation is then that directory; or, if "`gpgconf.exe`" has been installed directly below a directory named "`bin`", its parent directory. You also need to make sure that the following directories exist and are writable: "`ROOT/home`" for the GnuPG home and "`ROOT/var/cache/gnupg`" for internal cache files.

--display-charset name

Set the name of the native character set. This is used to convert some informational strings like user IDs to the proper UTF-8 encoding.

Note: that this has nothing to do with the character set of data to be encrypted or signed; GnuPG does not recode user-supplied data. If this option is not used, the default character set is determined from the current locale. A verbosity level of 3 shows the chosen set. This option should not be used on Windows. Valid values for name are:

iso-8859-1

This is the Latin 1 set.

iso-8859-2

The Latin 2 set.

iso-8859-15

This is currently an alias for the Latin 1 set.

koi8-r

The usual Russian set (RFC-1489).

utf-8

Bypass all translations and assume that the OS uses native UTF-8 encoding.

--utf8-strings**--no-utf8-strings**

Assume that command line arguments are given as UTF-8 strings. The default (**--no-utf8-strings**) is to assume that arguments are encoded in the character set as specified by **--display-charset**. These options affect all following arguments. Both options may be used multiple times. This option should not be used in an option file.

This option has no effect on Windows. There the internal used UTF-8 encoding is translated for console input and output. The command line arguments are expected as Unicode and translated to UTF-8. Thus when calling this program from another, make sure to use the Unicode version of `CreateProcess`.

--options file

Read options from file and do not try to read them from the default options file in the homedir (see **--homedir**). This option is ignored if used in an options file.

--no-options

Shortcut for **--options /dev/null**. This option is detected before an attempt to open an option file. Using this option will also prevent the creation of a `~/.gnupg` homedir.

-z n**--compress-level n****--bzip2-compress-level n**

Set compression level to “n” for the ZIP and ZLIB compression algorithms. The default is to use the default compression level of zlib (normally 6). **--bzip2-compress-level** sets the compression level for the BZIP2 compression algorithm (defaulting to 6 as well). This is a different option from **--compress-level** since BZIP2 uses a significant amount of memory for each additional compression level. **-z** sets both. A value of 0 for n disables compression.

--bzip2-decompress-lowmem

Use a different decompression method for BZIP2 compressed files. This alternate method uses a bit more than half the memory, but also runs at half the speed. This is useful under extreme low memory circumstances when the file was originally compressed at a high **--bzip2-compress-level**.

--mangle-dos-filenames**--no-mangle-dos-filenames**

Older version of Windows cannot handle filenames with more than one dot. **--mangle-dos-filenames** causes GnuPG to replace (rather than add to) the extension of an output filename to avoid this problem. This option is off by default and has no effect on non-Windows platforms.

--ask-cert-level**--no-ask-cert-level**

When making a key signature, prompt for a certification level. If this option is not specified, the certification level used is set via **--default-cert-level**. See **--default-cert-level** for information on the specific levels and how they are used. **--no-ask-cert-level** disables this option. This option defaults to no.

--default-cert-level n

The default to use for the check level when signing a key.

0

means you make no particular claim as to how carefully you verified the key.

1

means you believe the key is owned by the person who claims to own it but you could not, or did not verify the key at all. This is useful for a "persona" verification, where you sign the key of a pseudonymous user.

2

means you did casual verification of the key. For example, this could mean that you verified the key fingerprint and checked the user ID on the key against a photo ID.

3

means you did extensive verification of the key. For example, this could mean that you verified the key fingerprint with the owner of the key in person, and that you checked, by means of a hard to forge document with a photo ID (such as a passport) that the name of the key owner matches the name in the user ID on the key, and finally that you verified (by exchange of email) that the email address on the key belongs to the key owner.

Note: that the examples given above for levels 2 and 3 are just that: examples. In the end, it is up to you to decide just what "casual" and "extensive" mean to you.

This option defaults to 0 (no particular claim).

--min-cert-level

When building the trust database, treat any signatures with a certification level below this as invalid. Defaults to 2, which disregards level 1 signatures.

Note: that level 0 "no particular claim" signatures are always accepted.

--trusted-key long key ID or fingerprint

Assume that the specified key (which should be given as fingerprint) is as trustworthy as one of your own secret keys. This option is useful if you don't want to keep your secret keys (or one of them) online but still want to be able to check the validity of a given recipient's or signator's key. If the given key is not locally available but an LDAP keyserver is configured the missing key is imported from that server.

--trust-model {pgp|classic|tofu|tofu+pgp|direct|always|auto}

Set what trust model GnuPG should follow. The models are:

pgp

This is the Web of Trust combined with trust signatures as used in PGP 5.x and later. This is the default trust model when creating a new trust database.

classic

This is the standard Web of Trust as introduced by PGP 2.

tofu

TOFU stands for Trust On First Use. In this trust model, the first time a key is seen, it is memorized. If later another key with a user id with the same email address is seen, both keys are marked as suspect. In that case, the next time either is used, a warning is displayed describing the conflict, why it might have occurred (either the user generated a new key and failed to cross sign the old and new keys, the key is forgery, or a man-in-the-middle attack is being attempted), and the user is prompted to manually confirm the validity of the key in question.

Because a potential attacker is able to control the email address and thereby circumvent the conflict detection algorithm by using an email address that is similar in appearance to a trusted email address, whenever a message is verified, statistics about the number of messages signed with the key are shown. In this way, a user can easily identify attacks using fake keys for regular correspondents.

When compared with the Web of Trust, TOFU offers significantly weaker security guarantees. In particular, TOFU only helps ensure consistency (that is, that the binding between a key and email address doesn't change). A major advantage of TOFU is that it requires little maintenance to use correctly. To use the web of trust properly, you need to actively sign keys and mark users as trusted introducers. This is a time-consuming process and anecdotal evidence suggests that even security-conscious users rarely take the time to do this thoroughly and instead rely on an ad-hoc TOFU process.

In the TOFU model, policies are associated with bindings between keys and email addresses (which are extracted from user ids and normalized). There are five policies, which can be set manually using the `--tofu-policy` option. The default policy can be set using the `--tofu-default-policy` option.

The TOFU policies are: auto, good, unknown, bad and ask. The auto policy is used by default (unless overridden by `--tofu-default-policy`) and marks a binding as marginally trusted. The good, unknown and bad policies mark a binding as fully trusted, as having unknown trust or as having trust never, respectively. The unknown policy is useful for just using TOFU to detect conflicts, but to never assign positive trust to a binding. The final policy, ask prompts the user to indicate the binding's trust. If batch mode is enabled (or input is inappropriate in the context), then the user is not prompted and the undefined trust level is returned.

tofu+pgp

This trust model combines TOFU with the Web of Trust. This is done by computing the trust level for each model and then taking the maximum trust level where the trust levels are ordered as follows: unknown < undefined < marginal < fully < ultimate < expired < never.

By setting `--tofu-default-policy=unknown`, this model can be used to implement the web of trust with TOFU's conflict detection algorithm, but without its assignment of positive trust values, which some security-conscious users don't like.

direct

Key validity is set directly by the user and not calculated via the Web of Trust. This model is solely based on the key and does not distinguish user IDs.

Note: that when changing to another trust model the trust values assigned to a key are transformed into ownertrust values, which also indicate how you trust the owner of the key to sign other keys.

always

Skip key validation and assume that used keys are always fully valid. You generally won't use this unless you are using some external validation scheme. This option also suppresses the "[uncertain]" tag printed with signature checks when there is no evidence that the user ID is bound to the key.

Note: that this trust model still does not allow the use of expired, revoked, or disabled keys.

auto

Select the trust model depending on whatever the internal trust database says. This is the default model if such a database already exists.

Note: that a tofu trust model is not considered here and must be enabled explicitly.

--auto-key-locate mechanisms**--no-auto-key-locate**

GnuPG can automatically locate and retrieve keys as needed using this option. This happens when encrypting to an email address (in the "user@example.com" form), and there are no "user@example.com" keys on the local keyring. This option takes any number of the mechanisms listed below, in the order they are to be tried. Instead of listing the mechanisms as comma delimited arguments, the option may also be given several times to add more mechanism. The option `--no-auto-key-locate` or the mechanism "clear" resets the list. The default is "local,wkd".

cert

Locate a key using DNS CERT, as specified in RFC-4398.

pka

Locate a key using DNS PKA.

dane

Locate a key using DANE, as specified in draft-ietf-dane-openpgpkey-05.txt.

wkd

Locate a key using the Web Key Directory protocol.

ldap

Using DNS Service Discovery, check the domain in question for any LDAP key servers to use. If this fails, attempt to locate the key using the PGP Universal method of checking "ldap://keys.{thedomain}".

ntds

Locate the key using the Active Directory (Windows only). This method also allows to search by fingerprint using the command `--locate-external-key`.

Note: that this mechanism is actually a shortcut for the mechanism "keyserver" but using "ldap:///" as the keyserver.

keyserver

Locate a key using a keyserver. This method also allows to search by fingerprint using the command `--locate-external-key` if any of the configured key servers is an LDAP server.

keyserver-URL

In addition, a keyserver URL as used in the dirmngr configuration may be used here to query that particular keyserver. This method also allows to search by fingerprint using the command `--locate-external-key` if the URL specifies an LDAP server.

local

Locate the key using the local keyrings. This mechanism allows the user to select the order a local key lookup is done. Thus using "`--auto-key-locate local`" is identical to `--no-auto-key-locate`.

nodefault

This flag disables the standard local key lookup, done before any of the mechanisms defined by the `--auto-key-locate` are tried. The position of this mechanism in the list does not matter. It is not required if local is also used.

clear

Clear all defined mechanisms. This is useful to override mechanisms given in a config file.

Note: that a nodefault in mechanisms will also be cleared unless it is given after the clear.

