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 or 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 a 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: With option **--batch** he verification of signatures stops at the first bad signature. This is a safe default for unattended processing but sometimes a status for all signatures is needed. To override this early bailout use the option **--proc-all-sigs**.

Note: To check whether a file was signed by a certain key the option **--assert-signer** can be used. As an alternative the **gpgv** tool can be used. **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 comain 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 and the methods defined by --auto-key-locate define LDAP servers, the key is fetched from these resources; defined non-LDAP keyservers are skipped.

--show-keys

This commands 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 . Please note that the command "openpgp" can be used to switch to the OpenPGP application of cards which by default are presenting another application (e.g. PIV).

--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: Keyservers 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 can not 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|16|30 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. The special level 16 uses a quality level of 1 and outputs an endless stream of hex-encoded octets. The special level 30 outputs random as 30 zBase-32 characters.

--gen-prime *mode bits*

Use the source, Luke :-). The output format is subject to change with any 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. The **--dearmor** command can also be used to dearmor PEM armors.

--unwrap
This option modifies the command **--decrypt** to output the original message with the encryption layer removed. Thus the output will be an OpenPGP data structure which often means a signed OpenPGP message. Note that this option may or may not remove a compression layer which is often found beneath the encryption layer.

--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 a 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 one 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.

--quick-add-adsk *fpr adskfpr*

Directly add an Additional Decryption Subkey to the key identified by the fingerprint *fpr*. *adskfpr* is the fingerprint of another key’s encryption subkey. A subkey is commonly used here because by default a primary key has no encryption capability. Use the option **--with-subkey-fingerprint** with a list command to display the subkey fingerprints. If the string "default" is used for *adskfpr* all missing ADSKs configured with **--default-new-key-adsk** are added.

--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 send 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 useful in distinct communities or groups to implement the concept of a Trusted Introducer. 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. Also 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. Note that an unattended version of this command is available as **--quick-update-pref**.

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.

keytotpm
Transfer the selected secret subkey (or the primary key if no subkey has been selected) to TPM form. The secret key in the keyring will be replaced by the TPM representation of that key, which can only be read by the particular TPM that created it (so the keyfile now becomes locked to the laptop containing the TPM). Only certain key types may be transferred to the TPM (all TPM 2.0 systems are mandated to have the rsa2048 and nistp256 algorithms but newer TPMs may have more). Note that the key itself is not transferred into the TPM, merely encrypted by the TPM in-place, so if the keyfile is deleted, the key will be lost. Once transferred to TPM representation, the key file can never be converted back to non-TPM form and the key will die when the TPM does, so you should first have a backup on secure offline

storage of the actual secret key file before conversion. It is essential to use the physical system TPM that you have rw permission on the TPM resource manager device (/dev/tpmrm0). Usually this means you must be a member of the tss group.

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**. Also 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 can not 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).

addadsk

Add an Additional Decryption Subkey. The user is asked to enter the fingerprint of another encryption subkey. Note that the exact fingerprint of another key’s encryption subkey needs to be entered. This is because commonly the primary key has no encryption capability. Use

the option **--with-subkey-fingerprint** with a list command to display the subkey fingerprints.

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-key**.

--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 sub-command **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.

--quick-update-pref *user-id*

This command updates the preference list of the key to the current default value (either built-in or set via **--default-preference-list**). This is the unattended version of using "setpref" in the **--key-edit** menu without giving a list. Note that you can show the preferences in a key listing by using **--list-options show-pref** or **--list-options show-pref-verbose**. You should also re-distribute updated keys to your peers.

--quick-set-ownertrust *user-id value*

This command sets the ownertrust of a key and can also be used to set the disable flag of a key. This is the unattended version of using "trust", "disable", or "enable" in the **--key-edit** menu.

--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. It is suggested to use a fingerprint or at least a long keyID for *name*. 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 and it is suggested to use a fingerprint for *name*.

--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.

-v, --verbose
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.

-q, --quiet
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.

--proc-all-sigs
This option overrides the behaviour of the **--batch** option to stop signature verification at the first bad signatures.

--list-filter {select=*expr***}**
A list filter can be used to output only certain keys during key listing commands. For the available property names, see the description of **--import-filter**.

