Short Contents

1 Introduction .................................................. 1
2 Preparation .................................................. 3
3 Protocols and Engines ................................. 10
4 Algorithms ...................................................... 15
5 Error Handling .................................................. 17
6 Exchanging Data .............................................. 24
7 Contexts .......................................................... 33
A The GnuPG UI Server Protocol .................... 113
B How to solve problems ................................. 122
C Deprecated Functions ........................................ 123
GNU Lesser General Public License .................. 134
GNU General Public License ............................ 143
Concept Index ...................................................... 154
Function and Data Index ................................. 157
# Table of Contents

1 Introduction ................................................. 1  
   1.1 Getting Started ........................................... 1  
   1.2 Features .................................................. 1  
   1.3 Overview ............................................... 2  

2 Preparation ................................................. 3  
   2.1 Header ................................................... 3  
   2.2 Building the Source ...................................... 3  
   2.3 Largefile Support (LFS) .................................. 4  
   2.4 Using Automake .......................................... 5  
   2.5 Using Libtool ............................................ 6  
   2.6 Library Version Check ................................... 6  
   2.7 Signal Handling ......................................... 8  
   2.8 Multi-Threading ......................................... 8  

3 Protocols and Engines ................................. 10  
   3.1 Engine Version Check .................................... 11  
   3.2 Engine Information ...................................... 12  
   3.3 Engine Configuration .................................... 14  
   3.4 OpenPGP .................................................. 14  
   3.5 Cryptographic Message Syntax ......................... 14  
   3.6 Assuan .................................................... 14  

4 Algorithms ................................................. 15  
   4.1 Public Key Algorithms ................................... 15  
   4.2 Hash Algorithms ......................................... 16  

5 Error Handling ............................................. 17  
   5.1 Error Values ............................................. 17  
   5.2 Error Sources ............................................ 19  
   5.3 Error Codes .............................................. 20  
   5.4 Error Strings ............................................ 22  

6 Exchanging Data ........................................... 24  
   6.1 Creating Data Buffers ................................... 24  
      6.1.1 Memory Based Data Buffers ......................... 24  
      6.1.2 File Based Data Buffers ............................ 25  
      6.1.3 Callback Based Data Buffers ...................... 26  
   6.2 Destroying Data Buffers ................................ 28  
   6.3 Manipulating Data Buffers .............................. 28  
      6.3.1 Data Buffer I/O Operations ....................... 28  
      6.3.2 Data Buffer Meta-Data ............................. 29  
      6.3.3 Data Buffer Convenience Functions .............. 31
1 Introduction

‘GnuPG Made Easy’ (GPGME) is a C language library that allows to add support for cryptography to a program. It is designed to make access to public key crypto engines like GnuPG or GpgSM easier for applications. GPGME provides a high-level crypto API for encryption, decryption, signing, signature verification and key management.

GPGME uses GnuPG and GpgSM as its backends to support OpenPGP and the Cryptographic Message Syntax (CMS).

1.1 Getting Started

This manual documents the GPGME library programming interface. All functions and data types provided by the library are explained.

The reader is assumed to possess basic knowledge about cryptography in general, and public key cryptography in particular. The underlying cryptographic engines that are used by the library are not explained, but where necessary, special features or requirements by an engine are mentioned as far as they are relevant to GPGME or its users.

This manual can be used in several ways. If read from the beginning to the end, it gives a good introduction into the library and how it can be used in an application. Forward references are included where necessary. Later on, the manual can be used as a reference manual to get just the information needed about any particular interface of the library. Experienced programmers might want to start looking at the examples at the end of the manual, and then only read up those parts of the interface which are unclear.

The documentation for the language bindings is currently not included in this manual. Those languages bindings follow the general programming model of GPGME but may provide some extra high level abstraction on top of the GPGME style API. For now please see the README files in the ‘lang/’ directory of the source distribution.

1.2 Features

GPGME has a couple of advantages over other libraries doing a similar job, and over implementing support for GnuPG or other crypto engines into your application directly.

it’s free software

Anybody can use, modify, and redistribute it under the terms of the GNU Lesser General Public License (see [Library Copying], page 134).

it’s flexible

GPGME provides transparent support for several cryptographic protocols by different engines. Currently, GPGME supports the OpenPGP protocol using GnuPG as the backend, and the Cryptographic Message Syntax using GpgSM as the backend.

it’s easy

GPGME hides the differences between the protocols and engines from the programmer behind an easy-to-use interface. This way the programmer can focus on the other parts of the program, and still integrate strong cryptography in his application. Once support for GPGME has been added to a program, it is easy to add support for other crypto protocols once GPGME backends provide them.
it’s language friendly

GPGME comes with languages bindings for several common programming languages: Common Lisp, C++, Python 2, and Python 3.

1.3 Overview

GPGME provides a data abstraction that is used to pass data to the crypto engine, and receive returned data from it. Data can be read from memory or from files, but it can also be provided by a callback function.

The actual cryptographic operations are always set within a context. A context provides configuration parameters that define the behaviour of all operations performed within it. Only one operation per context is allowed at any time, but when one operation is finished, you can run the next operation in the same context. There can be more than one context, and all can run different operations at the same time.

Furthermore, GPGME has rich key management facilities including listing keys, querying their attributes, generating, importing, exporting and deleting keys, and acquiring information about the trust path.

With some precautions, GPGME can be used in a multi-threaded environment, although it is not completely thread safe and thus needs the support of the application.
## 2 Preparation

To use GPGME, you have to perform some changes to your sources and the build system. The necessary changes are small and explained in the following sections. At the end of this chapter, it is described how the library is initialized, and how the requirements of the library are verified.

### 2.1 Header

All interfaces (data types and functions) of the library are defined in the header file `gpgme.h`. You must include this in all programs using the library, either directly or through some other header file, like this:

```c
#include <gpgme.h>
```

The name space of GPGME is `gpgme_*` for function names and data types and `GPGME_*` for other symbols. Symbols internal to GPGME take the form `_gpgme_*` and `_GPGME_*`.

Because GPGME makes use of the GPG Error library, using GPGME will also use the `GPG_ERR_*` name space directly, and the `gpg_err*`, `gpg_str*`, and `gpgrt_*` name space indirectly.

### 2.2 Building the Source

If you want to compile a source file including the `gpgme.h` header file, you must make sure that the compiler can find it in the directory hierarchy. This is accomplished by adding the path to the directory in which the header file is located to the compilers include file search path (via the `-I` option).

However, the path to the include file is determined at the time the source is configured. To solve this problem, gpgme ships with `gpgme.pc` file, that knows about the path to the include file and other configuration options. The command, `pkg-config`, can be used to handle information with `gpgme.pc` file. In an environment which doesn’t have `pkg-config` (like the one in early stage of OS bootstrap), for Automake, you can use `gpgme.m4` which invokes `gpgrt-config` with `gpgme.pc`. (In the past, gpgme used to ship with a small helper program `gpgme-config`. This functionality of `gpgme-config` is replaced by `pkg-config` with `gpgme.pc` file.)

The options that need to be added to the compiler invocation at compile time are output by the `--cflags` option to `pkg-config gpgme`. The following example shows how it can be used at the command line:

```bash
gcc -c foo.c ‘pkg-config --cflags gpgme'
```

Adding the output of `pkg-config --cflags gpgme` to the compiler command line will ensure that the compiler can find the GPGME header file.

A similar problem occurs when linking the program with the library. Again, the compiler has to find the library files. For this to work, the path to the library files has to be added to the library search path (via the `-L` option). For this, the option `--libs` to `pkg-config gpgme` can be used. For convenience, this option also outputs all other options that are required to link the program with GPGME (in particular, the `--lpgpme` option). The example shows how to link `foo.o` with the GPGME library to a program `foo`.
gcc -o foo foo.o 'pkg-config --libs gpgme'

Of course you can also combine both examples to a single command by specifying both options to pkg-config gpgme:

gcc -o foo foo.c 'pkg-config --cflags --libs gpgme'

2.3 Largefile Support (LFS)

GPGME is compiled with largefile support by default, if it is available on the system. This means that GPGME supports files larger than two gigabyte in size, if the underlying operating system can. On some systems, largefile support is already the default. On such systems, nothing special is required. However, some systems provide only support for files up to two gigabyte in size by default. Support for larger file sizes has to be specifically enabled.

To make a difficult situation even more complex, such systems provide two different types of largefile support. You can either get all relevant functions replaced with alternatives that are largefile capable, or you can get new functions and data types for largefile support added. Those new functions have the same name as their smallfile counterparts, but with a suffix of 64.

An example: The data type off_t is 32 bit wide on GNU/Linux PC systems. To address offsets in large files, you can either enable largefile support add-on. Then a new data type off64_t is provided, which is 64 bit wide. Or you can replace the existing off_t data type with its 64 bit wide counterpart. All occurrences of off_t are then automagically replaced.

As if matters were not complex enough, there are also two different types of file descriptors in such systems. This is important because if file descriptors are exchanged between programs that use a different maximum file size, certain errors must be produced on some file descriptors to prevent subtle overflow bugs from occurring.

As you can see, supporting two different maximum file sizes at the same time is not at all an easy task. However, the maximum file size does matter for GPGME, because some data types it uses in its interfaces are affected by that. For example, the off_t data type is used in the gpgme_data_seek function, to match its POSIX counterpart. This affects the call-frame of the function, and thus the ABI of the library. Furthermore, file descriptors can be exchanged between GPGME and the application.

For you as the user of the library, this means that your program must be compiled in the same file size mode as the library. Luckily, there is absolutely no valid reason for new programs to not enable largefile support by default and just use that. The compatibility modes (small file sizes or dual mode) can be considered an historic artefact, only useful to allow for a transitional period.

On POSIX platforms GPGME is compiled using largefile support by default. This means that your application must do the same, at least as far as it is relevant for using the ‘gpgme.h’ header file. All types in this header files refer to their largefile counterparts, if they are different from any default types on the system.

On 32 and 64 bit Windows platforms off_t is declared as 32 bit signed integer. There is no specific support for LFS in the C library. The recommendation from Microsoft is to use the native interface (CreateFile et al.) for large files. Released binary versions of GPGME (libgpgme-11.dll) have always been build with a 32 bit off_t. To avoid an ABI
break we stick to this convention for 32 bit Windows by using `long` there. GPGME versions for 64 bit Windows have never been released and thus we are able to use `int64_t` instead of `off_t` there. For easier migration the typedef `gpgme_off_t` has been defined. The reason we cannot use `off_t` directly is that some toolchains (e.g., mingw64) introduce a POSIX compatible hack for `off_t`. Some widely used toolkits make use of this hack and in turn GPGME would need to use it also. However, this would introduce an ABI break and existing software making use of libgpgme might suffer from a severe break. Thus with version 1.4.2 we redefined all functions using `off_t` to use `gpgme_off_t` which is defined as explained above. This way we keep the ABI well defined and independent of any toolchain hacks. The bottom line is that LFS support in GPGME is only available on 64 bit versions of Windows.

On POSIX platforms you can enable largefile support, if it is different from the default on the system the application is compiled on, by using the Autoconf macro `AC_SYS_LARGEFILE`. If you do this, then you don’t need to worry about anything else: It will just work. In this case you might also want to use `AC_FUNC_FSEEKO` to take advantage of some new interfaces, and `AC_TYPE_OFF_T` (just in case).

If you do not use Autoconf, you can define the preprocessor symbol `_FILE_OFFSET_BITS` to 64 before including any header files, for example by specifying the option `-D_FILE_OFFSET_BITS=64` on the compiler command line. You will also want to define the preprocessor symbol `LARGEFILE_SOURCE` to 1 in this case, to take advantage of some new interfaces.