--auto-key-import**--no-auto-key-import**

This is an offline mechanism to get a missing key for signature verification and for later encryption to this key. If this option is enabled and a signature includes an embedded key, that key is used to verify the signature and on verification success that key is imported. The default is `--no-auto-key-import`.

On the sender (signing) site the option `--include-key-block` needs to be used to put the public part of the signing key as "Key Block subpacket" into the signature.

--auto-key-retrieve**--no-auto-key-retrieve**

These options enable or disable the automatic retrieving of keys from a keyserver when verifying signatures made by keys that are not on the local keyring. The default is `--no-auto-key-retrieve`.

The order of methods tried to lookup the key is:

1

If the option `--auto-key-import` is set and the signature includes an embedded key, that key is used to verify the signature and on verification success that key is imported.

2

If a preferred keyserver is specified in the signature and the option `honor-keyserver-url` is active (which is not the default), that keyserver is tried.

Note: that the creator of the signature uses the option `--sig-keyserver-url` to specify the preferred keyserver for data signatures.

3

If the signature has the Signer's UID set (e.g. using `--sender` while creating the signature) a Web Key Directory (WKD) lookup is done. This is the default configuration but can be disabled by removing WKD from the `auto-key-locate` list or by using the option `--disable-signer-uid`.

4

If the option `honor-pka-record` is active, the legacy PKA method is used.

5

If any keyserver is configured and the Issuer Fingerprint is part of the signature (since GnuPG 2.1.16), the configured keyservers are tried.

Note: that this option makes a "web bug" like behavior possible. Keyserver or Web Key Directory operators can see which keys you request, so by sending you a message signed by a brand new key (which you naturally will not have on your local keyring), the operator can tell both your IP address and the time when you verified the signature.

--keyid-format {none|short|0xshort|long|0xlong}

Select how to display key IDs. "none" does not show the key ID at all but shows the fingerprint in a separate line. "short" is the traditional 8-character key ID. "long" is the more accurate (but less convenient) 16-character key ID. Add an "0x" to either to include an "0x" at the beginning of the key ID, as in 0x99242560.

Note: that this option is ignored if the option `--with-colons` is used.

--keyserver name

This option is deprecated – please use the `--keyserver` in `"dirmngr.conf"` instead.

Use name as your keyserver. This is the server that `--receive-keys`, `--send-keys`, and `--search-keys` will communicate with to receive keys from, send keys to, and search for keys on. The format of the name is a URI: `"scheme:[//]keyservername[:port]"` The scheme is the type of keyserver: `"hkp"/"hkps"` for the HTTP (or compatible) keyservers or `"ldap"/"ldaps"` for the LDAP keyservers.

Note: that your particular installation of GnuPG may have other keyserver types available as well. Keyserver schemes are case-insensitive.

Most keyservers synchronize with each other, so there is generally no need to send keys to more than one server. The keyserver `hkp://keys.gnupg.net` uses round robin DNS to give a different keyserver each time you use it.

--keyserver-options {name=value}

This is a space or comma delimited string that gives options for the keyserver. Options can be prefixed with a "no-" to give the opposite meaning. Valid import-options or export-options may be used here as well to apply to importing (*--recv-key*) or exporting (*--send-key*) a key from a keyserver. While not all options are available for all keyserver types, some common options are:

include-revoked

When searching for a key with *--search-keys*, include keys that are marked on the keyserver as revoked.

Note: that not all keyserver differentiate between revoked and unrevoked keys, and for such keyserver this option is meaningless.

Note: also that most keyserver do not have cryptographic verification of key revocations, and so turning this option off may result in skipping keys that are incorrectly marked as revoked.

include-disabled

When searching for a key with *--search-keys*, include keys that are marked on the keyserver as disabled.

Note: that this option is not used with HKP keyserver.

auto-key-retrieve

This is an obsolete alias for the option *auto-key-retrieve*. Please do not use it; it will be removed in future versions..

honor-keyserver-url

When using *--refresh-keys*, if the key in question has a preferred keyserver URL, then use that preferred keyserver to refresh the key from. In addition, if *auto-key-retrieve* is set, and the signature being verified has a preferred keyserver URL, then use that preferred keyserver to fetch the key from.

Note: that this option introduces a "web bug": The creator of the key can see when the key is refreshed. Thus this option is not enabled by default.

honor-pka-record

If *--auto-key-retrieve* is used, and the signature being verified has a PKA record, then use the PKA information to fetch the key. Defaults to "yes".

include-subkeys

When receiving a key, include subkeys as potential targets.

Note: that this option is not used with HKP keyserver, as they do not support retrieving keys by subkey id.

timeout***http-proxy=value******verbose******debug******check-cert******ca-cert-file***

These options have no more function since GnuPG 2.1. Use the *dirmngr* configuration options instead.

The default list of options is: "self-sigs-only, import-clean, repair-keys, repair-pks-subkey-bug, export-attributes, honor-pka-record". However, if the actual used source is an LDAP server "no-self-sigs-only" is assumed unless "self-sigs-only" has been explicitly configured.

--completes-needed n

Number of completely trusted users to introduce a new key signer (defaults to 1).

--marginals-needed n

Number of marginally trusted users to introduce a new key signer (defaults to 3)

--tofu-default-policy {auto|good|unknown|bad|ask}

The default TOFU policy (defaults to auto). For more information about the meaning of this option, see: [trust-model-tofu].

--max-cert-depth n

Maximum depth of a certification chain (default is 5).

--no-sig-cache

Do not cache the verification status of key signatures. Caching gives a much better performance in key listings. However, if you suspect that your public keyring is not safe against write modifications, you can use this option to disable the caching. It probably does not make sense to disable it because all kind of damage can be done if someone else has write access to your public keyring.

--auto-check-trustdb**--no-auto-check-trustdb**

If GnuPG feels that its information about the Web of Trust has to be updated, it automatically runs the `--check-trustdb` command internally. This may be a time consuming process. `--no-auto-check-trustdb` disables this option.

--use-agent**--no-use-agent**

This is dummy option. gpg always requires the agent.

--gpg-agent-info

This is dummy option. It has no effect when used with gpg.

--agent-program file

Specify an agent program to be used for secret key operations. The default value is determined by running `gpgconf` with the option `--list-dirs`.

Note: that the pipe symbol (`|`) is used for a regression test suite hack and may thus not be used in the file name.

--dirmngr-program file

Specify a `dirmngr` program to be used for keyserver access. The default value is `"/Users/Shared/Jenkins/Home/jobs/MacGPG-master/workspace/build/dist/arm64/bin/dirmngr"`.

--disable-dirmngr

Entirely disable the use of the `Dirmngr`.

--no-autostart

Do not start the `gpg-agent` or the `dirmngr` if it has not yet been started and its service is required. This option is mostly useful on machines where the connection to `gpg-agent` has been redirected to another machines. If `dirmngr` is required on the remote machine, it may be started manually using `gpgconf --launch dirmngr`.

--lock-once

Lock the databases the first time a lock is requested and do not release the lock until the process terminates.

--lock-multiple

Release the locks every time a lock is no longer needed. Use this to override a previous `--lock-once` from a config file.

--lock-never

Disable locking entirely. This option should be used only in very special environments, where it can be assured that only one process is accessing those files. A bootable floppy with a stand-alone encryption system will probably use this. Improper usage of this option may lead to data and key corruption.

--exit-on-status-write-error

This option will cause write errors on the status FD to immediately terminate the process. That should in fact be the default but it never worked this way and thus we need an option to enable this, so that the change won't break applications which close their end of a status fd connected pipe too early. Using this option along with `--enable-progress-filter` may be used to cleanly cancel long running gpg operations.

--limit-card-insert-tries n

With `n` greater than 0 the number of prompts asking to insert a smartcard gets limited to `N-1`. Thus with a value of 1 gpg won't at all ask to insert a card if none has been inserted at startup. This option is useful in the configuration file in case an application does not know about the smartcard support and waits ad infinitum for an inserted card.

--no-random-seed-file

GnuPG uses a file to store its internal random pool over invocations. This makes random generation faster; however sometimes write operations are not desired. This option can be used to achieve that with the cost of slower random generation.

--no-greeting

Suppress the initial copyright message.

--no-secmem-warning

Suppress the warning about "using insecure memory".

--no-permission-warning

Suppress the warning about unsafe file and home directory (--homedir) permissions.

Note: that the permission checks that GnuPG performs are not intended to be authoritative, but rather they simply warn about certain common permission problems. Do not assume that the lack of a warning means that your system is secure.

Note: that the warning for unsafe --homedir permissions cannot be suppressed in the gpg.conf file, as this would allow an attacker to place an unsafe gpg.conf file in place, and use this file to suppress warnings about itself. The --homedir permissions warning may only be suppressed on the command line.

--require-secmem**--no-require-secmem**

Refuse to run if GnuPG cannot get secure memory. Defaults to no (i.e. run, but give a warning).

--require-cross-certification**--no-require-cross-certification**

When verifying a signature made from a subkey, ensure that the cross certification "back signature" on the subkey is present and valid. This protects against a subtle attack against subkeys that can sign. Defaults to --require-cross-certification for gpg.

--expert**--no-expert**

Allow the user to do certain nonsensical or "silly" things like signing an expired or revoked key, or certain potentially incompatible things like generating unusual key types. This also disables certain warning messages about potentially incompatible actions. As the name implies, this option is for experts only. If you don't fully understand the implications of what it allows you to do, leave this off. --no-expert disables this option.

Key Related Options

--recipient name**-r**

Encrypt for user id name. If this option or --hidden-recipient is not specified, GnuPG asks for the user-id unless --default-recipient is given.

--hidden-recipient name**-R**

Encrypt for user ID name, but hide the key ID of this user's key. This option helps to hide the receiver of the message and is a limited countermeasure against traffic analysis. If this option or --recipient is not specified, GnuPG asks for the user ID unless --default-recipient is given.