--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-ownertrust
Show the ownertrust value for keys also in the standard key listing. Defaults to no.

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-unusable-sigs
Show key signature made using weak or unsupported algorithms.

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.

sort-sigs
With **--list-sigs** and **--check-sigs** sort the signatures by keyID and creation time to make it easier to view the history of these signatures. The self-signature is also listed before other signatures. Defaults to yes. This option has no effect in **-with-colons** mode.

--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.

--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.

Note that if the option **use-keyboxd** is enabled in '*common.conf*', no keyrings are used at all and keys are all maintained by the keyboxd process in its own database.

--primary-keyring *file*

This is a varian 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.

--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*
--no-compress
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.

Option **-z** sets both. A value of 0 for *n* disables compression. A value of -1 forces compression using the default level. Option **--no-compress** is identical to **-z0**.

Except for the **--store** command compression is always used unless **gpg** detects that the input is already compressed. To inhibit the use of compression use **-z0** or **--no-compress**; to force compression use **-z-1** or option **z** with another compression level than the default as indicated by -1. Note that this overriding of the default deection works only with **z** and not with the long variant of this option.

--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. The value "none" is explicitly allowed to distinguish between the use of any trusted-key option and no use of this option at all (e.g. due to the **--no-options** option).

--add-desig-revoker [sensitive:]*fingerprint*
Add the key specified by *fingerprint* as a designated revoker to newly created keys. If the

fingerprint is prefixed with the keyword ‘‘sensitive:’’ that info is normally not exported with the key. This option may be given several times to add more than one designated revoker. If the keyword ‘‘clear’’ is used instead of a fingerprint, all previously fiven fingerprints are discarded. Designated revokers are marked on the key as non-revocable. Note that a designated revoker specified using a parameter file will also be added to the key.

--default-new-key-adsk *fingerprint*

Add the subkey specified by *fingerprint* as an Additional Decryption Subkey (ADSK) to newly created keys. This option may be given several time to add more than one ADSK. It is also possible to give several fingerprints delimited by space or comma as value to this option. If the keyword ‘‘clear’’ is used instead of a fingerprint, all previously specified fingerprints are discarded (useful to override options given in a config file). The fingerprint is expected to specify a subkey and it does not need an exclamation mark as suffix; it must be given in cmpact format (40 or 64 hex-digits without any spaces).

--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 experimental 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 experimental 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.

--always-trust
Identical to **--trust-model always**.

--assert-signer *fpr_or_file*
This option checks whether at least one valid signature on a file has been made with the specified key. The key is either specified as a fingerprint or a file listing fingerprints. The fingerprint must be given or listed in compact format (no colons or spaces in between). This option can be given multiple times and each fingerprint is checked against the signing key as well as the corresponding primary key. If *fpr_or_file* specifies a file, empty lines are ignored as well as all lines starting with a hash sign. With this option gpg is guaranteed to return with an exit code of 0 if and only if a signature has been encountered, is valid, and the key matches one of the fingerprints given by this option.

--assert-pubkey-algo *algolist*
During data signature verification this options checks whether the used public key algorithm matches the algorithms given by *algolist*. This option can be given multiple times to concatenate more algorithms to the list; the delimiter of the list are either commas or spaces.

The algorithm names given in the list may either be verbatim names like "ed25519" with an optional leading single equal sign, or being prefixed with ">", ">=", "<=", or "<". That prefix operator is applied to the number part of the algorithm name; for example 2048 in "rsa2048" or 384 in "brainpoolP384r1". If the the leading non-digits in the name matches, the prefix operator is used to compare the number part, a trailing suffix is ignored in this case. For example an algorithm list ">rsa3000, >=brainpool384r1, =ed25519" allows RSA signatures with more that 3000 bits, Brainpool curves 384 and 512, and the ed25519 algorithm.

With this option gpg (and also gpgv) is guaranteed to return with an exit code of 0 if and only if all valid signatures on data are made using a matching algorithm from the given list.

--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.