If you do not want to do either of the above, you probably know enough about the issue to invent your own solution. Just keep in mind that the GPGME header file expects that largefile support is enabled, if it is available. In particular, we do not support dual mode (`_LARGEFILE64_SOURCE`).

### 2.4 Using Automake

You can simply use `PKG_CHECK_MODULES` macro with `pkg-config`:

```bash
PKG_CHECK_MODULES([GPGME], [gpgme >= 1.23.1])
```

Alternatively, instead of using `pkg-config`, for building on an environment with no pkg-config, GPGME provides an extension to Automake that does all the work for you. Please note that it is required to have gpgrt-config from libgpg-error installed in this case.

```makefile
AM_PATH_GPGME ([minimum-version], [action-if-found], [action-if-not-found])
```

Check whether GPGME (at least version `minimum-version`, if given) exists on the host system. If it is found, execute `action-if-found`, otherwise do `action-if-not-found`, if given.

This macro locates for `gpgme.pc`, with cross-compile support.

Additionally, the function defines `GPGME_CFLAGS` to the flags needed for compilation of the program to find the ‘`gpgme.h`’ header file, and `GPGME_LIBS` to the linker flags needed to link the program to the GPGME library.

`AM_PATH_GPGME_PTHREAD` was provided to check for the version of GPGME with the native pthread implementation, and it defined `GPGME_PTHREAD_CFLAGS` and `GPGME_PTHREAD_LIBS`. Since version 1.8.0 this is no longer necessary, as GPGME itself is
thread safe. Please use plain AM_PATH_GPGME instead, with GPGME_CFLAGS and GPGME_LDFLAGS.

You can use the defined Autoconf variables like this in your ‘Makefile.am’:

```makefile
AM_CPPFLAGS = $(GPGME_CFLAGS)
LDADD = $(GPGME_LIBS)
```

### 2.5 Using Libtool

The easiest way is to just use GNU Libtool. If you use libtool, and link to libgpgme.la, everything will be done automatically by Libtool.

### 2.6 Library Version Check

```c
const char * gpgme_check_version (const char *required_version)
```

The function `gpgme_check_version` has four purposes. It can be used to retrieve the version number of the library. In addition it can verify that the version number is higher than a certain required version number. In either case, the function initializes some sub-systems, and for this reason alone it must be invoked early in your program, before you make use of the other functions in GPGME. The last purpose is to run selftests.

As a side effect for W32 based systems, the socket layer will get initialized.

If `required_version` is NULL, the function returns a pointer to a statically allocated string containing the version number of the library.

If `required_version` is not NULL, it should point to a string containing a version number, and the function checks that the version of the library is at least as high as the version number provided. In this case, the function returns a pointer to a statically allocated string containing the version number of the library. If `REQUIRED_VERSION` is not a valid version number, or if the version requirement is not met, the function returns NULL.

If you use a version of a library that is backwards compatible with older releases, but contains additional interfaces which your program uses, this function provides a run-time check if the necessary features are provided by the installed version of the library.

If a selftest fails, the function may still succeed. Selftest errors are returned later when invoking `gpgme_new` or `gpgme-data_new`, so that a detailed error code can be returned (historically, `gpgme_check_version` does not return a detailed error code).

```c
int gpgme_set_global_flag (const char *name, const char *value)
```

Since: 1.4.0

On some systems it is not easy to set environment variables and thus hard to use GPGME’s internal trace facility for debugging. This function has been introduced as an alternative way to enable debugging and for a couple of other rarely used tweaks. It is important to assure that only one thread accesses GPGME functions between a call to this function and after the return from the call to `gpgme_check_version`. 
All currently supported features require that this function is called as early as possible — even before `gpgme_check_version`. The features are identified by the following values for `name`:

**debug**  
To enable debugging use the string “debug” for `name` and `value` identical to the value used with the environment variable `GPGME_DEBUG`.

**disable-gpgconf**  
Using this feature with any value disables the detection of the gpgconf program and thus forces GPGME to fallback into the simple OpenPGP only mode. It may be used to force the use of GnuPG-1 on systems which have both GPG versions installed. Note that in general the use of `gpgme_set_engine_info` is a better way to select a specific engine version.

**gpgconf-name**  
**gpg-name**  
Set the name of the gpgconf respective gpg binary. The defaults are `GNU/GnuPG/gpgconf` and `GNU/GnuPG/gpg`. Under Unix the leading directory part is ignored. Under Windows the leading directory part is used as the default installation directory; the `.exe` suffix is added by GPGME. Use forward slashed even under Windows.

**require-gnupg**  
Set the minimum version of the required GnuPG engine. If that version is not met, GPGME fails early instead of trying to use the existent version. The given version must be a string with major, minor, and micro number. Example: "2.1.0".

**inst-type**  
The installation type is used to prefer a certain GnuPG installation. The value is interpreted as an integer: A value of 0 is ignored, a value of 1 indicates an installation scheme as used by Gpg4win, a value of 2 indicates an installation scheme as used by GnuPG Desktop on Windows. All other values are reserved.

**w32-inst-dir**  
On Windows GPGME needs to know its installation directory to find its spawn helper. This is in general no problem because a DLL has this information. Some applications however link statically to GPGME and thus GPGME can only figure out the installation directory of this application which may be wrong in certain cases. By supplying an installation directory as value to this flag, GPGME will assume that that directory is the installation directory. This flag has no effect on non-Windows platforms.

This function returns 0 on success. In contrast to other functions the non-zero return value on failure does not convey any error code. For setting “debug” the only possible error cause is an out of memory condition; which would exhibit itself later anyway. Thus the return value may be ignored.

After initializing GPGME, you should set the locale information to the locale required for your output terminal. This locale information is needed for example for the curses and Gtk pinentry. Here is an example of a complete initialization:
#include <locale.h>
#include <gpgme.h>

void
init_gpgme (void)
{
    /* Initialize the locale environment. */
    setlocale (LC_ALL, "");
    gpgme_check_version (NULL);
    gpgme_set_locale (NULL, LC_CTYPE, setlocale (LC_CTYPE, NULL));
#ifdef LC_MESSAGES
    gpgme_set_locale (NULL, LC_MESSAGES, setlocale (LC_MESSAGES, NULL));
#endif
    gpgme_set_locale (NULL, LC_MESSAGES, setlocale (LC_MESSAGES, NULL));
}

Note that you are highly recommended to initialize the locale settings like this. GPGME cannot do this for you because it would not be thread safe. The conditional on LC_MESSAGES is only necessary for portability to W32 systems.

## 2.7 Signal Handling

The GPGME library communicates with child processes (the crypto engines). If a child process dies unexpectedly, for example due to a bug, or system problem, a SIGPIPE signal will be delivered to the application. The default action is to abort the program. To protect against this, `gpgme_check_version` sets the SIGPIPE signal action to SIG_IGN, which means that the signal will be ignored.

GPGME will only do that if the signal action for SIGPIPE is SIG_DEF at the time `gpgme_check_version` is called. If it is something different, GPGME will take no action.

This means that if your application does not install any signal handler for SIGPIPE, you don’t need to take any precautions. If you do install a signal handler for SIGPIPE, you must be prepared to handle any SIGPIPE events that occur due to GPGME writing to a defunct pipe. Furthermore, if your application is multi-threaded, and you install a signal action for SIGPIPE, you must make sure you do this either before `gpgme_check_version` is called or afterwards.

## 2.8 Multi-Threading

The GPGME library is mostly thread-safe, and can be used in a multi-threaded environment but there are some requirements for multi-threaded use:

- The function `gpgme_check_version` must be called before any other function in the library, because it initializes the thread support subsystem in GPGME. To achieve this in multi-threaded programs, you must synchronize the memory with respect to other threads that also want to use GPGME. For this, it is sufficient to call `gpgme_check_version` before creating the other threads using GPGME\(^1\).

\(^1\) At least this is true for POSIX threads, as `pthread_create` is a function that synchronizes memory with respects to other threads. There are many functions which have this property, a complete list can be found in POSIX, IEEE Std 1003.1-2003, Base Definitions, Issue 6, in the definition of the term “Memory Synchronization”. For other thread packages other, more relaxed or more strict rules may apply.
• Any `gpgme_data_t` and `gpgme_ctx_t` object must only be accessed by one thread at a time. If multiple threads want to deal with the same object, the caller has to make sure that operations on that object are fully synchronized.

• Only one thread at any time is allowed to call `gpgme_wait`. If multiple threads call this function, the caller must make sure that all invocations are fully synchronized. It is safe to start asynchronous operations while a thread is running in `gpgme_wait`.

• The function `gpgme_strerror` is not thread safe. You have to use `gpgme_strerror_r` instead.
3 Protocols and Engines

GPGME supports several cryptographic protocols, however, it does not implement them. Rather it uses backends (also called engines) which implement the protocol. GPGME uses inter-process communication to pass data back and forth between the application and the backend, but the details of the communication protocol and invocation of the backend is completely hidden by the interface. All complexity is handled by GPGME. Where an exchange of information between the application and the backend is necessary, GPGME provides the necessary callback function hooks and further interfaces.

enum gpgme_protocol_t [Data type]
The gpgme_protocol_t type specifies the set of possible protocol values that are supported by GPGME. The following protocols are supported:

- GPGME_PROTOCOL_OpenPGP
- GPGME_PROTOCOL_OPENPGP
  This specifies the OpenPGP protocol.
- GPGME_PROTOCOL_CMS
  This specifies the Cryptographic Message Syntax.
- GPGME_PROTOCOL_GPGCONF
  Under development. Please ask on gnupg-devel@gnupg.org for help.
- GPGME_PROTOCOL_ASSUAN
  Since: 1.2.0
  This specifies the raw Assuan protocol.
- GPGME_PROTOCOL_G13
  Since: 1.3.0
  Under development. Please ask on gnupg-devel@gnupg.org for help.
- GPGME_PROTOCOL_UISERVER
  Under development. Please ask on gnupg-devel@gnupg.org for help.
- GPGME_PROTOCOL_SPAWN
  Since: 1.5.0
  Special protocol for use with gpgme_op_spawn.
- GPGME_PROTOCOL_UNKNOWN
  Reserved for future extension. You may use this to indicate that the used protocol is not known to the application. Currently, GPGME does not accept this value in any operation, though, except for gpgme_get_protocol_name.

const char * gpgme_get_protocol_name [Function]

The function gpgme_get_protocol_name returns a statically allocated string describing the protocol protocol, or NULL if the protocol number is not valid.
3.1 Engine Version Check

const char * gpgme_get_dirinfo (const char *what)   [Function]

SINCE: 1.5.0

The function gpgme_get_dirinfo returns a statically allocated string with the value associated to what. The returned values are the defaults and won’t change even after gpgme_set_engine_info has been used to configure a different engine. NULL is returned if no value is available. Commonly supported values for what are:

- homedir Return the default home directory.
- sysconfdir Return the name of the system configuration directory.
- bindir Return the name of the directory with GnuPG program files.
- libdir Return the name of the directory with GnuPG related library files.
- libexecdir Return the name of the directory with GnuPG helper program files.
- datadir Return the name of the directory with GnuPG shared data.
- localedir Return the name of the directory with GnuPG locale data.
- socketdir Return the name of the directory with the following sockets.
  - agent-socket Return the name of the socket to connect to the gpg-agent.
  - agent-ssh-socket Return the name of the socket to connect to the ssh-agent component of gpg-agent.
  - dirmngr-socket Return the name of the socket to connect to the dirmngr.
  - uiserver-socket Return the name of the socket to connect to the user interface server.
  - gpgconf-name Return the file name of the engine configuration tool.
  - gpg-name Return the file name of the OpenPGP engine.
  - gpgsm-name Return the file name of the CMS engine.
  - g13-name Return the name of the file container encryption engine.
  - keyboxd-name Return the name of the key database daemon.
  - agent-name Return the name of gpg-agent.
scdaemon-name
Return the name of the smart card daemon.

dirmngr-name
Return the name of dirmngr.

pinentry-name
Return the name of the pinentry program.

gpg-wks-client-name
Return the name of the Web Key Service tool.

gpgtar-name
Return the name of the gpgtar program.

gpgme_error_t gpgme_engine_check_version

([Function]
  (gpgme_protocol_t protocol)

The function gpgme_engine_check_version verifies that the engine implementing the protocol PROTOCOL is installed in the expected path and meets the version requirement of GPGME.