--recipient-file file**-f**

This option is similar to --recipient except that it encrypts to a key stored in the given file. file must be the name of a file containing exactly one key. gpg assumes that the key in this file is fully valid.

--hidden-recipient-file file**-F**

This option is similar to --hidden-recipient except that it encrypts to a key stored in the given file. file must be the name of a file containing exactly one key. gpg assumes that the key in this file is fully valid.

--encrypt-to name

Same as --recipient but this one is intended for use in the options file and may be used with your own user-id as an "encrypt-to-self". These keys are only used when there are other recipients given either by use of --recipient or by the asked user id. No trust checking is performed for these user ids and even disabled keys can be used.

--hidden-encrypt-to name

Same as --hidden-recipient but this one is intended for use in the options file and may be used with your own user-id as a hidden "encrypt-to-self". These keys are only used when there are other recipients given either by use of --recipient or by the asked user id. No trust checking is performed for these user ids and even disabled keys can be used.

--no-encrypt-to

Disable the use of all --encrypt-to and --hidden-encrypt-to keys.

--group {name=value}

Sets up a named group, which is similar to aliases in email programs. Any time the group name is a recipient (-r or --recipient), it will be expanded to the values specified. Multiple groups with the same name are automatically merged into a single group.

The values are key IDs or fingerprints, but any key description is accepted.

Note: that a value with spaces in it will be treated as two different values.

Note: also there is only one level of expansion --- you cannot make an group that points to another group. When used from the command line, it may be necessary to quote the argument to this option to prevent the shell from treating it as multiple arguments.

--ungroup name

Remove a given entry from the --group list.

--no-groups

Remove all entries from the --group list.

--local-user name

-u

Use name as the key to sign with.

Note: that this option overrides --default-key.

--sender mbox

This option has two purposes. mbox must either be a complete user id with a proper mail address or just a mail address. When creating a signature this option tells gpg the user id of a key used to make a signature if the key was not directly specified by a user id. When verifying a signature the mbox is used to restrict the information printed by the TOFU code to matching user ids.

--try-secret-key name

For hidden recipients GPG needs to know the keys to use for trial decryption. The key set with --default-key is always tried first, but this is often not sufficient. This option allows setting more keys to be used for trial decryption. Although any valid user-id specification may be used for name it makes sense to use at least the long keyid to avoid ambiguities.

Note: that gpg-agent might pop up a pinentry for a lot keys to do the trial decryption. If you want to stop all further trial decryption you may use close-window button instead of the cancel button.

--try-all-secrets

Don't look at the key ID as stored in the message but try all secret keys in turn to find the right decryption key. This option forces the behaviour as used by anonymous recipients (created by using --throw-keyids or --hidden-recipient) and might come handy in case where an encrypted message contains a bogus key ID.

--skip-hidden-recipients**--no-skip-hidden-recipients**

During decryption skip all anonymous recipients. This option helps in the case that people use the hidden recipients feature to hide their own encrypt-to key from others. If one has many secret keys this may lead to a major annoyance because all keys are tried in turn to decrypt something which was not really intended for it. The drawback of this option is that it is currently not possible to decrypt a message which includes real anonymous recipients.

Input and Output

--armor**-a**

Create ASCII armored output. The default is to create the binary OpenPGP format.

--no-armor

Assume the input data is not in ASCII armored format.

--output file**-o file**

Write output to file. To write to stdout use - as the filename.

--max-output n

This option sets a limit on the number of bytes that will be generated when processing a file. Since OpenPGP supports various levels of compression, it is possible that the plaintext of a given message may be significantly larger than the original OpenPGP message. While GnuPG works properly with such messages, there is often a desire to set a maximum file size that will be generated before processing is forced to stop by the OS limits. Defaults to 0, which means "no limit".

--input-size-hint n

This option can be used to tell GPG the size of the input data in bytes. *n* must be a positive base-10 number. This option is only useful if the input is not taken from a file. GPG may use this hint to optimize its buffer allocation strategy. It is also used by the `--status-fd` line "PROGRESS" to provide a value for "total" if that is not available by other means.

--key-origin string[,url]

gpg can track the origin of a key. Certain origins are implicitly known (e.g. keyserver, web key directory) and set. For a standard import the origin of the keys imported can be set with this option. To list the possible values use "help" for string. Some origins can store an optional url argument. That URL can be appended to string after a comma.

--import-options <parameters>

This is a space or comma delimited string that gives options for importing keys. Options can be prepended with a "no-" to give the opposite meaning. The options are:

import-local-sigs

Allow importing key signatures marked as "local". This is not generally useful unless a shared keyring scheme is being used. Defaults to no.

keep-ownertrust

Normally possible still existing owner trust values of a key are cleared if a key is imported. This is in general desirable so that a formerly deleted key does not automatically gain an owner trust values merely due to import. On the other hand it is sometimes necessary to re-import a trusted set of keys again but keeping already assigned owner trust values. This can be achieved by using this option.

repair-pks-subkey-bug

During import, attempt to repair the damage caused by the PKS keyserver bug (pre version 0.9.6) that mangles keys with multiple subkeys.

Note: that this cannot completely repair the damaged key as some crucial data is removed by the keyserver, but it does at least give you back one subkey. Defaults to no for regular `--import` and to yes for keyserver `--receive-keys`.

import-show***show-only***

Show a listing of the key as imported right before it is stored. This can be combined with the option `--dry-run` to only look at keys; the option `show-only` is a shortcut for this combination. The command `--show-keys` is another shortcut for this.

Note: that suffixes like "#" for "sec" and "sbb" lines may or may not be printed.

import-export

Run the entire import code but instead of storing the key to the local keyring write it to the output. The export options `export-pka` and `export-dane` affect the output. This option can be used to remove all invalid parts from a key without the need to store it.

merge-only

During import, allow key updates to existing keys, but do not allow any new keys to be imported. Defaults to no.

import-clean

After import, compact (remove all signatures except the self-signature) any user IDs from the new key that are not usable. Then, remove any signatures from the new key that are not usable. This includes signatures that were issued by keys that are not present on the keyring. This option is the same as running the `--edit-key` command "clean" after import. Defaults to no.

self-sigs-only

Accept only self-signatures while importing a key. All other key signatures are skipped at an early import stage. This option can be used with `keyserver-options` to mitigate attempts to flood a key with bogus signatures from a keyserver. The drawback is that all other valid key signatures, as required by the Web of Trust are also not imported.

Note: that when using this option along with `import-clean` it suppresses the final clean step after merging the imported key into the existing key.

repair-keys

After import, fix various problems with the keys. For example, this reorders signatures, and strips duplicate signatures. Defaults to yes.

import-minimal

Import the smallest key possible. This removes all signatures except the most recent self-signature on each user ID. This option is the same as running the `--edit-key` command "minimize" after import. Defaults to no.

restore***import-restore***

Import in key restore mode. This imports all data which is usually skipped during import; including all GnuPG specific data. All other contradicting options are overridden.

`--import-filter {name=expr}`

`--export-filter {name=expr}`

These options define an import/export filter which are applied to the imported/exported keyblock right before it will be stored/written. `name` defines the type of filter to use, `expr` the expression to evaluate. The option can be used several times which then appends more expression to the same name.

The available filter types are:

keep-uid

This filter will keep a user id packet and its dependent packets in the keyblock if the expression evaluates to true.

drop-subkey

This filter drops the selected subkeys. Currently only implemented for `--export-filter`.

drop-sig

This filter drops the selected key signatures on user ids. Self-signatures are not considered. Currently only implemented for `--import-filter`.

For the syntax of the expression see the chapter "FILTER EXPRESSIONS". The property names for the expressions depend on the actual filter type and are indicated in the following table.

The available properties are:

uid

A string with the user id. (`keep-uid`)

mbx

The `addr-spec` part of a user id with mailbox or the empty string. (`keep-uid`)

key_algo

A number with the public key algorithm of a key or subkey packet. (`drop-subkey`)

key_created**key_created_d**

The first is the timestamp a public key or subkey packet was created. The second is the same but given as an ISO string, e.g. "2016-08-17". (drop-subkey)

fpr

The hexified fingerprint of the current subkey or primary key. (drop-subkey)

primary

Boolean indicating whether the user id is the primary one. (keep-uid)

expired

Boolean indicating whether a user id (keep-uid), a key (drop-subkey), or a signature (drop-sig) expired.

revoked

Boolean indicating whether a user id (keep-uid) or a key (drop-subkey) has been revoked.

disabled

Boolean indicating whether a primary key is disabled. (not used)

secret

Boolean indicating whether a key or subkey is a secret one. (drop-subkey)

usage

A string indicating the usage flags for the subkey, from the sequence "ecsa?". For example, a subkey capable of just signing and authentication would be an exact match for "sa". (drop-subkey)

sig_created**sig_created_d**

The first is the timestamp a signature packet was created. The second is the same but given as an ISO date string, e.g. "2016-08-17". (drop-sig)

sig_algo

A number with the public key algorithm of a signature packet. (drop-sig)

sig_digest_algo

A number with the digest algorithm of a signature packet. (drop-sig)

--export-options <parameters>

This is a space or comma delimited string that gives options for exporting keys. Options can be prepended with a "no-" to give the opposite meaning. The options are:

export-local-sigs

Allow exporting key signatures marked as "local". This is not generally useful unless a shared keyring scheme is being used. Defaults to no.

export-attributes

Include attribute user IDs (photo IDs) while exporting. Not including attribute user IDs is useful to export keys that are going to be used by an OpenPGP program that does not accept attribute user IDs. Defaults to yes.

export-sensitive-revkeys

Include designated revoker information that was marked as "sensitive". Defaults to no.

backup**export-backup**

Export for use as a backup. The exported data includes all data which is needed to restore the key or keys later with GnuPG. The format is basically the OpenPGP format but enhanced with GnuPG specific data. All other contradicting options are overridden.

export-clean

Compact (remove all signatures from) user IDs on the key being exported if the user IDs are not usable. Also, do not export any signatures that are not usable. This includes signatures that were issued by keys that are not present on the keyring. This option is the same as running the --edit-key command "clean" before export except that the local copy of the key is not modified. Defaults to no.

export-minimal

Export the smallest key possible. This removes all signatures except the most recent self-signature on each user ID. This option is the same as running the `--edit-key` command "minimize" before export except that the local copy of the key is not modified. Defaults to no.

export-pka

Instead of outputting the key material output PKA records suitable to put into DNS zone files. An ORIGIN line is printed before each record to allow diverting the records to the corresponding zone file.

export-dane

Instead of outputting the key material output OpenPGP DANE records suitable to put into DNS zone files. An ORIGIN line is printed before each record to allow diverting the records to the corresponding zone file.

