NAME

gpgsm − CMS encryption and signing tool

SYNOPSIS

gpgsm [**−−homedir** *dir*] [**−−options** *file*] [*options*] *command* [*args*]

DESCRIPTION

gpgsm is a tool similar to **gpg** to provide digital encryption and signing services on X.509 certificates and the CMS protocol. It is mainly used as a backend for S/MIME mail processing. **gpgsm** includes a full featured certificate management and complies with all rules defined for the German Sphinx project.

COMMANDS

Commands are not distinguished from options except for the fact that only one command is allowed.

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 abbreviate this command.

--warranty
Print warranty information. Note that you cannot abbreviate this command.

--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

--encrypt
Perform an encryption. The keys the data is encrypted to must be set using the option **--recipient**.

--decrypt
Perform a decryption; the type of input is automatically determined. It may either be in binary form or PEM encoded; automatic determination of base-64 encoding is not done.

--sign Create a digital signature. The key used is either the fist one found in the keybox or those set with the **--local-user** option.

--verify
Check a signature file for validity. Depending on the arguments a detached signature may also be checked.

--server
Run in server mode and wait for commands on the **stdin**.

--call-dirmngr *command* [*args*]
Behave as a Dirmngr client issuing the request *command* with the optional list of *args*. The output of the Dirmngr is printed stdout. Please note that file names given as arguments should have an absolute file name (i.e. commencing with /) because they are passed verbatim to the Dirmngr and the working directory of the Dirmngr might not be the same as the one of this client. Currently it is not possible to pass data via stdin to the Dirmngr. *command* should not contain spaces.

This is command is required for certain maintaining tasks of the dirmngr where a dirmngr must be able to call back to **gpgsm**. See the Dirmngr manual for details.

--call-protect-tool *arguments*
Certain maintenance operations are done by an external program call **gpg-protect-tool**; this is usually not installed in a directory listed in the PATH variable. This command provides a simple wrapper to access this tool. *ar guments* are passed verbatim to this command; use '--help' to get a list of supported operations.

How to manage the certificates and keys

--generate-key
--gen-key
This command allows the creation of a certificate signing request or a self-signed certificate. It is commonly used along with the **--output** option to save the created CSR or certificate into a file. If used with the **--batch** a parameter file is used to create the CSR or certificate and it is further possible to create non-self-signed certificates.

--list-keys
-k List all available certificates stored in the local key database. Note that the displayed data might be reformatted for better human readability and illegal characters are replaced by safe substitutes.

--list-secret-keys
-K List all available certificates for which a corresponding a secret key is available.

--list-external-keys *pattern*
List certificates matching *pattern* using an external server. This utilizes the**dirmngr** service.

--list-chain
Same as **--list-keys** but also prints all keys making up the chain.

--dump-cert
--dump-keys
List all available certificates stored in the local key database using a format useful mainly for debugging.

--dump-chain
Same as **--dump-keys** but also prints all keys making up the chain.

--dump-secret-keys
List all available certificates for which a corresponding a secret key is available using a format useful mainly for debugging.

--dump-external-keys *pattern*
List certificates matching *pattern* using an external server. This utilizes the**dirmngr** service. It uses a format useful mainly for debugging.

--keydb-clear-some-cert-flags
This is a debugging aid to reset certain flags in the key database which are used to cache certain certificate stati. It is especially useful if a bad CRL or a weird running OCSP responder did accidentally revoke certificate. There is no security issue with this command because **gpgsm** always make sure that the validity of a certificate is checked right before it is used.

--delete-keys *pattern*
Delete the keys matching *pattern*. Note that there is no command to delete the secret part of the key directly. In case you need to do this, you should run the command**gpgsm --dump-secr et-keys KEYID** before you delete the key, copy the string of hex-digits in the ''keygrip'' line and delete the file consisting of these hex-digits and the suffix **.key** from the '*private-keys-v1.d*' directory below our GnuPG home directory (usually '*~/.gnupg*').

--export [*pattern*]
Export all certificates stored in the Keybox or those specified by the optional *pattern*. Those pattern consist of a list of user ids (see: [how-to-specify-a-user-id]). When used along with the**--ar - mor** option a few informational lines are prepended before each block. There is one limitation: As there is no commonly agreed upon way to pack more than one certificate into an ASN.1 structure, the binary export (i.e. without using **armor**) works only for the export of one certificate. Thus it is required to specify a *pattern* which yields exactly one certificate. Ephemeral certificate are only exported if all *pattern* are given as fingerprints or keygrips.

--export-secret-key-p12 *key-id*
Export the private key and the certificate identified by *key-id* using the PKCS#12 format. When used with the **--armor** option a few informational lines are prepended to the output. Note, that the PKCS#12 format is not very secure and proper transport security should be used to convey the exported key. (See: [option --p12-charset].)