This function returns the error code GPG_ERR_NO_ERROR if the engine is available and GPG_ERR_INV_ENGINE if it is not.

3.2 Engine Information

The gpgme_engine_info_t type specifies a pointer to a structure describing a crypto engine. The structure contains the following elements:

gpgme_engine_info_t next
This is a pointer to the next engine info structure in the linked list, or NULL if this is the last element.

const char *protocol
This is the protocol for which the crypto engine is used. You can convert this to a string with gpgme_get_protocol_name for printing.

const char *file_name
This is a string holding the file name of the executable of the crypto engine. Currently, it is never NULL, but using NULL is reserved for future use, so always check before you use it.

const char *home_dir
This is a string holding the directory name of the crypto engine’s configuration directory. If it is NULL, then the default directory is used. See gpgme_get_dirinfo on how to get the default directory.

const char *version
This is a string containing the version number of the crypto engine. It might be NULL if the version number can not be determined, for example because the executable doesn’t exist or is invalid.
const char *req_version
This is a string containing the minimum required version number of the
crypto engine for GPGME to work correctly. This is the version num-
ber that gpgme_engine_check_version verifies against. Currently, it is
never NULL, but using NULL is reserved for future use, so always check
before you use it.

gpgme_error_t gpgme_get_engine_info
(gpgme_engine_info_t *info)
The function gpgme_get_engine_info returns a linked list of engine info structures
in info. Each info structure describes the defaults of one configured backend.
The memory for the info structures is allocated the first time this function is invoked,
and must not be freed by the caller.
This function returns the error code GPG_ERR_NO_ERROR if successful, and a system
error if the memory could not be allocated.

Here is an example how you can provide more diagnostics if you receive an error message
which indicates that the crypto engine is invalid.

gpgme_ctxt_t ctx;
gpgme_error_t err;

[...]

if (gpgme_err_code (err) == GPG_ERR_INV_ENGINE)
{
  gpgme_engine_info_t info;
  err = gpgme_get_engine_info (&info);
  if (!err)
  {
    while (info && info->protocol != gpgme_get_protocol (ctx))
      info = info->next;
    if (!info)
      fprintf (stderr, "GPGME compiled without support for protocol %s", ■
        gpgme_get_protocol_name (info->protocol));
    else if (info->file_name && !info->version)
      fprintf (stderr, "Engine %s not installed properly", ■
        info->file_name);
    else if (info->file_name && info->version && info->req_version)
      fprintf (stderr, "Engine %s version %s installed, " ■
        "but at least version %s required", info->file_name,
        info->version, info->req_version);
    else
      fprintf (stderr, "Unknown problem with engine for protocol %s", ■
        gpgme_get_protocol_name (info->protocol));
  }
}
Chapter 3: Protocols and Engines

3.3 Engine Configuration

You can change the configuration of a backend engine, and thus change the executable program and configuration directory to be used. You can make these changes the default or set them for some contexts individually.

```c
#include <gpgme.h>

[gpgme_error_t gpgme_set_engine_info (gpgme_protocol_t proto, const char *file_name, const char *home_dir)]
```

Since: 1.1.0

The function `gpgme_set_engine_info` changes the default configuration of the crypto engine implementing the protocol `proto`.

`file_name` is the file name of the executable program implementing this protocol, and `home_dir` is the directory name of the configuration directory for this crypto engine. If `home_dir` is `NULL`, the engine’s default will be used.

The new defaults are not applied to already created GPGME contexts.

This function returns the error code `GPG_ERR_NO_ERROR` if successful, or an error code on failure.

The functions `gpgme_ctx_get_engine_info` and `gpgme_ctx_set_engine_info` can be used to change the engine configuration per context. See Section 7.4.2 [Crypto Engine], page 34.

3.4 OpenPGP

OpenPGP is implemented by GnuPG, the GNU Privacy Guard. This is the first protocol that was supported by GPGME.

The OpenPGP protocol is specified by `GPGME_PROTOCOL_OpenPGP`.

3.5 Cryptographic Message Syntax

CMS is implemented by GpgSM, the S/MIME implementation for GnuPG.

The CMS protocol is specified by `GPGME_PROTOCOL_CMS`.

3.6 Assuan

Assuan is the RPC library used by the various GnuPG components. The Assuan protocol allows one to talk to arbitrary Assuan servers using GPGME. See Section 7.7.2 [Using the Assuan protocol], page 97.

The ASSUAN protocol is specified by `GPGME_PROTOCOL_ASSUAN`. 
4 Algorithms

The crypto backends support a variety of algorithms used in public key cryptography. The following sections list the identifiers used to denote such an algorithm.

4.1 Public Key Algorithms

Public key algorithms are used for encryption, decryption, signing and verification of signatures.

```c
enum gpgme_pubkey_algo_t
```

The `gpgme_pubkey_algo_t` type specifies the set of all public key algorithms that are supported by GPGME. Possible values are:

- **GPGME_PK_RSA**
  - This value indicates the RSA (Rivest, Shamir, Adleman) algorithm.

- **GPGME_PK_RSA_E**
  - Deprecated. This value indicates the RSA (Rivest, Shamir, Adleman) algorithm for encryption and decryption only.

- **GPGME_PK_RSA_S**
  - Deprecated. This value indicates the RSA (Rivest, Shamir, Adleman) algorithm for signing and verification only.

- **GPGME_PK_DSA**
  - This value indicates DSA, the Digital Signature Algorithm.

- **GPGME_PK_ELG**
  - This value indicates ElGamal.

- **GPGME_PK_ELG_E**
  - This value also indicates ElGamal and is used specifically in GnuPG.

- **GPGME_PK_ECC**
  - **Since:** 1.5.0
  - This value is a generic indicator for elliptic curve algorithms.

- **GPGME_PK_ECDSA**
  - **Since:** 1.3.0
  - This value indicates ECDSA, the Elliptic Curve Digital Signature Algorithm as defined by FIPS 186-2 and RFC-6637.

- **GPGME_PK_ECDH**
  - **Since:** 1.3.0
  - This value indicates ECDH, the Elliptic Curve Diffie-Hellmann encryption algorithm as defined by RFC-6637.

- **GPGME_PK_EDDSA**
  - **Since:** 1.7.0
  - This value indicates the EdDSA algorithm.

---

1 Some engines also provide symmetric only encryption; see the description of the encryption function on how to use this.
const char * gpgme_pubkey_algo_name (const gpgme_pubkey_algo_t algo)
    The function gpgme_pubkey_algo_name returns a pointer to a statically allocated string containing a description of the public key algorithm algo. This string can be used to output the name of the public key algorithm to the user.
    If algo is not a valid public key algorithm, NULL is returned.

char * gpgme_pubkey_algo_string (gpgme_subkey_t key)
    The function gpgme_pubkey_algo_string is a convenience function to build and return an algorithm string in the same way GnuPG does (e.g., "rsa2048" or "ed25519"). The caller must free the result using gpgme_free. On error (e.g., invalid argument or memory exhausted), the function returns NULL and sets ERRNO.

4.2 Hash Algorithms

Hash (message digest) algorithms are used to compress a long message to make it suitable for public key cryptography.

enum gpgme_hashalgo_t
    The gpgme_hashalgo_t type specifies the set of all hash algorithms that are supported by GPGME. Possible values are:
    GPGME_MD_MD5
    GPGME_MD_SHA1
    GPGME_MD_RMD160
    GPGME_MD_MD2
    GPGME_MD_TIGER
    GPGME_MD_HAVAL
    GPGME_MD_SHA256
    GPGME_MD_SHA384
    GPGME_MD_SHA512
    GPGME_MD_SHA224
    Since: 1.5.0
    GPGME_MD_MD4
    GPGME_MD_CRC32
    GPGME_MD_CRC32_RFC1510
    GPGME_MD_CRC24_RFC2440

const char * gpgme_hashalgo_name (gpgme_hashalgo_t algo)
    The function gpgme_hashalgo_name returns a pointer to a statically allocated string containing a description of the hash algorithm algo. This string can be used to output the name of the hash algorithm to the user.
    If algo is not a valid hash algorithm, NULL is returned.
5 Error Handling

Many functions in GPGME can return an error if they fail. For this reason, the application should always catch the error condition and take appropriate measures, for example by releasing the resources and passing the error up to the caller, or by displaying a descriptive message to the user and cancelling the operation.

Some error values do not indicate a system error or an error in the operation, but the result of an operation that failed properly. For example, if you try to decrypt a tempered message, the decryption will fail. Another error value actually means that the end of a data buffer or list has been reached. The following descriptions explain for many error codes what they mean usually. Some error values have specific meanings if returned by a certain functions. Such cases are described in the documentation of those functions.

GPGME uses the libgpg-error library. This allows to share the error codes with other components of the GnuPG system, and thus pass error values transparently from the crypto engine, or some helper application of the crypto engine, to the user. This way no information is lost. As a consequence, GPGME does not use its own identifiers for error codes, but uses those provided by libgpg-error. They usually start with GPG_ERR_.

However, GPGME does provide aliases for the functions defined in libgpg-error, which might be preferred for name space consistency.

5.1 Error Values

gpgme_err_code_t

The gpgme_err_code_t type is an alias for the libgpg-error type gpg_err_code_t. The error code indicates the type of an error, or the reason why an operation failed.

A list of important error codes can be found in the next section.

gpgme_err_source_t

The gpgme_err_source_t type is an alias for the libgpg-error type gpg_err_source_t. The error source has not a precisely defined meaning. Sometimes it is the place where the error happened, sometimes it is the place where an error was encoded into an error value. Usually the error source will give an indication to where to look for the problem. This is not always true, but it is attempted to achieve this goal.

A list of important error sources can be found in the next section.

gpgme_error_t

The gpgme_error_t type is an alias for the libgpg-error type gpg_error_t. An error value like this has always two components, an error code and an error source. Both together form the error value.

Thus, the error value can not be directly compared against an error code, but the accessor functions described below must be used. However, it is guaranteed that only 0 is used to indicate success (GPG_ERR_NO_ERROR), and that in this case all other parts of the error value are set to 0, too.

Note that in GPGME, the error source is used purely for diagnostical purposes. Only the error code should be checked to test for a certain outcome of a function. The
manual only documents the error code part of an error value. The error source is left unspecified and might be anything.

**static inline gpgme_err_code_t gpgme_err_code**

```c
(gpgme_error_t err)
```

The static inline function `gpgme_err_code` returns the `gpgme_err_code_t` component of the error value `err`. This function must be used to extract the error code from an error value in order to compare it with the GPG_ERR_* error code macros.