--with-colons

Print key listings delimited by colons.

Note: that the output will be encoded in UTF-8 regardless of any `--display-charset` setting. This format is useful when GnuPG is called from scripts and other programs as it is easily machine parsed. The details of this format are documented in the file "doc/DETAILS", which is included in the GnuPG source distribution.

--fixed-list-mode

Do not merge primary user ID and primary key in `--with-colon` listing mode and print all timestamps as seconds since 1970-01-01. Since GnuPG 2.0.10, this mode is always used and thus this option is obsolete; it does not harm to use it though.

--legacy-list-mode

Revert to the pre-2.1 public key list mode. This only affects the human readable output and not the machine interface (i.e. `--with-colons`).

Note: that the legacy format does not convey suitable information for elliptic curves.

--with-fingerprint

Same as the command `--fingerprint` but changes only the format of the output and may be used together with another command.

--with-subkey-fingerprint

If a fingerprint is printed for the primary key, this option forces printing of the fingerprint for all subkeys. This could also be achieved by using the `--with-fingerprint` twice but by using this option along with `keyid-format "none"` a compact fingerprint is printed.

--with-icao-spelling

Print the ICAO spelling of the fingerprint in addition to the hex digits.

--with-keygrip

Include the keygrip in the key listings. In `--with-colons` mode this is implicitly enable for secret keys.

--with-key-origin

Include the locally held information on the origin and last update of a key in a key listing. In `--with-colons` mode this is always printed. This data is currently experimental and shall not be considered part of the stable API.

--with-wkd-hash

Print a Web Key Directory identifier along with each user ID in key listings. This is an experimental feature and semantics may change.

--with-secret

Include info about the presence of a secret key in public key listings done with `--with-colons`.

OpenPGP Protocol Specific Options

-t, --textmode

--no-textmode

Treat input files as text and store them in the OpenPGP canonical text form with standard "CRLF" line endings. This also sets the necessary flags to inform the recipient that the encrypted or signed data is text and may need its line endings converted back to whatever the local system uses. This option is useful when communicating between two platforms that have different line ending conventions (UNIX-like to Mac, Mac to Windows, etc). **--no-textmode** disables this option, and is the default.

--force-v3-sigs

--no-force-v3-sigs

--force-v4-certs

--no-force-v4-certs

These options are obsolete and have no effect since GnuPG 2.1.

--force-mdc

--disable-mdc

These options are obsolete and have no effect since GnuPG 2.2.8. The MDC is always used.

Note: If the creation of a legacy non-MDC message is exceptionally required, the option **--rfc2440** allows for this.

--disable-signer-uid

By default the user ID of the signing key is embedded in the data signature. As of now this is only done if the signing key has been specified with local-user using a mail address, or with sender. This information can be helpful for verifier to locate the key; see option **--auto-key-retrieve**.

--include-key-block

This option is used to embed the actual signing key into a data signature. The embedded key is stripped down to a single user id and includes only the signing subkey used to create the signature as well as valid encryption subkeys. All other info is removed from the key to keep it and thus the signature small. This option is the OpenPGP counterpart to the **gpgsm** option **--include-certs**.

--personal-cipher-preferences string

Set the list of personal cipher preferences to string. Use **gpg --version** to get a list of available algorithms, and use none to set no preference at all. This allows the user to safely override the algorithm chosen by the recipient key preferences, as GPG will only select an algorithm that is usable by all recipients. The most highly ranked cipher in this list is also used for the **--symmetric** encryption command.

--personal-digest-preferences string

Set the list of personal digest preferences to string. Use **gpg --version** to get a list of available algorithms, and use none to set no preference at all. This allows the user to safely override the algorithm chosen by the recipient key preferences, as GPG will only select an algorithm that is usable by all recipients. The most highly ranked digest algorithm in this list is also used when signing without encryption (e.g. **--clear-sign** or **--sign**).

--personal-compress-preferences string

Set the list of personal compression preferences to string. Use **gpg --version** to get a list of available algorithms, and use none to set no preference at all. This allows the user to safely override the algorithm chosen by the recipient key preferences, as GPG will only select an algorithm that is usable by all recipients. The most highly ranked compression algorithm in this list is also used when there are no recipient keys to consider (e.g. **--symmetric**).

--s2k-cipher-algo name

Use name as the cipher algorithm for symmetric encryption with a passphrase if **--personal-cipher-preferences** and **--cipher-algo** are not given. The default is AES-128.

--s2k-digest-algo name

Use name as the digest algorithm used to mangle the passphrases for symmetric encryption. The default is SHA-1.

--s2k-mode n

Selects how passphrases for symmetric encryption are mangled. If *n* is 0 a plain passphrase (which is in general not recommended) will be used, a 1 adds a salt (which should not be used) to the passphrase and a 3 (the default) iterates the whole process a number of times (see `--s2k-count`).

--s2k-count n

Specify how many times the passphrases mangling for symmetric encryption is repeated. This value may range between 1024 and 65011712 inclusive. The default is inquired from `gpg-agent`.

Note: that not all values in the 1024-65011712 range are legal and if an illegal value is selected, GnuPG will round up to the nearest legal value. This option is only meaningful if `--s2k-mode` is set to the default of 3.

Compliance Options

These options control what GnuPG is compliant to. Only one of these options may be active at a time.

Note: that the default setting of this is nearly always the correct one. See the INTEROPERABILITY WITH OTHER OPENPGP PROGRAMS section below before using one of these options.

--gnupg

Use standard GnuPG behavior. This is essentially OpenPGP behavior (see `--openpgp`), but with some additional workarounds for common compatibility problems in different versions of PGP. This is the default option, so it is not generally needed, but it may be useful to override a different compliance option in the `gpg.conf` file.

--openpgp

Reset all packet, cipher and digest options to strict OpenPGP behavior. Use this option to reset all previous options like `--s2k-*`, `--cipher-algo`, `--digest-algo` and `--compress-algo` to OpenPGP compliant values. All PGP workarounds are disabled.

--rfc4880

Reset all packet, cipher and digest options to strict RFC-4880 behavior.

Note: that this is currently the same thing as `--openpgp`.

--rfc4880bis

Enable experimental features from proposed updates to RFC-4880. This option can be used in addition to the other compliance options.

Warning: The behavior may change with any GnuPG release and created keys or data may not be usable with future GnuPG versions.

--rfc2440

Reset all packet, cipher and digest options to strict RFC-2440 behavior.

Note: that by using this option encryption packets are created in a legacy mode without MDC protection. This is dangerous and should thus only be used for experiments. See also option `--ignore-mdc-error`.

--pgp6

Set up all options to be as PGP 6 compliant as possible. This restricts you to the ciphers IDEA (if the IDEA plugin is installed), 3DES, and CAST5, the hashes MD5, SHA1 and RIPEMD160, and the compression algorithms none and ZIP. This also disables `--throw-keyids`, and making signatures with signing subkeys as PGP 6 does not understand signatures made by signing subkeys.

This option implies `--escape-from-lines`.

--pgp7

Set up all options to be as PGP 7 compliant as possible. This is identical to `--pgp6` except that MDCs are not disabled, and the list of allowable ciphers is expanded to add AES128, AES192, AES256, and TWOFISH.

--pgp8

Set up all options to be as PGP 8 compliant as possible. PGP 8 is a lot closer to the OpenPGP standard than previous versions of PGP, so all this does is disable `--throw-keyids` and set `--escape-from-lines`. All algorithms are allowed except for the SHA224, SHA384, and SHA512 digests.

--compliance string

This option can be used instead of one of the options above. Valid values for string are the above option names (without the double dash) and possibly others as shown when using "help" for string.

--min-rsa-length n

This option adjusts the compliance mode "de-vs" for stricter key size requirements. For example, a value of 3000 turns rsa2048 and dsa2048 keys into non-VS-NfD compliant keys.

--require-compliance

To check that data has been encrypted according to the rules of the current compliance mode, a gpg user needs to evaluate the status lines. This allows frontends to handle compliance check in a more flexible way. However, for scripted use the required evaluation of the status-line requires quite some effort; this option can be used instead to make sure that the gpg process exits with a failure if the compliance rules are not fulfilled.

Note: that this option has currently an effect only in "de-vs" mode

Doing Things One Usually Doesn't Want to Do

-n**--dry-run**

Don't make any changes (this is not completely implemented).

--list-only

Changes the behaviour of some commands. This is like --dry-run but different in some cases. The semantic of this option may be extended in the future. Currently it only skips the actual decryption pass and therefore enables a fast listing of the encryption keys.

-i**--interactive**

Prompt before overwriting any files.

--debug-level level

Select the debug level for investigating problems. level may be a numeric value or by a keyword:

None	No debugging at all. A value of less than 1 may be used of the keyword.
Basic	Some basic debug messages. A value between 1 and 2 may be used instead of the keyword.
Advanced	verbose debug messages. A value between 3 and 5 may be used instead of the keyword.
Expert	Even more detailed messages. A value between 6 and 8 may be used instead of the keyword.
Guru	All of the debug messages you can get. A value greater than 8 may be used instead of the keyword. The creation of hash tracing files is only enabled if the keyword is used.