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

Locate the key using the configured LDAP servers. This method is similar to the **keyserver** mechanism but always uses only LDAP servers.

ntds
Locate the key using the Active Directory (Windows only). This method also allows one to search by fingerprint using the command **--locate-external-key**. Note that this mechanism is actually a shortcut for the mechanism ‘ldap’ using only "ldap:///" as the keyserver.

keyserver
Locate a key using a keyserver. This method also allows one to search by fingerprint using the command **--locate-external-key** if any of the configured keyservers 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 one 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 the 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 signatures 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 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 keyservers differentiate between revoked and unrevoked keys, and for such keyservers this option is meaningless. Note also that most keyservers 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 keyservers.

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 keys is refreshed. Thus this option is not enabled by default.

include-subkeys
When receiving a key, include subkeys as potential targets. Note that this option is not used with HKP keyservers, as they do not support retrieving keys by subkey id.

only-pubkeys
Do now allow to import secret keys.

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, repair-keys, repair-pks-subkey-bug, export-attributes". 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 '*/usr/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". It is suggested to use a fingerprint or at least a long keyID for *name*. 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". It is suggested to use a fingerprint or at least a long keyID for *name*. 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 containing a proper mail address or just a plain mail address. The option can be given multiple times.

When creating a signature this option tells gpg the signing key's user id used to make the signature and embeds that user ID into the created signature (using OpenPGP's ''Signer's User ID'' subpacket). If the option is given multiple times a suitable user ID is picked. However, if the signing key was specified directly by using a mail address (i.e. not by using a fingerprint or key ID) this option is used and the mail address is embedded in the created signature.

When verifying a signature *mbox* is used to restrict the information printed by the TOFU code to matching user IDs. If the option is used and the signature contains a ''Signer's User ID'' subpacket that information is is also used to restrict the printed information. Note that GnuPG considers only the mail address part of a User ID.

If this option or the said subpacket is available the TRUST lines as printed by option **status-fd** correspond to the corresponding User ID; if no User ID is known the TRUST lines are computed directly on the key and do not give any information about the User ID. In the latter case it his highly recommended to scripts and other frontends to evaluate the VALIDSIG line, retrieve the key and print all User IDs along with their validity (trust) information.

--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".

--chunk-size *n*

The AEAD encryption mode encrypts the data in chunks so that a receiving side can check for transmission errors or tampering at the end of each chunk and does not need to delay this until all data has been received. The used chunk size is 2^*n* byte. The lowest allowed value for *n* is 6 (64 byte) and the largest is the default of 22 which creates chunks not larger than 4 MiB.

--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 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 ownertrust 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 ownertrust 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 ownertrust 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 option **export-dane** affect the output. This option can for example 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.

ignore-attributes
Ignore all attribute user IDs (photo IDs) and their signatures while importing a key.

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

bulk-import
When used the keyboxd (option **use-keyboxd** in '*common.conf*') does the import within a single transaction.

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.

select
This filter is only implemented by **--list-filter**. All property names may be used.

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. Note that all property names may also be used by **--list-filter**.

Property names may be prefix with a scope delimited by a slash. Valid scopes are "pub" for public and secret primary keys, "sub" for public and secret subkeys, "uid" for for user-ID packets, and "sig" for signature packets. Invalid scopes are currently ignored.

The available properties are:

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

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

algostr
A string with the key algorithm description. For example "rsa3072" or "ed25519".

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

key_size
A number with the effective key size 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)

key_expires
key_expires_d
The expiration time of a public key or subkey or 0 if it does not expire. The second is the same but given as an ISO date string or an empty string e.g. "2038-01-19".

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.

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_expires
sig_expires_d
The expiration time of a signature packet or 0 if it does not expire. The second is the same but given as an ISO date string or an empty string e.g. "2038-01-19".

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)

origin
A string with the key origin or a question mark. For example the string ‘‘wkd’’ is used if a

key originated from a Web Key Directory lookup.

lastupd
The timestamp the key was last updated from a keyserver or the Web Key Directory.

url A string with the the URL associated wit the last key lookup.

--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-revocs
Export only standalone revocation certificates of the key. This option does not export revocations of 3rd party certificate revocations.