**static inline gpgme_err_source_t gpgme_err_source**

```c
(gpgme_error_t err)
```

The static inline function `gpgme_err_source` returns the `gpgme_err_source_t` component of the error value `err`. This function must be used to extract the error source from an error value in order to compare it with the GPG_ERR_SOURCE_* error source macros.

**static inline gpgme_error_t gpgme_err_make**

```c
(gpgme_err_source_t source, gpgme_err_code_t code)
```

The static inline function `gpgme_err_make` returns the error value consisting of the error source `source` and the error code `code`.

This function can be used in callback functions to construct an error value to return it to the library.

**static inline gpgme_error_t gpgme_error**

```c
(gpgme_err_code_t code)
```

The static inline function `gpgme_error` returns the error value consisting of the default error source and the error code `code`.

For GPGME applications, the default error source is GPG_ERR_SOURCE_USER_1. You can define GPGME_ERR_SOURCE_DEFAULT before including 'gpgme.h' to change this default.

This function can be used in callback functions to construct an error value to return it to the library.

The libgpg-error library provides error codes for all system error numbers it knows about. If `err` is an unknown error number, the error code GPG_ERR_UNKNOWN_ERRNO is used.

The following functions can be used to construct error values from system error numbers.

**gpgme_error_t gpgme_err_make_from_errno**

```c
(gpgme_err_source_t source, int err)
```

The function `gpgme_err_make_from_errno` is like `gpgme_err_make`, but it takes a system error like `errno` instead of a `gpgme_err_code_t` error code.

**gpgme_error_t gpgme_error_from_errno**

```c
(int err)
```

The function `gpgme_error_from_errno` is like `gpgme_error`, but it takes a system error like `errno` instead of a `gpgme_err_code_t` error code.

Sometimes you might want to map system error numbers to error codes directly, or map an error code representing a system error back to the system error number. The following functions can be used to do that.
The function `gpgme_err_code_from_errno` returns the error code for the system error `err`. If `err` is not a known system error, the function returns `GPG_ERR_UNKNOWN_ERRNO`.

The function `gpgme_err_code_to_errno` returns the system error for the error code `err`. If `err` is not an error code representing a system error, or if this system error is not defined on this system, the function returns 0.

## 5.2 Error Sources

The library `libgpg-error` defines an error source for every component of the GnuPG system. The error source part of an error value is not well defined. As such it is mainly useful to improve the diagnostic error message for the user.

If the error code part of an error value is 0, the whole error value will be 0. In this case the error source part is of course `GPG_ERR_SOURCE_UNKNOWN`.

The list of error sources that might occur in applications using GPGME is:

- **GPG_ERR_SOURCE_UNKNOWN**
  - The error source is not known. The value of this error source is 0.

- **GPG_ERR_SOURCE_GPGME**
  - The error source is GPGME itself. This is the default for errors that occur in the GPGME library.

- **GPG_ERR_SOURCE_GPG**
  - The error source is GnuPG, which is the crypto engine used for the OpenPGP protocol.

- **GPG_ERR_SOURCE_GPGSM**
  - The error source is GPGSM, which is the crypto engine used for the CMS protocol.

- **GPG_ERR_SOURCE_GCRYPT**
  - The error source is `libgcrypt`, which is used by crypto engines to perform cryptographic operations.

- **GPG_ERR_SOURCE_GPGAGENT**
  - The error source is `gpg-agent`, which is used by crypto engines to perform operations with the secret key.

- **GPG_ERR_SOURCE_PINENTRY**
  - The error source is `pinentry`, which is used by `gpg-agent` to query the passphrase to unlock a secret key.

- **GPG_ERR_SOURCE_SCD**
  - The error source is the SmartCard Daemon, which is used by `gpg-agent` to delegate operations with the secret key to a SmartCard.

- **GPG_ERR_SOURCE_KEYBOX**
  - The error source is `libkbx`, a library used by the crypto engines to manage local keyrings.
GPG_ERR_SOURCE_USER_1
GPG_ERR_SOURCE_USER_2
GPG_ERR_SOURCE_USER_3
GPG_ERR_SOURCE_USER_4

These error sources are not used by any GnuPG component and can be used by other software. For example, applications using GPGME can use them to mark error values coming from callback handlers. Thus GPG_ERR_SOURCE_USER_1 is the default for errors created with gpgme_error and gpgme_error_from_errno, unless you define GPGME_ERR_SOURCE_DEFAULT before including ‘gpgme.h’.

5.3 Error Codes

The library libgpg-error defines many error values. Most of them are not used by GPGME directly, but might be returned by GPGME because it received them from the crypto engine. The below list only includes such error codes that have a specific meaning in GPGME, or which are so common that you should know about them.

GPG_ERR_EOF
This value indicates the end of a list, buffer or file.

GPG_ERR_NO_ERROR
This value indicates success. The value of this error code is 0. Also, it is guaranteed that an error value made from the error code 0 will be 0 itself (as a whole). This means that the error source information is lost for this error code, however, as this error code indicates that no error occurred, this is generally not a problem.

GPG_ERR_GENERAL
This value means that something went wrong, but either there is not enough information about the problem to return a more useful error value, or there is no separate error value for this type of problem.

GPG_ERR_ENOMEM
This value means that an out-of-memory condition occurred.

GPG_ERR_E...
System errors are mapped to GPG_ERR_FOO where FOO is the symbol for the system error.

GPG_ERR_INV_VALUE
This value means that some user provided data was out of range. This can also refer to objects. For example, if an empty gpgme_data_t object was expected, but one containing data was provided, this error value is returned.

GPG_ERR_UNUSABLE_PUBKEY
This value means that some recipients for a message were invalid.

GPG_ERR_UNUSABLE_SECKEY
This value means that some signers were invalid.

GPG_ERR_NO_DATA
This value means that a gpgme_data_t object which was expected to have content was found empty.
GPG_ERR_CONFLICT
This value means that a conflict of some sort occurred.

GPG_ERR_NOT_IMPLEMENTED
This value indicates that the specific function (or operation) is not implemented. This error should never happen. It can only occur if you use certain values or configuration options which do not work, but for which we think that they should work at some later time.

GPG_ERR_DECRYPT_FAILED
This value indicates that a decryption operation was unsuccessful.

GPG_ERR_BAD_PASSPHRASE
This value means that the user did not provide a correct passphrase when requested.

GPG_ERR_CANCELED
This value means that the operation was canceled.

GPG_ERR_FULLY_CANCELED
This value means that the operation was canceled. It is sometimes returned instead of GPG_ERR_CANCELED for internal reasons in GnuPG. You should treat both values identically.

GPG_ERR_INV_ENGINE
This value means that the engine that implements the desired protocol is currently not available. This can either be because the sources were configured to exclude support for this engine, or because the engine is not installed properly.

GPG_ERR_AMBIGUOUS_NAME
This value indicates that a user ID or other specifier did not specify a unique key.

GPG_ERR_WRONG_KEY_USAGE
This value indicates that a key is not used appropriately.

GPG_ERR_CERT_REVOKED
This value indicates that a key signature was revoked.

GPG_ERR_CERT_EXPIRED
This value indicates that a key signature expired.

GPG_ERR_NO_CRL_KNOWN
This value indicates that no certificate revocation list is known for the certificate.

GPG_ERR_NO_POLICY_MATCH
This value indicates that a policy issue occurred.

GPG_ERR_NO_SECKEY
This value indicates that no secret key for the user ID is available.

GPG_ERR_MISSING_CERT
This value indicates that a key could not be imported because the issuer certificate is missing.
Chapter 5: Error Handling

GPG_ERR_BAD_CERT_CHAIN
This value indicates that a key could not be imported because its certificate chain is not good, for example it could be too long.

GPG_ERR_UNSUPPORTED_ALGORITHM
This value means a verification failed because the cryptographic algorithm is not supported by the crypto backend.

GPG_ERR_BAD_SIGNATURE
This value means a verification failed because the signature is bad.

GPG_ERR_NO_PUBKEY
This value means a verification failed because the public key is not available.

GPG_ERR_USER_1
GPG_ERR_USER_2
...
GPG_ERR_USER_16
These error codes are not used by any GnuPG component and can be freely used by other software. Applications using GPGME might use them to mark specific errors returned by callback handlers if no suitable error codes (including the system errors) for these errors exist already.

5.4 Error Strings

const char * gpgme_strerror (gpgme_error_t err)
[Function]
The function gpgme_strerror returns a pointer to a statically allocated string containing a description of the error code contained in the error value err. This string can be used to output a diagnostic message to the user.

This function is not thread safe. Use gpgme_strerror_r in multi-threaded programs.

int gpgme_strerror_r (gpgme_error_t err, char *buf, size_t buflen)
[Function]
The function gpgme_strerror_r returns the error string for err in the user-supplied buffer buf of size buflen. This function is, in contrast to gpgme_strerror, thread-safe if a thread-safe_strerror_r function is provided by the system. If the function succeeds, 0 is returned and buf contains the string describing the error. If the buffer was not large enough, ERANGE is returned and buf contains as much of the beginning of the error string as fits into the buffer.

const char * gpgme_strsource (gpgme_error_t err)
[Function]
The function gpgme_strerror returns a pointer to a statically allocated string containing a description of the error source contained in the error value err. This string can be used to output a diagnostic message to the user.

The following example illustrates the use of gpgme_strerror:

gpgme_ctx_t ctx;
gpgme_error_t err = gpgme_new (&ctx);
if (err)
{


fprintf (stderr, "%s: creating GpgME context failed: %s: %s\n",
    argv[0], gpgme_strsource (err), gpgme_strerror (err));
exit (1);
}
Chapter 6: Exchanging Data

A lot of data has to be exchanged between the user and the crypto engine, like plaintext messages, ciphertext, signatures and information about the keys. The technical details about exchanging the data information are completely abstracted by GPGME. The user provides and receives the data via `gpgme_data_t` objects, regardless of the communication protocol between GPGME and the crypto engine in use.

`gpgme_data_t` objects do not provide notifications on events. It is assumed that read and write operations are blocking until data is available. If this is undesirable, the application must ensure that all GPGME data operations always have data available, for example by using memory buffers or files rather than pipes or sockets. This might be relevant, for example, if the external event loop mechanism is used.

```
gpgme_data_t
[Data type]
The `gpgme_data_t` type is a handle for a container for generic data, which is used by GPGME to exchange data with the user.
```

`gpgme_off_t` since: 1.4.1

On POSIX platforms the `gpgme_off_t` type is an alias for `off_t`; it may be used interchangeable. On Windows platforms `gpgme_off_t` is defined as a long (i.e., 32 bit) for 32 bit Windows and as a 64 bit signed integer for 64 bit Windows.

```
gpgme_ssize_t
[Data type]
The `gpgme_ssize_t` type is an alias for `ssize_t`. It has only been introduced to overcome portability problems pertaining to the declaration of `ssize_t` by different toolchains.
```

6.1 Creating Data Buffers

Data objects can be based on memory, files, or callback functions provided by the user. Not all operations are supported by all objects.

6.1.1 Memory Based Data Buffers

Memory based data objects store all data in allocated memory. This is convenient, but only practical for an amount of data that is a fraction of the available physical memory. The data has to be copied from its source and to its destination, which can often be avoided by using one of the other data object

```
gpgme_error_t gpgme_data_new (gpgme_data_t *dh)
[Function]
The function `gpgme_data_new` creates a new `gpgme_data_t` object and returns a handle for it in `dh`. The data object is memory based and initially empty.

The function returns the error code `GPG_ERR_NO_ERROR` if the data object was successfully created, `GPG_ERR_INV_VALUE` if `dh` is not a valid pointer, and `GPG_ERR_ENOMEM` if not enough memory is available.
```
gpgme_error_t gpgme_data_new_from_mem (gpgme_data_t *dh,  
const char *buffer, size_t size, int copy)

The function gpgme_data_new_from_mem creates a new gpgme_data_t object and fills it with size bytes starting from buffer.