How these messages are mapped to the actual debugging flags is not specified and may change with newer releases of this program. They are however carefully selected to best aid in debugging.

--debug flags

Set debugging flags. All flags are or-ed and flags may be given in C syntax (e.g. 0x0042) or as a comma separated list of flag names. To get a list of all supported flags the single word "help" can be used.

--debug-all

Set all useful debugging flags.

--debug-iolbf

Set stdout into line buffered mode. This option is only honored when given on the command line.

--faked-system-time epoch

This option is only useful for testing; it sets the system time back or forth to epoch which is the number of seconds elapsed since the year 1970. Alternatively epoch may be given as a full ISO time string (e.g. "20070924T154812").

If you suffix epoch with an exclamation mark (!), the system time will appear to be frozen at the specified time.

--enable-progress-filter

Enable certain PROGRESS status outputs. This option allows frontends to display a progress indicator while gpg is processing larger files. There is a slight performance overhead using it.

--status-fd n

Write special status strings to the file descriptor n. See the file DETAILS in the documentation for a listing of them.

--status-file file

Same as --status-fd, except the status data is written to file, file.

--logger-fd n

Write log output to file descriptor n and not to STDERR.

--log-file file**--logger-file file**

Same as --logger-fd, except the logger data is written to file, file. Use "socket://" to log to a socket.

Note: that in this version of gpg the option has only an effect if --batch is also used.

--attribute-fd n

Write attribute subpackets to the file descriptor n. This is most useful for use with --status-fd, since the status messages are needed to separate out the various subpackets from the stream delivered to the file descriptor.

--attribute-file file

Same as --attribute-fd, except the attribute data is written to file, file.

--comment string**--no-comments**

Use string as a comment string in cleartext signatures and ASCII armored messages or keys (see --armor). The default behavior is not to use a comment string. --comment may be repeated multiple times to get multiple comment strings. --no-comments removes all comments. It is a good idea to keep the length of a single comment below 60 characters to avoid problems with mail programs wrapping such lines.

Note: that comment lines, like all other header lines, are not protected by the signature.

--emit-version**--no-emit-version**

Force inclusion of the version string in ASCII armored output. If given once only the name of the program and the major number is emitted, given twice the minor is also emitted, given thrice the micro is added, and given four times an operating system identification is also emitted. --no-emit-version (default) disables the version line.

```
--sig-notation {name=value}
--cert-notation {name=value}
-N, --set-notation {name=value}
```

Put the name value pair into the signature as notation data. name must consist only of printable characters or spaces, and must contain a "@" character in the form keyname@domain.example.com (substituting the appropriate keyname and domain name, of course). This is to help prevent pollution of the IETF reserved notation namespace. The --expert flag overrides the "@" check. value may be any printable string; it will be encoded in UTF-8, so you should check that your --display-charset is set correctly. If you prefix name with an exclamation mark (!), the notation data will be flagged as critical (rfc4880:5.2.3.16). --sig-notation sets a notation for data signatures. --cert-notation sets a notation for key signatures (certifications). --set-notation sets both.

There are special codes that may be used in notation names. "%k" will be expanded into the key ID of the key being signed, "%K" into the long key ID of the key being signed, "%f" into the fingerprint of the key being signed, "%s" into the key ID of the key making the signature, "%S" into the long key ID of the key making the signature, "%g" into the fingerprint of the key making the signature (which might be a subkey), "%p" into the fingerprint of the primary key of the key making the signature, "%c" into the signature count from the OpenPGP smartcard, and "%%" results in a single "%". %k, %K, and %f are only meaningful when making a key signature (certification), and %c is only meaningful when using the OpenPGP smartcard.

--known-notation name

Adds name to a list of known critical signature notations. The effect of this is that gpg will not mark a signature with a critical signature notation of that name as bad.

Note: that gpg already knows by default about a few critical signatures notation names.

```
--sig-policy-url string
--cert-policy-url string
--set-policy-url string
```

Use string as a Policy URL for signatures (rfc4880:5.2.3.20). If you prefix it with an exclamation mark (!), the policy URL packet will be flagged as critical. --sig-policy-url sets a policy url for data signatures. --cert-policy-url sets a policy url for key signatures (certifications). --set-policy-url sets both.

The same %-expandos used for notation data are available here as well.

--sig-keyserver-url string

Use string as a preferred keyserver URL for data signatures. If you prefix it with an exclamation mark (!), the keyserver URL packet will be flagged as critical.

The same %-expandos used for notation data are available here as well.

--set-filename string

Use string as the filename which is stored inside messages. This overrides the default, which is to use the actual filename of the file being encrypted. Using the empty string for string effectively removes the filename from the output.

```
--for-your-eyes-only
--no-for-your-eyes-only
```

Set the "for your eyes only" flag in the message. This causes GnuPG to refuse to save the file unless the --output option is given, and PGP to use a "secure viewer" with a claimed Tempest-resistant font to display the message. This option overrides --set-filename. --no-for-your-eyes-only disables this option.

```
--use-embedded-filename
--no-use-embedded-filename
```

Try to create a file with a name as embedded in the data. This can be a dangerous option as it enables overwriting files. Defaults to no.

Note: that the option --output overrides this option.

--cipher-algo name

Use name as cipher algorithm. Running the program with the command --version yields a list of supported algorithms. If this is not used the cipher algorithm is selected from the preferences stored with the key. In general, you do not want to use this option as it allows you to violate the OpenPGP standard. --personal-cipher-preferences is the safe way to accomplish the same thing.

--digest-algo name

Use name as the message digest algorithm. Running the program with the command `--version` yields a list of supported algorithms. In general, you do not want to use this option as it allows you to violate the OpenPGP standard. `--personal-digest-preferences` is the safe way to accomplish the same thing.

--compress-algo name

Use compression algorithm name. "zlib" is RFC-1950 ZLIB compression. "zip" is RFC-1951 ZIP compression which is used by PGP. "bzip2" is a more modern compression scheme that can compress some things better than zip or zlib, but at the cost of more memory used during compression and decompression. "uncompressed" or "none" disables compression. If this option is not used, the default behavior is to examine the recipient key preferences to see which algorithms the recipient supports. If all else fails, ZIP is used for maximum compatibility.

ZLIB may give better compression results than ZIP, as the compression window size is not limited to 8k. BZIP2 may give even better compression results than that, but will use a significantly larger amount of memory while compressing and decompressing. This may be significant in low memory situations.

Note: however, that PGP (all versions) only supports ZIP compression. Using any algorithm other than ZIP or "none" will make the message unreadable with PGP. In general, you do not want to use this option as it allows you to violate the OpenPGP standard. `--personal-compress-preferences` is the safe way to accomplish the same thing.

--cert-digest-algo name

Use name as the message digest algorithm used when signing a key. Running the program with the command `--version` yields a list of supported algorithms. Be aware that if you choose an algorithm that GnuPG supports but other OpenPGP implementations do not, then some users will not be able to use the key signatures you make, or quite possibly your entire key.

--disable-cipher-algo name

Never allow the use of name as cipher algorithm. The given name will not be checked so that a later loaded algorithm will still get disabled.

--disable-pubkey-algo name

Never allow the use of name as public key algorithm. The given name will not be checked so that a later loaded algorithm will still get disabled.

--throw-keyids**--no-throw-keyids**

Do not put the recipient key IDs into encrypted messages. This helps to hide the receivers of the message and is a limited countermeasure against traffic analysis. ([Using a little social engineering anyone who is able to decrypt the message can check whether one of the other recipients is the one he suspects.]) On the receiving side, it may slow down the decryption process because all available secret keys must be tried. `--no-throw-keyids` disables this option. This option is essentially the same as using `--hidden-recipient` for all recipients.

--not-dash-escaped

This option changes the behavior of cleartext signatures so that they can be used for patch files. You should not send such an armored file via email because all spaces and line endings are hashed too. You cannot use this option for data which has 5 dashes at the beginning of a line, patch files don't have this. A special armor header line tells GnuPG about this cleartext signature option.

--escape-from-lines**--no-escape-from-lines**

Because some mailers change lines starting with "From" to "> From" it is good to handle such lines in a special way when creating cleartext signatures to prevent the mail system from breaking the signature.

Note: that all other PGP versions do it this way too. Enabled by default. `--no-escape-from-lines` disables this option.

--passphrase-repeat n

Specify how many times gpg will request a new passphrase be repeated. This is useful for helping memorize a passphrase. Defaults to 1 repetition; can be set to 0 to disable any passphrase repetition.

Note: that a n greater than 1 will pop up the pinentry window n+1 times even if a modern pinentry with two entry fields is used.

--passphrase-fd n

Read the passphrase from file descriptor n. Only the first line will be read from file descriptor n. If you use 0 for n, the passphrase will be read from STDIN. This can only be used if only one passphrase is supplied.

Note: that since Version 2.0 this passphrase is only used if the option `--batch` has also been given. Since Version 2.1 the `--pinentry-mode` also needs to be set to `loopback`.

--passphrase-file file

Read the passphrase from file, file. Only the first line will be read from file, file. This can only be used if only one passphrase is supplied. Obviously, a passphrase stored in a file is of questionable security if other users can read this file. Don't use this option if you can avoid it.

Note: that since Version 2.0 this passphrase is only used if the option `--batch` has also been given. Since Version 2.1 the `--pinentry-mode` also needs to be set to `loopback`.

--passphrase string

Use string as the passphrase. This can only be used if only one passphrase is supplied. Obviously, this is of very questionable security on a multi-user system. Don't use this option if you can avoid it.

Note: that since Version 2.0 this passphrase is only used if the option `--batch` has also been given. Since Version 2.1 the `--pinentry-mode` also needs to be set to `loopback`.

--pinentry-mode mode

Set the pinentry mode to mode. Allowed values for mode are:

default Use the default of the agent, which is ask.

ask Force the use of the Pinentry.

cancel Emulate use of Pinentry's cancel button.

error Return a Pinentry error ("No Pinentry").

loopback Redirect Pinentry queries to the caller.