--export-secret-key-p8 *key-id*
--export-secret-key-raw *key-id*
Export the private key of the certificate identified by *key-id* with any encryption stripped. The **...-raw** command exports in PKCS#1 format; the **...-p8** command exports in PKCS#8 format. When used with the **--armor** option a few informational lines are prepended to the output. These commands are useful to prepare a key for use on a TLS server.

--import [*files***]**
Import the certificates from the PEM or binary encoded files as well as from signed-only messages. This command may also be used to import a secret key from a PKCS#12 file.

--learn-card
Read information about the private keys from the smartcard and import the certificates from there. This command utilizes the **gpg-agent** and in turn the **scdaemon**.

--change-passphrase *user_id*
--passwd *user_id*
Change the passphrase of the private key belonging to the certificate specified as *user_id*. Note, that changing the passphrase/PIN of a smartcard is not yet supported.

OPTIONS

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

How to change the configuration

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

--options *file*
Reads configuration from *file* instead of from the default per-user configuration file. The default configuration file is named '*gpgsm.conf*' and expected in the '*.gnupg*' directory directly below the home directory of the user.

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

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

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

-v
--verbose
Outputs additional information while running. You can increase the verbosity by giving several verbose commands to **gpgsm**, such as '-vv'.

--policy-file *filename*
Change the default name of the policy file to *filename*.

--agent-program *file*
Specify an agent program to be used for secret key operations. The default value is determined by running the command **gpgconf**. 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 CRL checks. The default value is '*/usr/bin/dirmngr*'.

--prefer-system-dirmngr
This option is obsolete and ignored.

--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**.

--no-secmem-warning
Do not print a warning when the so called "secure memory" cannot be used.

--log-file *file*
When running in server mode, append all logging output to *file*. Use '*socket://*' to log to socket.

Certificate related options

--enable-policy-checks
--disable-policy-checks
By default policy checks are enabled. These options may be used to change it.

--enable-crl-checks
--disable-crl-checks
By default the CRL checks are enabled and the DirMngr is used to check for revoked certificates. The disable option is most useful with an off-line network connection to suppress this check and also to avoid that new certificates introduce a web bug by including a certificate specific CRL DP. The disable option also disables an issuer certificate lookup via the authorityInfoAccess property of the certificate; the **--enable-issuer-key-retrieve** can be used to make use of that property anyway.

--enable-trusted-cert-crl-check
--disable-trusted-cert-crl-check
By default the CRL for trusted root certificates are checked like for any other certificates. This allows a CA to revoke its own certificates voluntary without the need of putting all ever issued

certificates into a CRL. The disable option may be used to switch this extra check off. Due to the caching done by the Dirmngr, there will not be any noticeable performance gain. Note, that this also disables possible OCSP checks for trusted root certificates. A more specific way of disabling this check is by adding the ‘‘relax’’ keyword to the root CA line of the ‘*trustlist.txt*’

--force-crl-refresh
Tell the dirmngr to reload the CRL for each request. For better performance, the dirmngr will actually optimize this by suppressing the loading for short time intervals (e.g. 30 minutes). This option is useful to make sure that a fresh CRL is available for certificates hold in the keybox. The suggested way of doing this is by using it along with the option **--with-validation** for a key listing command. This option should not be used in a configuration file.

--enable-issuer-based-crl-check
Run a CRL check even for certificates which do not have any CRL distribution point. This requires that a suitable LDAP server has been configured in Dirmngr and that the CRL can be found using the issuer. This option reverts to what GnuPG did up to version 2.2.20. This option is in general not useful.

--enable-ocsp
--disable-ocsp
By default OCSP checks are disabled. The enable option may be used to enable OCSP checks via Dirmngr. If CRL checks are also enabled, CRLs will be used as a fallback if for some reason an OCSP request will not succeed. Note, that you have to allow OCSP requests in Dirmngr’s configuration too (option **--allow-ocsp**) and configure Dirmngr properly. If you do not do so you will get the error code ‘Not supported’.

--auto-issuer-key-retrieve
If a required certificate is missing while validating the chain of certificates, try to load that certificate from an external location. This usually means that Dirmngr is employed to search for the certificate. Note that this option makes a "web bug" like behavior possible. LDAP server 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 keybox), the operator can tell both your IP address and the time when you verified the signature.

--validation-model *name*
This option changes the default validation model. The only possible values are "shell" (which is the default), "chain" which forces the use of the chain model and "steed" for a new simplified model. The chain model is also used if an option in the ‘*trustlist.txt*’ or an attribute of the certificate requests it. However the standard model (shell) is in that case always tried first.