If copy is not zero, a private copy of the data is made. If copy is zero, the data is taken from the specified buffer as needed, and the user has to ensure that the buffer remains valid for the whole life span of the data object.

The function returns the error code GPG_ERR_NO_ERROR if the data object was successfully created, GPG_ERR_INV_VALUE if dh or buffer is not a valid pointer, and GPG_ERR_ENOMEM if not enough memory is available.

gpgme_error_t gpgme_data_new_from_file (gpgme_data_t *dh,  
const char *filename, int copy)

The function gpgme_data_new_from_file creates a new gpgme_data_t object and fills it with the content of the file filename.

If copy is not zero, the whole file is read in at initialization time and the file is not used anymore after that. This is the only mode supported currently. Later, a value of zero for copy might cause all reads to be delayed until the data is needed, but this is not yet implemented.

The function returns the error code GPG_ERR_NO_ERROR if the data object was successfully created, GPG_ERR_INV_VALUE if dh or filename is not a valid pointer, GPG_ERR_NOT_IMPLEMENTED if code is zero, and GPG_ERR_ENOMEM if not enough memory is available.

6.1.2 File Based Data Buffers

File based data objects operate directly on file descriptors or streams. Only a small amount of data is stored in core at any time, so the size of the data objects is not limited by GPGME.

gpgme_error_t gpgme_data_new_from_fd (gpgme_data_t *dh, int fd)

The function gpgme_data_new_from_fd creates a new gpgme_data_t object and uses the file descriptor fd to read from (if used as an input data object) and write to (if used as an output data object).

When using the data object as an input buffer, the function might read a bit more from the file descriptor than is actually needed by the crypto engine in the desired operation because of internal buffering.
Note that GPGME assumes that the file descriptor is set to blocking mode. Errors during I/O operations, except for EINTR, are usually fatal for crypto operations. The function returns the error code GPG_ERR_NO_ERROR if the data object was successfully created, and GPG_ERR_ENOMEM if not enough memory is available.

```c
void (*gpgme_data_read_cb_t) (void *handle,
                             void *buffer, size_t size)
```

The `gpgme_data_read_cb_t` type is the type of functions which GPGME calls if it wants to read data from a user-implemented data object. The function should read up to `size` bytes from the current read position into the space starting at `buffer`. The `handle` is provided by the user at data object creation time.

Note that GPGME assumes that the read blocks until data is available. Errors during I/O operations, except for EINTR, are usually fatal for crypto operations.

The function should return the number of bytes read, 0 on EOF, and -1 on error. If an error occurs, `errno` should be set to describe the type of the error.
Chapter 6: Exchanging Data

**ssize_t (**gpgme_data_write_cb_t**) (void *handle,**
const void *buffer,** size_t size)**
The **gpgme_data_write_cb_t** type is the type of functions which GPGME calls if it wants to write data to a user-implemented data object. The function should write up to size bytes to the current write position from the space starting at buffer. The handle is provided by the user at data object creation time.

Note that GPGME assumes that the write blocks until data is available. Errors during I/O operations, except for EINTR, are usually fatal for crypto operations.

The function should return the number of bytes written, and -1 on error. If an error occurs, errno should be set to describe the type of the error.

**off_t (**gpgme_data_seek_cb_t**) (void *handle,**
off_t offset,** int whence)**
The **gpgme_data_seek_cb_t** type is the type of functions which GPGME calls if it wants to change the current read/write position in a user-implemented data object, just like the lseek function.

The function should return the new read/write position, and -1 on error. If an error occurs, errno should be set to describe the type of the error.

**void (**gpgme_data_release_cb_t**) (void *handle)**
The **gpgme_data_release_cb_t** type is the type of functions which GPGME calls if it wants to destroy a user-implemented data object. The handle is provided by the user at data object creation time.

**struct gpgme_data_cbs**
This structure is used to store the data callback interface functions described above.
It has the following members:

- **gpgme_data_read_cb_t read**
  This is the function called by GPGME to read data from the data object.
  It is only required for input data object.

- **gpgme_data_write_cb_t write**
  This is the function called by GPGME to write data to the data object.
  It is only required for output data object.

- **gpgme_data_seek_cb_t seek**
  This is the function called by GPGME to change the current read/write pointer in the data object (if available). It is optional.

- **gpgme_data_release_cb_t release**
  This is the function called by GPGME to release a data object. It is optional.

**gpgme_error_t gpgme_data_new_from_cbs (gpgme_data_t *dh,**
gpgme_data_cbs_t cbs,** void *handle)**
The function **gpgme_data_new_from_cbs** creates a new **gpgme_data_t** object and uses the user-provided callback functions to operate on the data object.

The handle handle is passed as first argument to the callback functions. This can be used to identify this data object.
The function returns the error code GPG_ERR_NO_ERROR if the data object was successfully created, and GPG_ERR_ENOMEM if not enough memory is available.

### 6.2 Destroying Data Buffers

**void gpgme_data_release (gpgme_data_t dh)**

The function gpgme_data_release destroys the data object with the handle dh. It releases all associated resources that were not provided by the user in the first place.

**char * gpgme_data_release_and_get_mem (gpgme_data_t dh, size_t *length)**

The function gpgme_data_release_and_get_mem is like gpgme_data_release, except that it returns the data buffer and its length that was provided by the object.

The user has to release the buffer with gpgme_free. In case the user provided the data buffer in non-copy mode, a copy will be made for this purpose.

In case an error returns, or there is no suitable data buffer that can be returned to the user, the function will return NULL. In any case, the data object dh is destroyed.

**void gpgme_free (void *buffer)**

Since: 1.1.1

The function gpgme_free releases the memory returned by gpgme_data_release_and_get_mem and gpgme_pubkey_algo_string. It should be used instead of the system libraries free function in case different allocators are used by a program. This is often the case if gpgme is used under Windows as a DLL.

### 6.3 Manipulating Data Buffers

Data buffers contain data and meta-data. The following operations can be used to manipulate both.

#### 6.3.1 Data Buffer I/O Operations

**ssize_t gpgme_data_read (gpgme_data_t dh, void *buffer, size_t length)**

The function gpgme_data_read reads up to length bytes from the data object with the handle dh into the space starting at buffer.

If no error occurs, the actual amount read is returned. If the end of the data object is reached, the function returns 0.

In all other cases, the function returns -1 and sets errno.

**ssize_t gpgme_data_write (gpgme_data_t dh, const void *buffer, size_t size)**

The function gpgme_data_write writes up to size bytes starting from buffer into the data object with the handle dh at the current write position.

The function returns the number of bytes actually written, or -1 if an error occurs. If an error occurs, errno is set.
The function `gpgme_data_seek` changes the current read/write position. The `whence` argument specifies how the `offset` should be interpreted. It must be one of the following symbolic constants:

- **SEEK_SET**: Specifies that `offset` is a count of characters from the beginning of the data object.
- **SEEK_CUR**: Specifies that `offset` is a count of characters from the current file position. This count may be positive or negative.
- **SEEK_END**: Specifies that `offset` is a count of characters from the end of the data object. A negative count specifies a position within the current extent of the data object; a positive count specifies a position past the current end. If you set the position past the current end, and actually write data, you will extend the data object with zeros up to that position.

If successful, the function returns the resulting file position, measured in bytes from the beginning of the data object. You can use this feature together with `SEEK_CUR` to read the current read/write position.

If the function fails, -1 is returned and `errno` is set.

### 6.3.2 Data Buffer Meta-Data

#### char * gpgme_data_get_file_name (gpgme_data_t dh)

**Since**: 1.1.0

The function `gpgme_data_get_file_name` returns a pointer to a string containing the file name associated with the data object. The file name will be stored in the output when encrypting or signing the data and will be returned to the user when decrypting or verifying the output data.

If no error occurs, the string containing the file name is returned. Otherwise, `NULL` will be returned.

#### gpgme_error_t gpgme_data_set_file_name (gpgme_data_t dh, const char *file_name)

**Since**: 1.1.0

The function `gpgme_data_set_file_name` sets the file name associated with the data object. The file name will be stored in the output when encrypting or signing the data and will be returned to the user when decrypting or verifying the output data. If a signed or encrypted archive is created, then the file name will be interpreted as the base directory for the relative paths of the files and directories to put into the archive. This corresponds to the `--directory` option of `gpgtar`.

The function returns the error code `GPG_ERR_INV_VALUE` if `dh` is not a valid pointer and `GPG_ERR_ENOMEM` if not enough memory is available.

#### enum gpgme_data_encoding_t

The `gpgme_data_encoding_t` type specifies the encoding of a `gpgme_data_t` object. For input data objects, the encoding is useful to give the backend a hint on the type of data. For output data objects, the encoding can specify the output data format
on certain operations. Please note that not all backends support all encodings on all operations. The following data types are available:

**GPGME_DATA_ENCODING_NONE**
This specifies that the encoding is not known. This is the default for a new data object. The backend will try its best to detect the encoding automatically.

**GPGME_DATA_ENCODING_BINARY**
This specifies that the data is encoding in binary form; i.e., there is no special encoding.

**GPGME_DATA_ENCODING_BASE64**
This specifies that the data is encoded using the Base-64 encoding scheme as used by MIME and other protocols.

**GPGME_DATA_ENCODING_ARMOR**
This specifies that the data is encoded in an armored form as used by OpenPGP and PEM.

**GPGME_DATA_ENCODING_MIME**
Since: 1.7.0
This specifies that the data is encoded as a MIME part.

**GPGME_DATA_ENCODING_URL**
Since: 1.2.0
The data is a list of linefeed delimited URLs. This is only useful with gpgme_op_import.

**GPGME_DATA_ENCODING_URL0**
Since: 1.2.0
The data is a list of binary zero delimited URLs. This is only useful with gpgme_op_import.

**GPGME_DATA_ENCODING_URLESC**
Since: 1.2.0
The data is a list of linefeed delimited URLs with all control and space characters percent escaped. This mode is is not yet implemented.

```c
[Function]
gpgme_data_encoding_t gpgme_data_get_encoding
   (gpgme_data_t dh)

The function gpgme_data_get_encoding returns the encoding of the data object with the handle dh. If dh is not a valid pointer (e.g., NULL) GPGME_DATA_ENCODING_NONE is returned.
```

```c
[Function]
gpgme_error_t gpgme_data_set_encoding
   (gpgme_data_t dh, gpgme_data_encoding_t enc)

The function gpgme_data_set_encoding changes the encoding of the data object with the handle dh to enc.
```
Function

```c
int gpgme_data_set_flag (gpgme_data_t dh,
    const char *name, const char *value);
```

**Since:** 1.7.0

Some minor properties of the data object can be controlled with flags set by this function. The properties are identified by the following values for `name`:

- **size-hint**
  The value is a decimal number with the length gpgme shall assume for this data object. This is useful if the data is provided by callbacks or via file descriptors but the applications knows the total size of the data. If this is set the OpenPGP engine may use this to decide on buffer allocation strategies and to provide a total value for its progress information.

- **io-buffer-size**
  The value is a decimal number with the length of internal buffers to use for internal I/O operations. The value is capped at 1048576 (1 MiB). In certain environments large buffers can yield a performance boost for callback bases data object, but the details depend a lot on the circumstances and the operating system. This flag may only be set once and must be set before any actual I/O happens ion the data objects.