Note: that in contrast to Pinentry the user is not prompted again if he enters a bad password.

--no-symkey-cache

Disable the passphrase cache used for symmetrical encryption and decryption. This cache is based on the message specific salt value (cf. `--s2k-mode`).

--request-origin origin

Tell gpg to assume that the operation ultimately originated at origin. Depending on the origin certain restrictions are applied and the Pinentry may include an extra note on the origin. Supported values for origin are: `local` which is the default, `remote` to indicate a remote origin or `browser` for an operation requested by a web browser.

--command-fd n

This is a replacement for the deprecated shared-memory IPC mode. If this option is enabled, user input on questions is not expected from the TTY but from the given file descriptor. It should be used together with `--status-fd`. See the file `doc/DETAILS` in the source distribution for details on how to use it.

--command-file file

Same as `--command-fd`, except the commands are read out of file, file

--allow-non-selfsigned-uid**--no-allow-non-selfsigned-uid**

Allow the import and use of keys with user IDs which are not self-signed. This is not recommended, as a non-self-signed user ID is trivial to forge.

`--no-allow-non-selfsigned-uid` disables.

--allow-freeform-uid

Disable all checks on the form of the user ID while generating a new one. This option should only be used in very special environments as it does not ensure the de-facto standard format of user IDs.

--ignore-time-conflict

GnuPG normally checks that the timestamps associated with keys and signatures have plausible values. However, sometimes a signature seems to be older than the key due to clock problems. This option makes these checks just a warning. See also *--ignore-valid-from* for timestamp issues on subkeys.

--ignore-valid-from

GnuPG normally does not select and use subkeys created in the future. This option allows the use of such keys and thus exhibits the pre-1.0.7 behaviour. You should not use this option unless there is some clock problem. See also *--ignore-time-conflict* for timestamp issues with signatures.

--ignore-crc-error

The ASCII armor used by OpenPGP is protected by a CRC checksum against transmission errors. Occasionally the CRC gets mangled somewhere on the transmission channel but the actual content (which is protected by the OpenPGP protocol anyway) is still okay. This option allows GnuPG to ignore CRC errors.

--ignore-mdc-error

This option changes a MDC integrity protection failure into a warning. It is required to decrypt old messages which did not use an MDC. It may also be useful if a message is partially garbled, but it is necessary to get as much data as possible out of that garbled message. Be aware that a missing or failed MDC can be an indication of an attack. Use with great caution; see also option *--rfc2440*.

--allow-weak-digest-algos

Signatures made with known-weak digest algorithms are normally rejected with an “invalid digest algorithm” message. This option allows the verification of signatures made with such weak algorithms. MD5 is the only digest algorithm considered weak by default. See also *--weak-digest* to reject other digest algorithms.

--weak-digest name

Treat the specified digest algorithm as weak. Signatures made over weak digests algorithms are normally rejected. This option can be supplied multiple times if multiple algorithms should be considered weak. See also *--allow-weak-digest-algos* to disable rejection of weak digests. MD5 is always considered weak, and does not need to be listed explicitly.

--allow-weak-key-signatures

To avoid a minor risk of collision attacks on third-party key signatures made using SHA-1, those key signatures are considered invalid. This options allows to override this restriction.

--override-compliance-check

The signature verification only allows the use of keys suitable in the current compliance mode. If the compliance mode has been forced by a global option, there might be no way to check certain signature. This option allows to override this and prints an extra warning in such a case. This option is ignored in *--batch* mode so that no accidental unattended verification may happen.

--no-default-keyring

Do not add the default keyring to the list of keyrings.

Note: that GnuPG needs for almost all operations a keyring. Thus if you use this option and do not provide alternate keyrings via *--keyring*, then GnuPG will still use the default keyring.

--no-keyring

Do not use any keyring at all. This overrides the default and all options which specify keyrings.

--skip-verify

Skip the signature verification step. This may be used to make the decryption faster if the signature verification is not needed.

--with-key-data

Print key listings delimited by colons (like *--with-colons*) and print the public key data.

--list-signatures**--list-sigs**

Same as *--list-keys*, but the signatures are listed too. This command has the same effect as using *--list-keys* with *--with-sig-list*.

Note: that in contrast to `--check-signatures` the key signatures are not verified. This command can be used to create a list of signing keys missing in the local keyring; for example:

```
gpg --list-sigs --with-colons USERID | \
awk -F: ' $1=="sig" && $2=="?" {if($13){print $13}else{print $5}}'
```

--fast-list-mode

Changes the output of the list commands to work faster; this is achieved by leaving some parts empty. Some applications don't need the user ID and the trust information given in the listings. By using this options they can get a faster listing. The exact behaviour of this option may change in future versions. If you are missing some information, don't use this option.

--no-literal

This is not for normal use. Use the source to see for what it might be useful.

--set-filesize

This is not for normal use. Use the source to see for what it might be useful.

--show-session-key

Display the session key used for one message. See `--override-session-key` for the counterpart of this option.

We think that Key Escrow is a Bad Thing; however the user should have the freedom to decide whether to go to prison or to reveal the content of one specific message without compromising all messages ever encrypted for one secret key.

You can also use this option if you receive an encrypted message which is abusive or offensive, to prove to the administrators of the messaging system that the ciphertext transmitted corresponds to an inappropriate plaintext so they can take action against the offending user.

--override-session-key string

--override-session-key-fd fd

Don't use the public key but the session key string respective the session key taken from the first line read from file descriptor fd. The format of this string is the same as the one printed by `--show-session-key`. This option is normally not used but comes handy in case someone forces you to reveal the content of an encrypted message; using this option you can do this without handing out the secret key.

Note: that using `--override-session-key` may reveal the session key to all local users via the global process table. Often it is useful to combine this option with `--no-keyring`.

--ask-sig-expire

--no-ask-sig-expire

When making a data signature, prompt for an expiration time. If this option is not specified, the expiration time set via `--default-sig-expire` is used. `--no-ask-sig-expire` disables this option.

--default-sig-expire

The default expiration time to use for signature expiration. Valid values are "0" for no expiration, a number followed by the letter d (for days), w (for weeks), m (for months), or y (for years) (for example "2m" for two months, or "5y" for five years), or an absolute date in the form YYYY-MM-DD. Defaults to "0".

--ask-cert-expire

--no-ask-cert-expire

When making a key signature, prompt for an expiration time. If this option is not specified, the expiration time set via `--default-cert-expire` is used. `--no-ask-cert-expire` disables this option.

--default-cert-expire

The default expiration time to use for key signature expiration. Valid values are "0" for no expiration, a number followed by the letter d (for days), w (for weeks), m (for months), or y (for years) (for example "2m" for two months, or "5y" for five years), or an absolute date in the form YYYY-MM-DD. Defaults to "0".

--default-new-key-algo string

This option can be used to change the default algorithms for key generation. The string is similar to the arguments required for the command `--quick-add-key` but slightly different. For example the current default of `"rsa2048/cert,sign+rsa2048/encr"` (or `"rsa3072"`) can be changed to the value of what we currently call future default, which is `"ed25519/cert,sign+cv25519/encr"`. You need to consult the source code to learn the details.

Note: that the advanced key generation commands can always be used to specify a key algorithm directly.

--force-sign-key

This option modifies the behaviour of the commands `--quick-sign-key`, `--quick-lsign-key`, and the `"sign"` sub-commands of `--edit-key` by forcing the creation of a key signature, even if one already exists.

--forbid-gen-key

This option is intended for use in the global config file to disallow the use of generate key commands. Those commands will then fail with the error code for Not Enabled.

--allow-secret-key-import

This is an obsolete option and is not used anywhere.

--allow-multiple-messages**--no-allow-multiple-messages**

Allow processing of multiple OpenPGP messages contained in a single file or stream. Some programs that call GPG are not prepared to deal with multiple messages being processed together, so this option defaults to no.

Note: that versions of GPG prior to 1.4.7 always allowed multiple messages. Future versions of GnuPG will remove this option.

Warning: Do not use this option unless you need it as a temporary workaround!

--enable-special-filenames

This option enables a mode in which filenames of the form `"-&n"`, where `n` is a non-negative decimal number, refer to the file descriptor `n` and not to a file with that name.

--no-expensive-trust-checks

Experimental use only.

--preserve-permissions

Don't change the permissions of a secret keyring back to user read/write only. Use this option only if you really know what you are doing.

--default-preference-list string

Set the list of default preferences to string. This preference list is used for new keys and becomes the default for `"setpref"` in the edit menu.

--default-keyserver-url name

Set the default keyserver URL to name. This keyserver will be used as the keyserver URL when writing a new self-signature on a key, which includes key generation and changing preferences.

--list-config

Display various internal configuration parameters of GnuPG. This option is intended for external programs that call GnuPG to perform tasks, and is thus not generally useful. See the file `"doc/DETAILS"` in the source distribution for the details of which configuration items may be listed. `--list-config` is only usable with `--with-colons` set.

--list-gcrypt-config

Display various internal configuration parameters of Libgcrypt.

--gpgconf-list

This command is similar to `--list-config` but in general only internally used by the `gpgconf` tool.

--gpgconf-test

This is more or less dummy action. However it parses the configuration file and returns with failure if the configuration file would prevent `gpg` from startup. Thus it may be used to run a syntax check on the configuration file.

Deprecated Options

--show-photos

--no-show-photos

Causes `--list-keys`, `--list-signatures`, `--list-public-keys`, `--list-secret-keys`, and verifying a signature to also display the photo ID attached to the key, if any. See also `-photo-viewer`. These options are deprecated. Use `--list-options [no-]show-photos` and/or `--verify-options [no-]show-photos` instead.

--show-keyring

Display the keyring name at the head of key listings to show which keyring a given key resides on. This option is deprecated: use `--list-options [no-]show-keyring` instead.