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.

mode1003
Enable the use of a new secret key export format. This format avoids the re-encryption as required with the current OpenPGP format and also improves the security of the secret key if it has been protected with a passphrase. Note that an unprotected key is exported as-is and thus not secure; the general rule to convey secret keys in an OpenPGP encrypted file still applies with this mode. Versions of GnuPG before 2.4.0 are not able to import such a secret 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-v5-fingerprint

In a colon mode listing emit "fp2" lines for version 4 OpenPGP keys having a v5 style fingerprint of the key.

--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

--force-ocb
--force-aead
Force the use of AEAD encryption over MDC encryption. AEAD is a modern and faster way to do authenticated encryption than the old MDC method. **--force-aead** is an alias and deprecated. See also option **--chunk-size**.

--force-mdc
--disable-mdc
These options are obsolete and have no effect since GnuPG 2.2.8. The MDC is always used unless the keys indicate that an AEAD algorithm can be used in which case AEAD is used. But 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
--no-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 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** and allows the recipient of a signed message to reply encrypted to the sender without using any online directories to lookup the key. The default is **--no-include-key-block**. See also the option **--auto-key-import**.

--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 extension from the proposed update to OpenPGP and 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. This option implies **--allow-old-cipher-algos**. 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. This option implies **--allow-old-cipher-algos**. Note that this is currently the same thing as **--openpgp**.

--rfc4880bis

Reset all packet, cipher and digest options to strict according to the proposed updates of RFC-4880.

--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. This option implies **--allow-old-cipher-algos**. See also option **--ignore-mdc-error**.

--pgp6

This option is obsolete; it is handled as an alias for **--pgp7**

--pgp7

Set up all options to be as PGP 7 compliant as possible. This allowed the ciphers IDEA, 3DES, CAST5,AES128, AES192, AES256, and TWOFISH., the hashes MD5, SHA1 and RIPEMD160, and the compression algorithms none and ZIP. This option implies **--escape-from-lines** and disables **--throw-keyids**,

--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 is 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.

--compatibility-flags *flags*

Set compatibility flags to work around problems due to non-compliant keys or data. The *flags* are given as a comma separated list of flag names and are OR-ed together. The special flag "none" clears the list and allows one to start over with an empty list. To get a list of available flags the sole word "help" can be used.

--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 instead of the keyword.

basic

Some basic debug messages. A value between 1 and 2 may be used instead of the keyword.

advanced

More 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 debug 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. This option is only useful for debugging and the behavior may change at any time without notice.

--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.

--debug-set-iobuf-size *n*
Change the buffer size of the IOBUFs to *n* kilobyte. Using 0 prints the current size. Note well: This is a maintainer only option and may thus be changed or removed at any time without notice.

--debug-allow-large-chunks
To facilitate software tests and experiments this option allows one to specify a limit of up to 4 EiB (**--chunk-size 62**).

--debug-ignore-expiration
This option tries to override certain key expiration dates. It is only useful for certain regression tests.

--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.

--full-timestrings
Change the format of printed creation and expiration times from just the date to the date and time. This is in general not useful and the same information is anyway available in **--with-colons** mode. These longer strings are also not well aligned with other printed data.

--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 s socket.

--log-time
Prefix all log output with a timestamp even if no log file is 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 by giving the sender control on how to store files. Defaults to no. Note that the option **--output** overrides this option.

A better approach than using this option is to decrypt to a temporary filename and then rename that file to the embedded file name after checking that the embedded filename is harmless. When using

the **--status-fd** option gpg tells the filename as part of the PLAINTEXT status message. If the filename is important, the use of **gpgtar** is another option because gpgtar will never overwrite a file but decrypt the files to a new directory.

Note also that unless a modern version 5 signature is used the embedded filename is not part of the signed data.

--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. The option **--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. The option **--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. The option **--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. Note also that a public key algorithm must be compatible with the specified digest algorithm; thus selecting an arbitrary digest algorithm may result in error messages from lower crypto layers or lead to security flaws.