- **sensitive**
  If the numeric value is not 0 the data object is considered to contain sensitive information like passwords or key material. If this is set the internal buffers are securely overwritten with zeroes by gpgme_data_release.

This function returns 0 on success.

### 6.3.3 Data Buffer Convenience Functions

**Enum**

```c
enum gpgme_data_type_t { [Data type]
    GPGME_DATA_TYPE_INVALID,
    GPGME_DATA_TYPE_UNKNOWN,
    GPGME_DATA_TYPE_PGP_SIGNED,
    GPGME_DATA_TYPE_PGP_ENCRYPTED,
    GPGME_DATA_TYPE_INVALID
}
```

**Since:** 1.4.3

The `gpgme_data_type_t` type is used to return the detected type of the content of a data buffer.

- **GPGME_DATA_TYPE_INVALID**
  This is returned by `gpgme_data_identify` if it was not possible to identify the data. Reasons for this might be a non-seekable stream or a memory problem. The value is 0.

- **GPGME_DATA_TYPE_UNKNOWN**
  The type of the data is not known.

- **GPGME_DATA_TYPE_PGP_SIGNED**
  The data is an OpenPGP signed message. This may be a binary signature, a detached one or a cleartext signature.

- **GPGME_DATA_TYPE_PGP_ENCRYPTED**
  **Since:** 1.7.0
  The data is an OpenPGP encrypted message.
The data is an OpenPGP detached signature.

This is a generic OpenPGP message. In most cases this will be encrypted data.

This is an OpenPGP key (private or public).

This is a CMS signed message.

This is a CMS encrypted (enveloped data) message.

This is used for other CMS message types.

The data is a X.509 certificate

The data is a PKCS#12 message. This is commonly used to exchange private keys for X.509.

The function `gpgme_data_identify` returns the type of the data with the handle `dh`. If it is not possible to perform the identification, the function returns zero (GPGME_DATA_TYPE_INVALID). Note that depending on how the data object has been created the identification may not be possible or the data object may change its internal state (file pointer moved). For file or memory based data object, the state should not change. `reserved` should be zero.
Chapter 7: Contexts

All cryptographic operations in GPGME are performed within a context, which contains the internal state of the operation as well as configuration parameters. By using several contexts you can run several cryptographic operations in parallel, with different configuration.

`gpgme_ctx_t` [Data type]

The `gpgme_ctx_t` type is a handle for a GPGME context, which is used to hold the configuration, status and result of cryptographic operations.

7.1 Creating Contexts

`gpgme_error_t gpgme_new (gpgme_ctx_t *ctx)` [Function]

The function `gpgme_new` creates a new `gpgme_ctx_t` object and returns a handle for it in `ctx`.

The function returns the error code `GPG_ERR_NO_ERROR` if the context was successfully created, `GPG_ERR_INV_VALUE` if `ctx` is not a valid pointer, and `GPG_ERR_ENOMEM` if not enough memory is available. Also, it returns `GPG_ERR_NOT_OPERATIONAL` if `gpgme_check_version` was not called to initialize GPGME, and `GPG_ERR_SELFTEST_FAILED` if a selftest failed. Currently, the only selftest is for Windows MingW32 targets to see if `-mms-bitfields` was used (as required).

7.2 Destroying Contexts

`void gpgme_release (gpgme_ctx_t ctx)` [Function]

The function `gpgme_release` destroys the context with the handle `ctx` and releases all associated resources.

7.3 Result Management

The detailed result of an operation is returned in operation-specific structures such as `gpgme_decrypt_result_t`. The corresponding retrieval functions such as `gpgme_op_decrypt_result` provide static access to the results after an operation completes. Those structures shall be considered read-only and an application must not allocate such a structure on its own. The following interfaces make it possible to detach a result structure from its associated context and give it a lifetime beyond that of the current operation or context.

`void gpgme_result_ref (void *result)` [Function]

Since: 1.2.0

The function `gpgme_result_ref` acquires an additional reference for the result `result`, which may be of any type `gpgme_*_result_t`. As long as the user holds a reference, the result structure is guaranteed to be valid and unmodified.

`void gpgme_result_unref (void *result)` [Function]

Since: 1.2.0

The function `gpgme_result_unref` releases a reference for the result `result`. If this was the last reference, the result structure will be destroyed and all resources associated to it will be released.
Note that a context may hold its own references to result structures, typically until the context is destroyed or the next operation is started. In fact, these references are accessed through the `gpgme_op_*_result` functions.

### 7.4 Context Attributes

#### 7.4.1 Protocol Selection

**gpgme_error_t gpgme_set_protocol (gpgme_ctx_t ctx, gpgme_protocol_t proto)**

The function `gpgme_set_protocol` sets the protocol used within the context `ctx` to `proto`. All crypto operations will be performed by the crypto engine configured for that protocol. See Chapter 3 [Protocols and Engines], page 10.

Setting the protocol with `gpgme_set_protocol` does intentionally not check if the crypto engine for that protocol is available and installed correctly. See Section 3.1 [Engine Version Check], page 11.

The function returns the error code `GPG_ERR_NO_ERROR` if the protocol could be set successfully, and `GPG_ERR_INV_VALUE` if `protocol` is not a valid protocol.

**gpgme_protocol_t gpgme_get_protocol (gpgme_ctx_t ctx)**

The function `gpgme_get_protocol` retrieves the protocol currently use with the context `ctx`.

#### 7.4.2 Crypto Engine

The following functions can be used to set and retrieve the configuration of the crypto engines of a specific context. The default can also be retrieved without any particular context. See Section 3.2 [Engine Information], page 12. The default can also be changed globally. See Section 3.3 [Engine Configuration], page 14.

**gpgme_engine_info_t gpgme_ctx_get_engine_info (gpgme_ctx_t ctx)**

Since: 1.1.0

The function `gpgme_ctx_get_engine_info` returns a linked list of engine info structures. Each info structure describes the configuration of one configured backend, as used by the context `ctx`.

The result is valid until the next invocation of `gpgme_ctx_set_engine_info` for this particular context.

This function can not fail.

**gpgme_error_t gpgme_ctx_set_engine_info (gpgme_ctx_t ctx, gpgme_protocol_t proto, const char *file_name, const char *home_dir)**

Since: 1.1.0

The function `gpgme_ctx_set_engine_info` changes the configuration of the crypto engine implementing the protocol `proto` for the context `ctx`.

`file_name` is the file name of the executable program implementing this protocol, and `home_dir` is the directory name of the configuration directory for this crypto engine. If `home_dir` is NULL, the engine’s default will be used.
Currently this function must be used before starting the first crypto operation. It is unspecified if and when the changes will take effect if the function is called after starting the first operation on the context ctx.

This function returns the error code GPG_ERR_NO_ERROR if successful, or an error code on failure.

### 7.4.3 How to tell the engine the sender.

Some engines can make use of the sender’s address, for example to figure out the best user id in certain trust models. For verification and signing of mails, it is thus suggested to let the engine know the sender ("From:" address). GPGME provides two functions to accomplish that. Note that the esoteric use of multiple "From:" addresses is not supported.

**Function**

```c
#include <gpgme_DISPATCH.h>

int gpgme_set_sender (gpgme_ctx_t ctx, int address);
```

The function `gpgme_set_sender` specifies the sender address for use in sign and verify operations. `address` is expected to be the “addr-spec” part of an address but may also be a complete mailbox address, in which case this function extracts the “addr-spec” from it. Using NULL for `address` clears the sender address.

The function returns 0 on success or an error code on failure. The most likely failure is that no valid “addr-spec” was found in `address`.

**Function**

```c
#include <gpgme_DISPATCH.h>

const char * gpgme_get_sender (gpgme_ctx_t ctx);
```

The function `gpgme_get_sender` returns the current sender address from the context, or NULL if none was set. The returned value is valid as long as the `ctx` is valid and `gpgme_set_sender` has not been called again.

### 7.4.4 ASCII Armor

**Function**

```c
#include <gpgme DISPATCH.h>

void gpgme_set_armor (gpgme_ctx_t ctx, int yes);
```

The function `gpgme_set_armor` specifies if the output should be ASCII armored. By default, output is not ASCII armored.

ASCII armored output is disabled if `yes` is zero, and enabled otherwise.

**Function**

```c
#include <gpgme DISPATCH.h>

int gpgme_get_armor (gpgme_ctx_t ctx);
```

The function `gpgme_get Armor` returns 1 if the output is ASCII armored, and 0 if it is not, or if `ctx` is not a valid pointer.

### 7.4.5 Text Mode

**Function**

```c
#include <gpgme DISPATCH.h>

void gpgme_set_textmode (gpgme_ctx_t ctx, int yes);
```

The function `gpgme_set_textmode` specifies if canonical text mode should be used. By default, text mode is not used.

Text mode is for example used for the RFC2015 signatures; note that the updated RFC 3156 mandates that the mail user agent does some preparations so that text mode is not needed anymore.
This option is only relevant to the OpenPGP crypto engine, and ignored by all other engines.
Canonical text mode is disabled if yes is zero, and enabled otherwise.

```c
int gpgme_get_textmode (gpgme_ctx_t ctx)
```

The function `gpgme_get_textmode` returns 1 if canonical text mode is enabled, and 0 if it is not, or if `ctx` is not a valid pointer.

### 7.4.6 Offline Mode

```c
void gpgme_set_offline (gpgme_ctx_t ctx, int yes)
```

Since: 1.6.0

The function `gpgme_set_offline` specifies if offline mode should be used. Offline mode is disabled if `yes` is zero, and enabled otherwise. By default, offline mode is disabled.

The details of the offline mode depend on the used protocol and its backend engine. It may eventually be extended to be more stricter and for example completely disable the use of Dirmngr for any engine.

For the CMS protocol the offline mode specifies whether Dirmngr shall be used to do additional validation that might require connecting external services (e.g., CRL / OCSP checks). The offline mode is used for all operations on this context. It has only an effect with GnuPG versions 2.1.6 or later.

For the OpenPGP protocol offline mode entirely disables the use of the Dirmngr and will thus guarantee that no network connections are done as part of an operation on this context. It has only an effect with GnuPG versions 2.1.23 or later.

For all other protocols the offline mode is currently ignored.

```c
int gpgme_get_offline (gpgme_ctx_t ctx)
```

Since: 1.6.0

The function `gpgme_get_offline` returns 1 if offline mode is enabled, and 0 if it is not, or if `ctx` is not a valid pointer.

### 7.4.7 Pinentry Mode

```c
gpgme_error_t gpgme_set_pinentry_mode (gpgme_ctx_t ctx, gpgme_pinentry_mode_t mode)
```

Since: 1.4.0

The function `gpgme_set_pinentry_mode` specifies the pinentry mode to be used.

For GnuPG >= 2.1 this option is required to be set to `GPGME_PINENTRY_MODE_LOOPBACK` to enable the passphrase callback mechanism in GPGME through `gpgme_set_passphrase_cb`.

```c
gpgme_pinentry_mode_t gpgme_get_pinentry_mode (gpgme_ctx_t ctx)
```

Since: 1.4.0

The function `gpgme_get_pinentry_mode` returns the mode set for the context.
enum gpgme_pinentry_mode_t

Since: 1.4.0

The gpgme_pinentry_mode_t type specifies the set of possible pinentry modes that are supported by GPGME if GnuPG >= 2.1 is used. The following modes are supported:

GPGME_PINENTRY_MODE_DEFAULT
Since: 1.4.0
Use the default of the agent, which is ask.

GPGME_PINENTRY_MODE_ASK
Since: 1.4.0
Force the use of the Pinentry.

GPGME_PINENTRY_MODE_CANCEL
Since: 1.4.0
Emulate use of Pinentry’s cancel button.