--always-trust

Identical to `--trust-model always`. This option is deprecated.

--show-notation

--no-show-notation

Show signature notations in the `--list-signatures` or `--check-signatures` listings as well as when verifying a signature with a notation in it. These options are deprecated. Use `--list-options [no-]show-notation` and/or `--verify-options [no-]show-notation` instead.

--show-policy-url

--no-show-policy-url

Show policy URLs in the `--list-signatures` or `--check-signatures` listings as well as when verifying a signature with a policy URL in it. These options are deprecated. Use `--list-options [no-]show-policy-url` and/or `--verify-options [no-]show-policy-url` instead.

EXAMPLES

```
gpg -se -r Bob
```

file sign and encrypt for user Bob

```
gpg --clear-sign file
```

make a cleartext signature

```
gpg -sb file
```

make a detached signature

```
gpg -u 0x12345678 -sb file
```

make a detached signature with the key 0x12345678

```
gpg --list-keys user_ID
```

show keys

```
gpg --fingerprint user_ID
```

show fingerprint

```
gpg --verify pgpfile
```

```
gpg --verify sigfile [datafile]
```

Verify the signature of the file but do not output the data unless requested. The second form is used for detached signatures, where sigfile is the detached signature (either ASCII armored or binary) and datafile are the signed data; if this is not given, the name of the file holding the signed data is constructed by cutting off the extension (".asc" or ".sig") of sigfile or by asking the user for the filename. If the option `--output` is also used the signed data is written to the file specified by that option; use `-` to write the signed data to stdout.

HOW TO SPECIFY A USER ID

There are different ways to specify a user ID to GnuPG. Some are only valid for gpg others are only good for gpgsm. Here is the entire list of ways to specify a key:

By key Id.

This format is deduced from the length of the string and its content or 0x prefix. The key Id of an X.509 certificate is the low 64 bits of its SHA-1 fingerprint. The use of key Ids is just a shortcut; for all automated processing, the fingerprint should be used.

When using gpg an exclamation mark (!) may be appended to force using the specified primary or secondary key and not to try and calculate which primary or secondary key to use.

The last four lines of the example give the key ID in their long form as internally used by the OpenPGP protocol. You can see the long key ID using the option `--with-colons`.

```
234567C4
0F34E556E
01347A56A
0xAB123456

234AABBCC34567C4
0F323456784E56EAB
01AB3FED1347A5612
0x234AABBCC34567C4
```

By fingerprint.

This format is deduced from the length of the string and its content or the 0x prefix.

Note: that only the 20-byte version fingerprint is available with gpgsm (i.e. the SHA-1 hash of the certificate).

When using gpg an exclamation mark (!) may be appended to force using the specified primary or secondary key and not to try and calculate which primary or secondary key to use.

The best way to specify a key Id is by using the fingerprint. This avoids any ambiguities in case that there are duplicated key IDs.

```
1234343434343434C4343434343434
1234343434343434C3434343434343734349A3434
0E12343434343434343434EAB34843434343434
0xE12343434343434343434EAB34843434343434
```

gpgsm also accepts colons between each pair of hexadecimal digits because this is the de-facto standard on how to present X.509 fingerprints. Gpg also allows the use of the space separated SHA-1 fingerprint as printed by the key listing commands.

By exact match on OpenPGP user ID.

This is denoted by a leading equal sign. It does not make sense for X.509 certificates.

```
=Heinrich Heine <heinrichh@uni-duesseldorf.de>
```

By exact match on an email address.

This is indicated by enclosing the email address in the usual way with left and right angles.

```
<heinrichh@uni-duesseldorf.de>
```

By partial match on an email address.

This is indicated by prefixing the search string with an @. This uses a substring search but considers only the mail address (i.e. inside the angle brackets).

```
@heinrichh
```

By exact match on the subject's DN.

This is indicated by a leading slash, directly followed by the RFC-2253 encoded DN of the subject.

Note: that you can't use the string printed by `gpgsm --list-keys` because that one has been reordered and modified for better readability; use `--with-colons` to print the raw (but standard escaped) RFC-2253 string.

```
/CN=Heinrich Heine,O=Poets,L=Paris,C=FR
```

By exact match on the issuer's DN.

This is indicated by a leading hash mark, directly followed by a slash and then directly followed by the RFC-2253 encoded DN of the issuer. This should return the Root cert of the issuer. See note above.

```
#/CN=Root Cert,O=Poets,L=Paris,C=FR
```

By exact match on serial number and issuer's DN.

This is indicated by a hash mark, followed by the hexadecimal representation of the serial number, then followed by a slash and the RFC-2253 encoded DN of the issuer. See note above.

```
#4F03/CN=Root Cert,O=Poets,L=Paris,C=FR
```

By keygrip.

This is indicated by an ampersand followed by the 40 hex digits of a keygrip. `gpgsm` prints the keygrip when using the command `--dump-cert`.

```
&D75F22C3F86E355877348498CDC92BD21010A480
```

By substring match.

This is the default mode but applications may want to explicitly indicate this by putting the asterisk in front. Match is not case sensitive.

```
Heine
*Heine
```

. and + prefixes

These prefixes are reserved for looking up mails anchored at the end and for a word search mode. They are not yet implemented and using them is undefined.

Please note that we have reused the hash mark identifier which was used in old GnuPG versions to indicate the so called local-id. It is not anymore used and there should be no conflict when used with X.509 stuff.

Using the RFC-2253 format of DNs has the drawback that it is not possible to map them back to the original encoding, however we don't have to do this because our key database stores this encoding as meta data.

FILTER EXPRESSIONS

The options `--import-filter` and `--export-filter` use expressions with this syntax (square brackets indicate an optional part and curly braces a repetition, white space between the elements are allowed):

```
[lc] [{flag}] PROPNAME op VALUE [lc]
```

The name of a property (PROPNAME) may only consist of letters, digits, and underscores. The description for filter type describes which properties are defined. If an undefined property is used, it evaluates to the empty string. Unless otherwise noted, the VALUE must always be given and may not be the empty string. No quoting is defined for the value; thus, the value may not contain the strings `&&` or `||`, which are used as logical connection operators. The flag `--` can be used to remove this restriction.

Numerical values are computed as long int; standard C notation applies. `lc` is the logical connection operator; either `&&` for a conjunction or `||` for a disjunction. A conjunction is assumed at the beginning of an expression. Conjunctions have higher precedence than disjunctions. If VALUE starts with one of the characters used in any operator, a space after the operator is required.

The supported operators (op) are:

```
=~ Substring must match.
!~ Substring must not match.
= The full string must match.
<> The full string must not match.
== The numerical value must match.
!= The numerical value must not match.
<= The numerical value of the field must be LE than the value.
< The numerical value of the field must be LT than the value.
> The numerical value of the field must be GT than the value.
>= The numerical value of the field must be GE than the value.
-le The string value of the field must be less or equal to the value.
-lt The string value of the field must be less than the value.
-gt The string value of the field must be greater than the value.
-ge The string value of the field must be greater or equal to the value.
-n True if the value is not empty (no value allowed).
-z True if the value is empty (no value allowed).
-t Alias for "PROPNAME != 0" (no value allowed).
-f Alias for "PROPNAME == 0" (no value allowed).
```

Values for the flag must be space separated. The supported flags are:

- VALUE spans to the end of the expression.
- c The string match in this part is case-sensitive.
- t Leading and trailing spaces are not removed from VALUE. The optional single space after the operator is here required.

The filter options concatenate several specifications for a filter of the same type. For example, the four options in this example:

```
--import-filter keep-uid="uid =~ Alfa"
--import-filter keep-uid="&& uid !~ Test"
--import-filter keep-uid="|| uid =~ Alpha"
--import-filter keep-uid="uid !~ Test"
```

which is equivalent to

```
--import-filter \
keep-uid="uid =~ Alfa" && uid !~ Test" || uid =~ Alpha" && "uid !~ Test"
```

imports only the user ids of a key containing the strings "Alfa" or "Alpha" but not the string "test".

TRUST VALUES

Trust values indicate owner trust and validity of keys and user IDs. They are displayed with letters or strings:

```
-
unknown      No owner trust assigned / not yet calculated.

e
expired      Trust calculation has failed, probably due to an expired key.

q
undefined, undef  Not enough information for calculation.

n
never        Never trust this key.

m
marginal     Marginally trusted.

f
full         Fully trusted.

u
ultimate     Ultimately trusted.

r
revoked      For validity only: the key or the user ID has been revoked.

?
Err          The program encountered an unknown trust value.
```

FILES

There are a few configuration files to control certain aspects of gpg's operation. Unless noted, they are expected in the current home directory (see: [option --homedir]).

gpg.conf

This is the standard configuration file read by gpg on startup. It may contain any valid long option; the leading two dashes may not be entered, and the option may not be abbreviated. This default name may be changed on the command line (see: [gpg-option --options]). You should backup this file.

Note: that on larger installations, it is useful to put predefined files into the directory `"/usr/local/MacGPG2/etc/skel/.gnupg "` so that newly created users start up with a working configuration. For existing users a small helper script is provided to create these files (see: [addgnupghome]).

For internal purposes gpg creates and maintains a few other files; They all live in the current home directory (see: [option --homedir]). Only the gpg program may modify these files.

~/.gnupg

This is the default home directory used if neither the environment variable GNUPGHOME nor the option --homedir is given.

~/.gnupg/pubring.gpg

The public keyring uses a legacy format. You should backup this file.

If this file is not available, gpg defaults to the new keybox format and creates a file "pubring.kbx" unless that file already exists in which case that file will also be used for OpenPGP keys.

Note: in the case that both files, "pubring.gpg" and "pubring.kbx" exists, but the latter has no OpenPGP keys, the legacy file "pubring.gpg" will be used. Take care: GnuPG versions before 2.1 will always use the file "pubring.gpg" because they do not know the new keybox format. If you have to use GnuPG 1.4 to decrypt archived data, you should keep this file.