--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 can not 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 en- 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-old-cipher-algos

Old cipher algorithms like 3DES, IDEA, or CAST5 encrypt data using blocks of 64 bits; modern algorithms use blocks of 128 bit instead. To avoid certain attack on these old algorithms it is suggested not to encrypt more than 150 MiByte using the same key. For this reason gpg does not allow the use of 64 bit block size algorithms for encryption unless this option is specified.

--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 one to override this restriction.

--override-compliance-check
This was a temporary introduced option and has no more effect.

--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.

Note that if the option **use-keyboxd** is enabled in ‘*common.conf*’, no keyrings are used at all and keys are all maintained by the keyboxd process in its own database.

--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.

--no-auto-trust-new-key

When creating a new key the ownertrust of the new key is set to ultimate. This option disables this and the user needs to manually assign an ownertrust value.

--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

These are obsolete options; they have no more effect since GnuPG 2.2.8.

--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-key** 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.

--chuid *uid*

Change the current user to *uid* which may either be a number or a name. This can be used from the root account to run gpg for another user. If *uid* is not the current UID a standard PATH is set and the envvar GNUPGHOME is unset. To override the latter the option **--homedir** can be used. This option has only an effect when used on the command line. This option has currently no effect at all on Windows.

Deprecated 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 was useful when communicating between two platforms with different line ending conventions (UNIX-like to Mac, Mac to Windows, etc). **--no-textmode** disables this option, and is the default. Note that this is a legacy option which should not anymore be used by any modern software.

--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.

--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.

--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.

--personal-aead-preferences *string*
This option is deprecated and has no more effect since version 2.3.9.

--aead-algo *name*
This option is deprecated and has no more effect since version 2.3.9.

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 of them 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 are 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.

1234343434343434C434343434343434
123434343434343C34343434343434734349A3434
0E123434343434343434EAB3484343434343434
0xE123434343434343434EAB3484343434343434

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 the 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 begin of an expression. Conjunctions have higher precedence than disjunctions. If *VALUE* starts with one of the characters used in any *op* a space after the *op* 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 than 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 than the value.

-n True if value is not empty (no value allowed).

-z True if value is empty (no value allowed).

-t Alias for "PROPNAME != 0" (no value allowed).

-f Alias for "PROPNAME == 0" (no value allowed).

Values for *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 done case-sensitive.

-t Leading and trailing spaces are not removed from *VALUE*. The optional single space after *op* 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 are used to indicate ownertrust and validity of keys and user IDs. They are displayed with letters or strings:

-
unknown
No ownertrust 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.

common.conf
This is an optional configuration file read by **gpg** on startup. It may contain options pertaining to all components of GnuPG. Its current main use is for the "use-keyboxd" option. If the default home directory '*~/.gnupg*' does not exist, GnuPG creates this directory and a '*common.conf*' file with "use-keyboxd".

Note that on larger installations, it is useful to put predefined files into the directory '*/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 which is used if neither the environment variable **GNUPGHOME** nor the option **--homedir** is given.