GPGME_PINENTRY_MODE_ERROR
Since: 1.4.0
Return a Pinentry error No Pinentry.

GPGME_PINENTRY_MODE_LOOPBACK
Since: 1.4.0
Redirect Pinentry queries to the caller. This enables the use of gpgme_set_passphrase_cb because pinentry queries are redirected to gpgme.

Note: For 2.1.0 - 2.1.12 this mode requires allow-loopback-pinentry to be enabled in the ‘gpg-agent.conf’ or an agent started with that option.

7.4.8 Included Certificates

void gpgme_set_include_certs (gpgme_ctx_t ctx,
int nr_of_certs)

The function gpgme_set_include_certs specifies how many certificates should be included in an S/MIME signed message. By default, only the sender’s certificate is included. The possible values of nr.of.certs are:

GPGME_INCLUDE_CERTS_DEFAULT
Since: 1.0.3
Fall back to the default of the crypto backend. This is the default for GPGME.

-2 Include all certificates except the root certificate.
-1 Include all certificates.
0 Include no certificates.
1 Include the sender’s certificate only.
n Include the first n certificates of the certificates path, starting from the sender’s certificate. The number n must be positive.
Values of \( nr_{of\_certs} \) smaller than -2 are undefined.

This option is only relevant to the CMS crypto engine, and ignored by all other engines.

```c
int gpgme_get_include_certs (gpgme_ctx_t ctx) [Function]
```

The function `gpgme_get_include_certs` returns the number of certificates to include into an S/MIME signed message.

### 7.4.9 Key Listing Mode

```c
int gpgme_set_keylist_mode (gpgme_ctx_t ctx, gpgme_keylist_mode_t mode) [Function]
```

The function `gpgme_set_keylist_mode` changes the default behaviour of the key listing functions. The value in `mode` is a bitwise-or combination of one or multiple of the following bit values:

#### GPGME_KEYLIST_MODE_LOCAL

The `GPGME_KEYLIST_MODE_LOCAL` symbol specifies that the local keyring should be searched for keys in the keylisting operation. This is the default.

Using only this option results in a `--list-keys`.

#### GPGME_KEYLIST_MODE_EXTERN

The `GPGME_KEYLIST_MODE_EXTERN` symbol specifies that an external source should be searched for keys in the keylisting operation. The type of external source is dependent on the crypto engine used and whether it is combined with `GPGME_KEYLIST_MODE_LOCAL`. For example, it can be a remote keyserver or LDAP certificate server.

Using only this option results in a `--search-keys` for `GPGME_PROTOCOL_OpenPGP` and something similar to `--list-external-keys` for `GPGME_PROTOCOL_CMS`.

#### GPGME_KEYLIST_MODE_LOCATE

This is a shortcut for the combination of `GPGME_KEYLIST_MODE_LOCAL` and `GPGME_KEYLIST_MODE_EXTERN`, which results in a `--locate-keys` for `GPGME_PROTOCOL_OpenPGP`.

#### GPGME_KEYLIST_MODE_SIGS

The `GPGME_KEYLIST_MODE_SIGS` symbol specifies that the key signatures should be included in the listed keys.

#### GPGME_KEYLIST_MODE_SIG NOTATIONS

Since: 1.1.1

The `GPGME_KEYLIST_MODE_SIG NOTATIONS` symbol specifies that the signature notations on key signatures should be included in the listed keys. This only works if `GPGME_KEYLIST_MODE_SIGS` is also enabled.

#### GPGME_KEYLIST_MODE_WITH_TOFU

Since: 1.7.0

The `GPGME_KEYLIST_MODE_WITH_TOFU` symbol specifies that information pertaining to the TOFU trust model should be included in the listed keys.
GPGME_KEYLIST_MODE_WITH_KEYGRIp
Since: 1.14.0
The GPGME_KEYLIST_MODE_WITH_KEYGRIp symbol specifies that the keygrip is always included in the listing. The default depends on the version of the backend and the used protocol.

GPGME_KEYLIST_MODE_WITH_SECRET
Since: 1.5.1
The GPGME_KEYLIST_MODE_WITH_SECRET returns information about the presence of a corresponding secret key in a public key listing. A public key listing with this mode is slower than a standard listing but can be used instead of a second run to list the secret keys. This is only supported for GnuPG versions >= 2.1. Note that using this option also makes sure that the keygrip is available in the output.

GPGME_KEYLIST_MODE_EPHEMERAL
Since: 1.2.0
The GPGME_KEYLIST_MODE_EPHEMERAL symbol specifies that keys flagged as ephemeral are included in the listing.

GPGME_KEYLIST_MODE_WITH_V5FPR
Since: 1.23.0
The GPGME_KEYLIST_MODE_WITH_V5FPR symbol specifies that key listings shall also provide v5 style fingerprints for v4 OpenPGP keys.

GPGME_KEYLIST_MODE_VALIDATE
Since: 0.4.5
The GPGME_KEYLIST_MODE_VALIDATE symbol specifies that the backend should do key or certificate validation and not just get the validity information from an internal cache. This might be an expensive operation and is in general not useful. Currently only implemented for the S/MIME backend and ignored for other backends.

GPGME_KEYLIST_MODE_FORCE_EXTERN
Since: 1.18.0
The GPGME_KEYLIST_MODE_FORCE_EXTERN symbol specifies that only external sources should be searched for keys in the keylisting operation. If used in combination with GPGME_KEYLIST_MODE_LOCATE, the keylisting results in a --locate-external-keys for GPGME_PROTOCOL_OpenPGP. The combination with GPGME_KEYLIST_MODE_LOCAL, but without GPGME_KEYLIST_MODE_EXTERN is not allowed. Currently only implemented for the OpenPGP backend and ignored for other backends.

GPGME_KEYLIST_MODE_LOCATE_EXTERNAL
Since: 1.18.0
This is a shortcut for the combination of GPGME_KEYLIST_MODE_LOCATE and GPGME_KEYLIST_MODE_FORCE_EXTERN, which results in a --locate-external-keys for GPGME_PROTOCOL_OpenPGP.
At least one of `GPGME_KEYLIST_MODE_LOCAL` and `GPGME_KEYLIST_MODE_EXTERN` must be specified. For future binary compatibility, you should get the current mode with `gpgme_get_keylist_mode` and modify it by setting or clearing the appropriate bits, and then using that calculated value in the `gpgme_set_keylisting_mode` operation. This will leave all other bits in the mode value intact (in particular those that are not used in the current version of the library).

The function returns the error code `GPG_ERR_NO_ERROR` if the mode could be set correctly, and `GPG_ERR_INV_VALUE` if `ctx` is not a valid pointer or `mode` is not a valid mode.

```c
void gpgme_set_passphrase_cb (gpgme_ctx_t ctx,
    gpgme_passphrase_cb_t passfunc, void *hook_value)
```

The function `gpgme_set_passphrase_cb` sets the function that is used when a passphrase needs to be provided by the user to `passfunc`. The function `passfunc`
needs to implemented by the user, and whenever it is called, it is called with its first argument being \textit{hook\_value}. By default, no passphrase callback function is set.

Not all crypto engines require this callback to retrieve the passphrase. It is better if the engine retrieves the passphrase from a trusted agent (a daemon process), rather than having each user to implement their own passphrase query. Some engines do not even support an external passphrase callback at all, in this case the error code \texttt{GPG\_ERR\_NOT\_SUPPORTED} is returned.

For GnuPG $\geq 2.1$ the pinentry mode has to be set to \texttt{GPGME\_PINENTRY\_MODE\_LOOPBACK} to enable the passphrase callback. See \texttt{gpgme\_set\_pinentry\_mode}.

The user can disable the use of a passphrase callback function by calling \texttt{gpgme\_set\_passphrase\_cb} with \texttt{passfunc} being \texttt{NULL}.

\begin{verbatim}
void gpgme_get_passphrase_cb (gpgme_ctx_t ctx, [Function]
gpgme_passphrase_cb_t *passfunc, void **hook_value)
\end{verbatim}

The function \texttt{gpgme\_get\_passphrase\_cb} returns the function that is used when a passphrase needs to be provided by the user in \texttt{*passfunc}, and the first argument for this function in \texttt{*hook\_value}. If no passphrase callback is set, or \texttt{ctx} is not a valid pointer, \texttt{NULL} is returned in both variables.

\texttt{passfunc} or \texttt{hook\_value} can be \texttt{NULL}. In this case, the corresponding value will not be returned.

### 7.4.11 Progress Meter Callback

\begin{verbatim}
void (*gpgme_progress_cb_t)(void *hook, const char *what, [Data type]
int type, int current, int total)
\end{verbatim}

The \texttt{gpgme\_progress\_cb\_t} type is the type of functions usable as progress callback function.

The arguments are specific to the crypto engine. More information about the progress information returned from the GnuPG engine can be found in the GnuPG source code in the file ‘doc/DETAILS’ in the section PROGRESS.

\begin{verbatim}
void gpgme_set_progress_cb (gpgme_ctx_t ctx, [Function]
gpgme_progress_cb_t proffunc, void *hook_value)
\end{verbatim}

The function \texttt{gpgme\_set\_progress\_cb} sets the function that is used when progress information about a cryptographic operation is available. The function \texttt{proffunc} needs to implemented by the user, and whenever it is called, it is called with its first argument being \texttt{hook\_value}. By default, no progress callback function is set.

Setting a callback function allows an interactive program to display progress information about a long operation to the user.

The user can disable the use of a progress callback function by calling \texttt{gpgme\_set\_progress\_cb} with \texttt{proffunc} being \texttt{NULL}.

\begin{verbatim}
void gpgme_get_progress_cb (gpgme_ctx_t ctx, [Function]
gpgme_progress_cb_t *proffunc, void **hook_value)
\end{verbatim}

The function \texttt{gpgme\_get\_progress\_cb} returns the function that is used to inform the user about the progress made in \texttt{*proffunc}, and the first argument for this function
in *hook_value. If no progress callback is set, or ctx is not a valid pointer, NULL is returned in both variables.

`progfunc` or `hook_value` can be NULL. In this case, the corresponding value will not be returned.

### 7.4.12 Status Message Callback

**gpgme_error_t (**gpgme_status_cb_t**(void *hook, const char *keyword, const char *args))**

The `gpgme_status_cb_t` type is the type of function usable as a status message callback function.

The argument `keyword` is the name of the status message while the `args` argument contains any arguments for the status message.

If an error occurs, return the corresponding `gpgme_error_t` value. Otherwise, return 0.

**void gpgme_set_status_cb (gpgme_ctx_t ctx,**
  
  gpgme_status_cb_t statusfunc, void *hook_value)**

**Since:** 1.6.0

The function `gpgme_set_status_cb` sets the function that is used when a status message is received from gpg to `statusfunc`. The function `statusfunc` needs to be implemented by the user, and whenever it is called, it is called with its first argument being `hook_value`. By default, no status message callback function is set.

The user can disable the use of a status message callback function by calling `gpgme_set_status_cb` with `statusfunc` being NULL.

**void gpgme_get_status_cb (gpgme_ctx_t ctx,**
  
  gpgme_status_cb_t *statusfunc, void **hook_value)**

**Since:** 1.6.0

The function `gpgme_get_status_cb` returns the function that is used to process status messages from gpg in `statusfunc`, and the first argument for this function in `*hook_value`. If no status message callback is set, or `ctx` is not a valid pointer, NULL is returned in both variables.