--ignore-cert-extension *oid*
Add *oid* to the list of ignored certificate extensions. The*oid* is e xpected to be in dotted decimal form, like **2.5.29.3**. This option may be used more than once. Critical flagged certificate extensions matching one of the OIDs in the list are treated as if they are actually handled and thus the certificate will not be rejected due to an unknown critical extension. Use this option with care because extensions are usually flagged as critical for a reason.

Input and Output

--armor
-a Create PEM encoded output. Default is binary output.

--base64
Create Base-64 encoded output; i.e. PEM without the header lines.

--assume-armor
Assume the input data is PEM encoded. Default is to autodetect the encoding but this is may fail.

--assume-base64
Assume the input data is plain base-64 encoded.

--assume-binary
Assume the input data is binary encoded.

--p12-charset *name*
gpgsm uses the UTF-8 encoding when encoding passphrases for PKCS#12 files. This option may be used to force the passphrase to be encoded in the specified encoding *name*. This is useful if the application used to import the key uses a different encoding and thus will not be able to import a file generated by **gpgsm**. Commonly used values for *name* are **Latin1** and **CP850**. Note that **gpgsm** itself automagically imports any file with a passphrase encoded to the most commonly used encodings.

--default-key *user_id*
Use *user_id* as the standard key for signing. This key is used if no other key has been defined as a signing key. Note, that the first**--local-users** option also sets this k ey if it has not yet been set; however **--default-key** always overrides this.

--local-user *user_id*
-u *user_id*
Set the user(s) to be used for signing. The default is the first secret key found in the database.

--recipient *name*
-r Encrypt to the user id *name*. There are several ways a user id may be given (see: [how-to-specify-a-user-id]).

--output *file*
-o *file* Write output to *file*. The default is to write it to stdout.

--with-key-data
Displays extra information with the **--list-keys** commands. Especially a line tagged **grp** is printed which tells you the keygrip of a key. This string is for example used as the file name of the secret key. Implies**--with-colons**.

--with-validation
When doing a key listing, do a full validation check for each key and print the result. This is usually a slow operation because it requires a CRL lookup and other operations.

When used along with **--import**, a validation of the certificate to import is done and only imported if it succeeds the test. Note that this does not affect an already available certificate in the DB. This option is therefore useful to simply verify a certificate.

--with-md5-fingerprint
For standard key listings, also print the MD5 fingerprint of the certificate.

--with-keygrip
Include the keygrip in standard key listings. Note that the keygrip is always listed in **--with-colons** mode.

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

How to change how the CMS is created

--include-certs *n*
Using *n* of -2 includes all certificate except for the root cert, -1 includes all certs, 0 does not include any certs, 1 includes only the signers cert and all other positive values include up to *n* certificates starting with the signer cert. The default is -2.

--cipher-algo *oid*
Use the cipher algorithm with the ASN.1 object identifier *oid* for encryption. For convenience the strings **3DES**, **AES** and **AES256** may be used instead of their OIDs. The default is **AES** (2.16.840.1.101.3.4.1.2).

--digest-algo name
Use **name** as the message digest algorithm. Usually this algorithm is deduced from the respective signing certificate. This option forces the use of the given algorithm and may lead to severe interoperability problems.

Doing things one usually do not want to do

--extra-digest-algo *name*
Sometimes signatures are broken in that they announce a different digest algorithm than actually used. **gpgsm** uses a one-pass data processing model and thus needs to rely on the announced digest algorithms to properly hash the data. As a workaround this option may be used to tell **gpgsm** to also hash the data using the algorithm *name*; this slows processing down a little bit but allows verification of such broken signatures. If **gpgsm** prints an error like ‘‘digest algo 8 has not been enabled’’ you may want to try this option, with ‘SHA256’ for *name*.

--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").

--with-ephemeral-keys
Include ephemeral flagged keys in the output of key listings. Note that they are included anyway if the key specification for a listing is given as fingerprint or keygrip.

--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*
This option is only useful for debugging and the behaviour may change at any time without notice; using **--debug-levels** is the preferred method to select the debug verbosity. FLAGS are bit encoded and may be given in usual C-Syntax. The currently defined bits are:

0 (1) X.509 or OpenPGP protocol related data

1 (2) values of big number integers

2 (4) low level crypto operations

5 (32) memory allocation

6 (64) caching

7 (128) show memory statistics

9 (512) write hashed data to files named **dbgmd-000***

10 (1024)
trace Assuan protocol

Note, that all flags set using this option may get overridden by **--debug-level**.