~/.gnupg/pubring.gpg
The public keyring using 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 that 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 about the new keybox format. In the case that 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 it 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.backup
$ 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 one 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}'
```

NOTES

gpg is often used as a backend engine by other software. To help with this a machine interface has been defined to have an unambiguous way to do this. The options **--status-fd** and **--batch** are almost always required for this.

Programmatic use of GnuPG

Please consider using GPGME instead of calling **gpg** directly. GPGME offers a stable, backend-independent interface for many cryptographic operations. It supports OpenPGP and S/MIME, and also allows interaction with various GnuPG components.

GPGME provides a C-API, and comes with bindings for C++, Qt, and Python. Bindings for other languages are available.

Ephemeral home directories

Sometimes you want to contain effects of some operation, for example you want to import a key to inspect it, but you do not want this key to be added to your keyring. In earlier versions of GnuPG, it was possible to specify alternate keyring files for both public and secret keys. In modern GnuPG versions, however, we changed how secret keys are stored in order to better protect secret key material, and it was not possible to preserve this interface.

The preferred way to do this is to use ephemeral home directories. This technique works across all versions of GnuPG.

Create a temporary directory, create (or copy) a configuration that meets your needs, make **gpg** use this directory either using the environment variable *GNUPGHOME*, or the option **--homedir**. GPGME supports this too on a per-context basis, by modifying the engine info of contexts. Now execute whatever operation you like, import and export key material as necessary. Once finished, you can delete the directory. All GnuPG backend services that were started will detect this and shut down.

The quick key manipulation interface

Recent versions of GnuPG have an interface to manipulate keys without using the interactive command **--edit-key**. This interface was added mainly for the benefit of GPGME (please consider using GPGME, see the manual subsection ‘‘Programmatic use of GnuPG’’). This interface is described in the subsection ‘‘How to manage your keys’’.

Unattended key generation

The command **--generate-key** may be used along with the option **--batch** for unattended key generation. This is the most flexible way of generating keys, but it is also the most complex one. Consider using the quick key manipulation interface described in the previous subsection ‘‘The quick key manipulation interface’’.

The parameters for the key are either read from stdin or given as a file on the command line. The format of the parameter file is as follows: Text only, line length is limited to about 1000 characters.

UTF-8 encoding must be used to specify non-ASCII characters. Empty lines are ignored. Leading and trailing white space is ignored. A hash sign as the first non white space character indicates a comment line. Control statements are indicated by a leading percent sign, their arguments are separated by white space from the keyword. Parameters are specified by a keyword, followed by a colon; arguments are separated by white space. The first parameter must be ‘Key-Type’ but control statements may be placed anywhere. The order of the parameters does not matter except for ‘Key-Type’. The parameters are only used for the generated keyblock (primary and subkeys); parameters from previous sets are not used. Some syntax checks may be performed. Key commences when either the end of the parameter file is reached, the next ‘Key-Type’ parameter is encountered, or the control statement ‘%commit’ is encountered.

Control statements:

%echo *text*
Print *text* as diagnostic.

%dry-run
Suppress actual key generation (useful for syntax checking).

%commit
Perform the key generation. Note that an implicit commit is done at the next parameter.

%pubring *filename*
Do not write the key to the default or commandline given keyring but to *filename*. This must be given before the first commit to take place, duplicate specification of the same filename is ignored, the last filename before a commit is used. The filename is used until a new filename is used (at commit points) and all keys are written to that file. If a new filename is given, this file is created (and overwrites an existing one).

See the previous subsection ‘‘Ephemeral home directories’’ for a more robust way to contain side-effects.

%secring *filename*

This option is a no-op for GnuPG 2.1 and later.

See the previous subsection ‘‘Ephemeral home directories’’.

%ask-passphrase
%no-ask-passphrase
This option is a no-op since GnuPG version 2.1.

%no-protection
Using this option allows the creation of keys without any passphrase protection. This option is mainly intended for regression tests.

%transient-key
If given the keys are created using a faster and a somewhat less secure random number generator. This option may be used for keys which are only used for a short time and do not require full cryptographic strength. It takes only effect if used together with the control statement ‘%no-protection’.

General Parameters:

Key-Type: *algo*
Starts a new parameter block by giving the type of the primary key. The algorithm must be capable of signing. This is a required parameter. *algo* may either be an OpenPGP algorithm number or a string with the algorithm name. The special value ‘default’ may be used for *algo* to create the default key type; in this case a ‘Key-Usage’ shall not be given and ‘default’ also be used for ‘Subkey-Type’.

Key-Length: *nbits*
The requested length of the generated key in bits. The default is returned by running the command ‘gpg --gpgconf-list’. For ECC keys this parameter is ignored.

Key-Curve: *curve*
The requested elliptic curve of the generated key. This is a required parameter for ECC keys. It is ignored for non-ECC keys.

Key-Grip: *hexstring*
This is optional and used to generate a CSR or certificate for an already existing key. Key-Length will be ignored when given.

Key-Usage: *usage-list*
Space or comma delimited list of key usages. Allowed values are ‘encrypt’, ‘sign’, and ‘auth’. This is used to generate the key flags. Please make sure that the algorithm is capable of this usage. Note that OpenPGP requires that all primary keys are capable of certification, so no matter what usage is given here, the ‘cert’ flag will be on. If no ‘Key-Usage’ is specified and the ‘Key-Type’ is not ‘default’, all allowed usages for that particular algorithm are used; if it is not given but ‘default’ is used the usage will be ‘sign’.

Subkey-Type: *algo*
This generates a secondary key (subkey). Currently only one subkey can be handled. See also ‘Key-Type’ above.

Subkey-Length: *nbits*
Length of the secondary key (subkey) in bits. The default is returned by running the command ‘gpg --gpgconf-list’.

Subkey-Curve: *curve*
Key curve for a subkey; similar to ‘Key-Curve’.

Subkey-Usage: *usage-list*
Key usage lists for a subkey; similar to ‘Key-Usage’.

Passphrase: *string*
If you want to specify a passphrase for the secret key, enter it here. Default is to use the Pinentry dialog to ask for a passphrase.

Name-Real: *name*
Name-Comment: *comment*
Name-Email: *email*

The three parts of a user name. Remember to use UTF-8 encoding here. If you don't give any of them, no user ID is created.

Expire-Date: *iso-date*|**(***number***[d|w|m|y])**

Set the expiration date for the key (and the subkey). It may either be entered in ISO date format (e.g. "20000815T145012") or as number of days, weeks, month or years after the creation date. The special notation "seconds=N" is also allowed to specify a number of seconds since creation. Without a letter days are assumed. Note that there is no check done on the overflow of the type used by OpenPGP for timestamps. Thus you better make sure that the given value make sense. Although OpenPGP works with time intervals, GnuPG uses an absolute value internally and thus the last year we can represent is 2105.

Creation-Date: *iso-date*

Set the creation date of the key as stored in the key information and which is also part of the fingerprint calculation. Either a date like "1986-04-26" or a full timestamp like "19860426T042640" may be used. The time is considered to be UTC. The special notation "seconds=N" may be used to directly specify a the number of seconds since Epoch (Unix time). If it is not given the current time is used.

Preferences: *string*

Set the cipher, hash, and compression preference values for this key. This expects the same type of string as the sub-command 'setpref' in the **--edit-key** menu.

Revoker: *algo***:***fpr* **[sensitive]**

Add a designated revoker to the generated key. Algo is the public key algorithm of the designated revoker (i.e. RSA=1, DSA=17, etc.) *fpr* is the fingerprint of the designated revoker. *fpr* may not contain spaces or colons. The optional 'sensitive' flag marks the designated revoker as sensitive information. Only v4 and v5 keys may be designated revokers.

Keyserver: *string*

This is an optional parameter that specifies the preferred keyserver URL for the key.

Handle: *string*

This is an optional parameter only used with the status lines KEY_CREATED and KEY_NOT_CREATED. *string* may be up to 100 characters and should not contain spaces. It is useful for batch key generation to associate a key parameter block with a status line.

Here is an example on how to create a key in an ephemeral home directory:

```
$ export GNUPGHOME="$(mktemp -d)"
$ cat >foo <<EOF
    %echo Generating a basic OpenPGP key
    Key-Type: DSA
    Key-Length: 1024
    Subkey-Type: ELG-E
    Subkey-Length: 1024
    Name-Real: Joe Tester
    Name-Comment: with stupid passphrase
    Name-Email: joe@foo.bar
    Expire-Date: 0
    Passphrase: abc
    # Do a commit here, so that we can later print "done" :-)
    %commit
    %echo done
EOF
$ gpg --batch --generate-key foo
 [...]
$ gpg --list-secret-keys
/tmp/tmp.0NQxB74PEf/pubring.kbx
------------------------------
sec   dsa1024 2016-12-16 [SCA]
      768E895903FC1C44045C8CB95EEBDB71E9E849D0
uid           [ultimate] Joe Tester (with stupid passphrase) <joe@foo.bar>
ssb   elg1024 2016-12-16 [E]
```

If you want to create a key with the default algorithms you would use these parameters:

```
    %echo Generating a default key
    Key-Type: default
    Subkey-Type: default
    Name-Real: Joe Tester
```

```
Name-Comment: with stupid passphrase
Name-Email: joe@foo.bar
Expire-Date: 0
Passphrase: abc
# Do a commit here, so that we can later print "done" :-)
%commit
%echo done
```

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.