~/.gnupg/pubring.gpg.lock

The lock file for the public keyring.

~/.gnupg/pubring.kbx

The public keyring using the new keybox format. This file is shared with gpgsm. You should backup this file. See above for the relation between this file and its predecessor.

To convert an existing "pubring.gpg" file to the keybox format, you first backup the ownertrust values, then rename "pubring.gpg" to "publickeys.backup", so it won't be recognized by any GnuPG version, run import, and finally restore the ownertrust values:

```
$ cd ~/.gnupg
$ gpg --export-ownertrust >otrust.lst
$ mv pubring.gpg publickeys.backup
$ gpg --import-options restore --import publickeys.backups
$ gpg --import-ownertrust otrust.lst
```

~/.gnupg/pubring.kbx.lock

The lock file for "pubring.kbx".

~/.gnupg/secring.gpg

The legacy secret keyring as used by GnuPG versions before 2.1. It is not used by GnuPG 2.1 and later. You may want to keep it in case you have to use GnuPG 1.4 to decrypt archived data.

~/.gnupg/secring.gpg.lock

The lock file for the legacy secret keyring.

~/.gnupg/.gpg-v21-migrated

File indicating that a migration to GnuPG 2.1 has been done.

~/.gnupg/trustdb.gpg

The trust database. There is no need to backup this file; it is better to backup the ownertrust values (see: [option --export-ownertrust]).

~/.gnupg/trustdb.gpg.lock

The lock file for the trust database.

~/.gnupg/random_seed

A file used to preserve the state of the internal random pool.

~/.gnupg/openpgp-revocs.d/

This is the directory where gpg stores pre-generated revocation certificates. The file name corresponds to the OpenPGP fingerprint of the respective key. It is suggested to backup those certificates and if the primary private key is not stored on the disk to move them to an external storage device. Anyone who can access these files is able to revoke the corresponding key. You may want to print them out. You should backup all files in this directory and take care to keep this backup closed away.

Operation is further controlled by a few environment variables:

HOME

Used to locate the default home directory.

GNUPGHOME

If set directory used instead of "~/.gnupg".

GPG_AGENT_INFO

This variable is obsolete; it was used by GnuPG versions before 2.1.

PINENTRY_USER_DATA

This value is passed via gpg-agent to pinentry. It is useful to convey extra information to a custom pinentry.

COLUMNS**LINES**

Used to size some displays to the full size of the screen.

LANGUAGE

Apart from its use by GNU, it is used in the W32 version to override the language selection done through the Registry. If used and set to a valid and available language name (langid), the file with the translation is loaded from gpgdir/gnupg.nls/langid.mo. Here gpgdir is the directory out of which the gpg binary has been loaded. If it can't be loaded the Registry is tried and as last resort the native Windows locale system is used.

GNUPG_BUILD_ROOT

This variable is only used by the regression test suite as a helper under operating systems without proper support to figure out the name of a process' text file.

GNUPG_EXEC_DEBUG_FLAGS

This variable allows to enable diagnostics for process management. A numeric decimal value is expected. Bit 0 enables general diagnostics, bit 1 enables certain warnings on Windows.

When calling the gpg-agent component gpg sends a set of environment variables to gpg-agent. The names of these variables can be listed using the command:

```
gpg-connect-agent 'getinfo std_env_names' /bye | awk ' $1=="D" {print $2}'
```


BUGS

On older systems, this program should be installed as `setuid(root)`. This is necessary to lock memory pages. Locking memory pages prevents the operating system from writing memory pages (which may contain passphrases or other sensitive material) to disk. If you get no warning message about insecure memory, your operating system supports locking without being root. The program drops root privileges as soon as locked memory is allocated.

Note: also that some systems (especially laptops) have the ability to “suspend to disk” (also known as “safe sleep” or “hibernate”). This writes all memory to disk before going into a low-power or even powered-off mode. Unless measures are taken in the operating system to protect the saved memory, passphrases or other sensitive material may be recoverable from it later.

Before you report a bug, you should first search the mailing list archives for similar problems and second check whether such a bug has already been reported to our bug tracker at <https://bugs.gnupg.org>.

SEE ALSO

gpgv(1), gpgsm(1), gpg-agent(1)

The full documentation for this tool is maintained as a Texinfo manual. If GnuPG and the info program are properly installed at your site, the command ***info gnupg*** should give you access to the complete manual, including a menu structure and an index.

APPENDIX B

Recovering your lab environment. Use the following steps to recreate the lab environment from the ***Sending Encrypted Messages*** lab.

Recovery Task 1: The Command-Line Interface (CLI)

This recovery assumes that you are already at your operating systems command prompt. If you do not know how to get there, refer to Task 1 earlier in this lab

The Windows Command Prompt

1. Create a folder for your encryption keys by typing “md student@example.com-keys”

```
C:\Users\username> md student@example.com-keys
```

2. Change into your ***student@example.com-keys*** folder by typing “cd student@example.com-keys,” and your command prompt will change to your directory name.

```
C:\Users\username> cd student@example.com-keys  
C:\Users\username\student@example.com-keys>
```

NOTE: In these examples, ensure that you replace ***student@example.com*** with your e-mail address.

The macOS Terminal

1. Create a folder for your encryption keys by typing “mkdir student@example.com-keys.”

```
computername:~ username$ mkdir student@example.com-keys
```

2. Change into your ***student@example.com-keys*** folder by typing “cd student@example.com-keys,” and your command prompt will change to your directory name.

```
computername:~ username$ cd student@example.com-keys  
computername:student@example.com username$
```

NOTE: In these examples, ensure that you replace ***student@example.com*** with your e-mail address.

Recovery Task 2: Import your Keys

You will import your public and private key pair from the backup you created in the ***Sending Encrypted Messages*** lab, then reimport your instructor's public key from the OpenPGP key server into your gpg key chain.

1. From the command prompt, list all of the keys in the key-chain by typing:

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

The following step may have already been carried out in a previous lab. To ensure that your public/private key pair already exists, scrutinize the output of the previous command to ensure you will not overwrite your public /private key pair.

Importing your Public/Private Key Pair

Import your public/private key pair from the backup using the **gpg** command line.

1. The **gpg** command has two options for importing a public/private key pair. The below command will allow us to import our key pair from the backup.

```
gpg --pinentry-mode=loopback --import <key name>.asc5
```

Use the following two options with the **gpg** command to import our public/private key pair.

```
--import <key name>.asc
```

Allows us to import the key from the filename specified.

```
--pinentry-mode=loopback
```

Ensures that the passphrase for your private key stays confined to the command line without explicitly calling a GUI agent.

```
gpg --pinentry-mode=loopback --import public-key.asc
```

```
gpg: key 3B8F1D139D70141E: "Lab Student <student@example.com>" imported
gpg: Total number processed: 1
gpg:             imported: 1
```

```
gpg --pinentry-mode=loopback --import private-key.asc
```

```
gpg: key 3B8F1D139D70141E: "Lab Student <student@example.com>" not changed
gpg: key 3B8F1D139D70141E: secret key imported
gpg: Total number processed: 1
gpg:             unchanged: 1
gpg:             secret keys read: 1
gpg:             secret keys imported: 1
```

⁵ When prompted, enter the password for your private key.

Importing Your Instructors Public Key

You will now import your instructor's public key from one of the following two key servers, keys.openpgp.org or pgp.mit.edu.

12. The **gpg** command has two options for importing your instructor's public key from the key server using your instructor's e-mail address.

```
gpg --keyserver <keyserver> --search <email address>
```

Use the following two options with the **gpg** command to retrieve your instructors public key from the key server.

```
--keyserver <keyserver>
```

Allows us to choose the name of the key server where we will upload our public key.

```
--search <email address>
```

Allows us to search the public key server for the public key associated with an email address.

```
gpg --keyserver htps://keys.openpgp.org --search mkisow@ccac.edu
gpg: data source: https://keys.openpgp.org:443
(1)  Matthew R. Kisow <mkisow@ccac.edu>
      4096 bit RSA key 007A2EFE19C015A2, created: 2022-01-30
Keys 1-1 of 1 for "mkisow@ccac.edu".  Enter number(s), N)ext, or Q)uit > 1
gpg: key 007A2EFE19C015A2: public key " Matthew R. Kisow mkisow@ccac.edu imported
gpg: Total number processed: 1
gpg:          imported: 1
```

13. Select and import the key into your keychain by choosing the number in front of it.

Listing the Keys

You will now review your key chain to ensure that the lab environment was how you left it after the previous lab.

1. Using the **gpg** command, you will list all keys in your key chain.

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

Use the following options with the **gpg** command to list the public and private keys in your key chain.

```
--list-keys
Allows us to list ALL of the public keys in our key chain.

--list-secret-keys
Allows us to list only the private keys in our key chain.
```

```
gpg --list-keys
/Users/labstudent/.gnupg/pubring.kbx
-----
pub   rsa4096 2022-09-27 [SC]
      C56EA6745BF05DF4313A29DD3B8F1D139D70141E
uid   [ultimate] Lab Student <student@example.com>
sub   rsa4096 2022-09-27 [E]

pub   rsa4096 2022-01-30 [SC]
      38752E510AEFF60B2CA4A5E7007A2EFE19C015A2
uid   [ unknown] Matthew R. Kisow <mkisow@ccac.edu>
sub   rsa4096 2022-01-30 [E]

gpg --list-secret-keys
/Users/labstudent/.gnupg/pubring.kbx
-----
pub   rsa4096 2022-09-27 [SC]
      C56EA6745BF05DF4313A29DD3B8F1D139D70141E
uid   [ultimate] Lab Student <student@example.com>
sub   rsa4096 2022-09-27 [E]
```

The selection in red is similar to what your public/private key pair will look like in the keychain. The entry with your instructor's email should look similar as well.

When your recovery is complete, you may continue where you left off with this lab.