--debug-all
Same as **--debug=0xffffffff**

--debug-allow-core-dump
Usually **gpgsm** tries to avoid dumping core by well written code and by disabling core dumps for security reasons. However, bugs are pretty durable beasts and to squash them it is sometimes useful to have a core dump. This option enables core dumps unless the Bad Thing happened before the option parsing.

--debug-no-chain-validation
This is actually not a debugging option but only useful as such. It lets **gpgsm** bypass all certificate chain validation checks.

--debug-ignore-expiration
This is actually not a debugging option but only useful as such. It lets **gpgsm** ignore all notAfter dates, this is used by the regression tests.

--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 this passphrase is only used if the option **--batch** has also been given.

--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.

--request-origin *origin*
Tell gpgsm 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.

--no-common-certs-import
Suppress the import of common certificates on keybox creation.

All the long options may also be given in the configuration file after stripping off the two leading dashes.

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
123434343434343C34343434343734349A3434
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 allo ws 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 b ut 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.

EXAMPLES

$ gpgsm -er goo@bar.net <plaintext >ciphertext

FILES

There are a few configuration files to control certain aspects of **gpgsm**'s operation. Unless noted, they are expected in the current home directory (see: [option --homedir]).

gpgsm.conf

This is the standard configuration file read by **gpgsm** 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: [gpgsm-option --options]). You should backup this file.

policies.txt

This is a list of allowed CA policies. This file should list the object identifiers of the policies line by line. Empty lines and lines starting with a hash mark are ignored. Policies missing in this file and not marked as critical in the certificate will print only a warning; certificates with policies marked as critical and not listed in this file will fail the signature verification. You should backup this file.

For example, to allow only the policy 2.289.9.9, the file should look like this:

```
# Allowed policies
2.289.9.9
```

qualified.txt

This is the list of root certificates used for qualified certificates. They are defined as certificates capable of creating legally binding signatures in the same way as handwritten signatures are. Comments start with a hash mark and empty lines are ignored. Lines do have a length limit but this is not a serious limitation as the format of the entries is fixed and checked by **gpgsm**: A non-comment line starts with optional whitespace, followed by exactly 40 hex characters, white space and a lowercased 2 letter country code. Additional data delimited with by a white space is current ignored but might late be used for other purposes.

Note that even if a certificate is listed in this file, this does not mean that the certificate is trusted; in general the certificates listed in this file need to be listed also in '*trustlist.txt*'.

This is a global file an installed in the data directory (e.g. '*/usr/share/gnupg/qualified.txt*'). GnuPG installs a suitable file with root certificates as used in Germany. As new Root-CA

certificates may be issued over time, these entries may need to be updated; new distributions of this software should come with an updated list but it is still the responsibility of the Administrator to check that this list is correct.

Every time **gpgsm** uses a certificate for signing or verification this file will be consulted to check whether the certificate under question has ultimately been issued by one of these CAs. If this is the case the user will be informed that the verified signature represents a legally binding (''qualified'') signature. When creating a signature using such a certificate an extra prompt will be issued to let the user confirm that such a legally binding signature shall really be created.

Because this software has not yet been approved for use with such certificates, appropriate notices will be shown to indicate this fact.

help.txt

This is plain text file with a few help entries used with **pinentry** as well as a large list of help items for **gpg** and **gpgsm**. The standard file has English help texts; to install localized versions use filenames like '*help.LL.txt*' with LL denoting the locale. GnuPG comes with a set of predefined help files in the data directory (e.g. '*/usr/share/gnupg/gnupg/help.de.txt*') and allows overriding of any help item by help files stored in the system configuration directory (e.g. '*/etc/gnupg/help.de.txt*'). For a reference of the help file's syntax, please see the installed '*help.txt*' file.

com-certs.pem

This file is a collection of common certificates used to populated a newly created '*pubring.kbx*'. An administrator may replace this file with a custom one. The format is a concatenation of PEM encoded X.509 certificates. This global file is installed in the data directory (e.g. '*/usr/share/gnupg/com-certs.pem*').

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 **gpgsm** creates and maintains a few other files; they all live in the current home directory (see: [option --homedir]). Only **gpgsm** may modify these files.

pubring.kbx

This a database file storing the certificates as well as meta information. For debugging purposes the tool **kbxutil** may be used to show the internal structure of this file. You should backup this file.

random_seed

This content of this file is used to maintain the internal state of the random number generator across invocations. The same file is used by other programs of this software too.

S.gpg-agent

If this file exists **gpgsm** will first try to connect to this socket for accessing **gpg-agent** before starting a new **gpg-agent** instance. Under Windows this socket (which in reality be a plain file describing a regular TCP listening port) is the standard way of connecting the **gpg-agent**.

SEE ALSO

gpg2(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.