### 7.4.13 Context Flags

**gpgme_error_t gpgme_set_ctx_flag (gpgme_ctx_t ctx,**
  
  const char *name, const char *value)**

**Since:** 1.7.0

Some minor properties of the context can be controlled with flags set by this function. The properties are identified by the following values for `name`:

"redraw" This flag is normally not changed by the caller because GPGME sets and clears it automatically: The flag is cleared before an operation and set if an operation noticed that the engine has launched a Pinentry. A Curses based application may use this information to redraw the screen; for example:
err = gpgme_op_keylist_start (ctx, "foo@example.org", 0);
while (!err)
{
    err = gpgme_op_keylist_next (ctx, &key);
    if (err)
        break;
    show_key (key);
    gpgme_key_release (key);
}
if ((s = gpgme_get_ctx_flag (ctx, "redraw")) && *s)
    redraw_screen ();
gpgme_release (ctx);

"full-status"
Using a value of "1" the status callback set by gpgme_set_status_cb returns all status lines with the exception of PROGRESS lines. With the default of "0" the status callback is only called in certain situations.

"raw-description"
Setting the value to "1" returns human readable strings in a raw format. For example the non breaking space characters ("\-\") will not be removed from the description field of the gpgme_tofu_info_t object.

"export-session-key"
Using a value of "1" specifies that the context should try to export the symmetric session key when decrypting data. By default, or when using an empty string or "0" for value, session keys are not exported.

"override-session-key"
The string given in value is passed to the GnuPG engine to override the session key for decryption. The format of that session key is specific to GnuPG and can be retrieved during a decrypt operation when the context flag "export-session-key" is enabled. Please be aware that using this feature with GnuPG < 2.1.16 or when decrypting an archive will leak the session key on many platforms via ps(1).

"auto-key-retrieve"
Setting the value to "1" asks the backend to automatically retrieve a key for signature verification if possible. 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.

"auto-key-import"
Setting the value to "1" forces the GPG backend to automatically import a missing key for signature verification from the signature.
"include-key-block"
Setting the value to "1" forces the GPG backend to embed the signing key as well as an encryption subkey into the the signature.

"request-origin"
The string given in value is passed to the GnuPG engines to request restrictions based on the origin of the request. Valid values are documented in the GnuPG manual and the gpg man page under the option "--request-origin". Requires at least GnuPG 2.2.6 to have an effect.

"no-symkey-cache"
For OpenPGP disable the passphrase cache used for symmetrical en- and decryption. This cache is based on the message specific salt value. Requires at least GnuPG 2.2.7 to have an effect.

"ignore-mdc-error"
This flag passes the option '--ignore-mdc-error' to gpg. This can be used to force decryption of a message which failed due to a missing integrity check. This flag must be used with great caution and only if it is a known non-corrupted old message and the decryption result of the former try had the decryption result flag legacy_cipher_nomdc set. For failsafe reasons this flag is reset after each operation.

"auto-key-locate"
The string given in value is passed to gpg. This can be used to change the behavior of a GPGME_KEYLIST_MODE_LOCATE keylisting. Valid values are documented in the GnuPG manual and the gpg man page under the option '--auto-key-locate'. Requires at least GnuPG 2.1.18.
Note: Keys retrieved through auto-key-locate are automatically imported in the keyring.

trust-model
Since: 1.11.2
Change the trust-model for all GnuPG engine operations. An empty string sets the trust-model back to the users default. If the trust-model is not supported by GnuPG the behavior is undefined and will likely cause all operations to fail. Example: "tofu+pgp".
This options should be used carefully with a strict version requirement. In some versions of GnuPG setting the trust-model changes the default trust-model for future operations. A change in the trust-model also can have unintended side effects, like rebuilding the trust-db.

"extended-edit"
This flag passes the option '--expert' to gpg key edit. This can be used to get additional callbacks in gpgme_op_edit.

"cert-expire"
Since: 1.15.2 The string given in value is passed to the GnuPG engine to set the expiration time to use for key signature expiration. Valid values are documented in the GnuPG manual and the gpg man page under the option '--default-cert-expire'.
"key-origin"

Since: 1.16.1 The string given in value is passed to the GnuPG engine to set the origin of imported keys. Valid values are documented in the GnuPG manual and the gpg man page under the option ‘--key-origin’.

"import-filter"

Since: 1.16.1 The string given in value is passed to the GnuPG engine to use as filter when importing keys. Valid values are documented in the GnuPG manual and the gpg man page under the option ‘--import-filter’.

"no-auto-check-trustdb"

Since: 1.19.0 Setting the value to "1" forces the GPG backend to disable the automatic check of the trust database.

This function returns 0 on success.

const char * gpgme_get_ctx_flag (gpgme_ctx_t ctx, const char *name)

Since: 1.8.0

The value of flags settable by gpgme_set_ctx_flag can be retrieved by this function. If name is unknown the function returns NULL. For boolean flags an empty string is returned for False and the string "1" is returned for True; either atoi(3) or a test for an empty string can be used to get the boolean value.

7.4.14 Locale

A locale setting can be associated with a context. This locale is passed to the crypto engine, and used for applications like the PIN entry, which is displayed to the user when entering a passphrase is required.

The default locale is used to initialize the locale setting of all contexts created afterwards.

gpgme_error_t gpgme_set_locale (gpgme_ctx_t ctx, int category, const char *value)

Since: 0.4.3

The function gpgme_set_locale sets the locale of the context ctx, or the default locale if ctx is a null pointer.

The locale settings that should be changed are specified by category. Supported categories are LC_CTYPE, LC_MESSAGES, and LC_ALL, which is a wildcard you can use if you want to change all the categories at once.

The value to be used for the locale setting is value, which will be copied to GPGME’s internal data structures. value can be a null pointer, which disables setting the locale, and will make PIN entry and other applications use their default setting, which is usually not what you want.

Note that the settings are only used if the application runs on a text terminal, and that the settings should fit the configuration of the output terminal. Normally, it is sufficient to initialize the default value at startup.

The function returns an error if not enough memory is available.
7.4.15 Additional Logs

Additional logs can be associated with a context. These logs are engine specific and can be obtained with `gpgme_op_getauditlog`.

```c
[gpgme_error_t gpgme_op_getauditlog (gpgme_ctx_t ctx,
gpgme_data_t output, unsigned int flags)
```

Since: 1.1.1

The function `gpgme_op_getauditlog` is used to obtain additional logs as specified by `flags` into the `output` data. If

The function returns the error code `GPG_ERR_NO_ERROR` if a log could be queried from the engine, and `GPG_ERR_NOT_IMPLEMENTED` if the log specified in `flags` is not available for this engine. If no log is available `GPG_ERR_NO_DATA` is returned.

The value in `flags` is a bitwise-or combination of one or multiple of the following bit values:

- **GPGME_AUDITLOG_DIAG**
  - Since: 1.11.2
  - Obtain diagnostic output which would be written to `stderr` in interactive use of the engine. This can be used to provide additional diagnostic information in case of errors in other operations.
  - Note: If log-file has been set in the configuration the log will be empty and `GPG_ERR_NO_DATA` will be returned.
  - Implemented for: `GPGME_PROTOCOL_OpenPGP`

- **GPGME_AUDITLOG_DEFAULT**
  - Since: 1.11.2
  - This flag has the value 0 for compatibility reasons. Obtains additional information from the engine by issuing the `GETAUDITLOG` command. For `GPGME_PROTOCOL_CMS` this provides additional information about the X509 certificate chain.
  - Implemented for: `GPGME_PROTOCOL_CMS`

- **GPGME_AUDITLOG_HTML**
  - Since: 1.1.1
  - Same as `GPGME_AUDITLOG_DEFAULT` but in HTML.
  - Implemented for: `GPGME_PROTOCOL_CMS`

```c
[gpgme_error_t gpgme_op_getauditlog_start (gpgme_ctx_t ctx,
gpgme_data_t output, unsigned int flags)
```

Since: 1.1.1

This is the asynchronous variant of `gpgme_op_getauditlog`.

7.5 Key Management

Some of the cryptographic operations require that recipients or signers are specified. This is always done by specifying the respective keys that should be used for the operation. The following section describes how such keys can be selected and manipulated.
7.5.1 Key objects

The keys are represented in GPGME by structures which may only be read by the application but never be allocated or changed. They are valid as long as the key object itself is valid.

`gpgme_key_t`

The `gpgme_key_t` type is a pointer to a key object. It has the following members:

- `gpgme_keylist_mode_t keylist_mode`
  
  Since: 0.9.0
  
  The keylist mode that was active when the key was retrieved.

- `unsigned int revoked : 1`
  
  This is true if the key is revoked.

- `unsigned int expired : 1`
  
  This is true if the key is expired.

- `unsigned int disabled : 1`
  
  This is true if the key is disabled.

- `unsigned int invalid : 1`
  
  This is true if the key is invalid. This might have several reasons, for example for the S/MIME backend, it will be set during key listings if the key could not be validated due to missing certificates or unmatched policies.

- `unsigned int can_encrypt : 1`
  
  This is true if the key or one of its subkeys can be used for encryption and the encryption will likely succeed.

- `unsigned int can_sign : 1`
  
  This is true if the key or one of its subkeys can be used to create data signatures and the signing will likely succeed.

- `unsigned int can_certify : 1`
  
  This is true if the key or one of its subkeys can be used to create key certificates.

- `unsigned int can_authenticate : 1`
  
  Since: 0.4.5
  
  This is true if the key (ie one of its subkeys) can be used for authentication and the authentication will likely succeed.

- `unsigned int has_encrypt : 1`
  
  Since: 1.23.0
  
  This is true if the key or one of its subkeys is capable of encryption. Note that this flag is set even if the key is expired.

- `unsigned int has_sign : 1`
  
  Since: 1.23.0
  
  This is true if the key or one of its subkeys is capable of signing. Note that this flag is set even if the key is expired.
unsigned int has_certify : 1
    Since: 1.23.0
    This is true if the key or one of its subkeys is capable of certification.
    Note that this flag is set even if the key is expired.

unsigned int has_authenticate : 1
    Since: 1.23.0
    This is true if the key or one of its subkeys is capable of authentication.
    Note that this flag is set even if the key is expired.

unsigned int is_qualified : 1
    Since: 1.1.0
    This is true if the key can be used for qualified signatures according to
    local government regulations.

unsigned int secret : 1
    This is true if the key is a secret key. Note, that this will always be true
    even if the corresponding subkey flag may be false (offline/stub keys).
    This is only set if a listing of secret keys has been requested or if
    GPGME_KEYLIST_MODE_WITH_SECRET is active.

unsigned int origin : 5
    Since: 1.8.0
    Reserved for the origin of this key.

gpgme_protocol_t protocol
    This is the protocol supported by this key.

char *issuer_serial
    If protocol is GPGME_PROTOCOL_CMS, then this is the issuer serial.

char *issuer_name
    If protocol is GPGME_PROTOCOL_CMS, then this is the issuer name.

char *chain_id
    If protocol is GPGME_PROTOCOL_CMS, then this is the chain ID, which can
    be used to built the certificate chain.

gpgme_validity_t owner_trust
    If protocol is GPGME_PROTOCOL_OpenPGP, then this is the owner trust.

gpgme_subkey_t subkeys
    This is a linked list with the subkeys of the key. The first subkey in the
    list is the primary key and usually available.

gpgme_user_id_t uids
    This is a linked list with the user IDs of the key. The first user ID in the
    list is the main (or primary) user ID.

char *fpr
    Since: 1.7.0
    This field gives the fingerprint of the primary key. Note that this is a copy
    of the fingerprint of the first subkey. For an incomplete key (for example
    from a verification result) a subkey may be missing but this field may be
    set nevertheless.
unsigned long last_update
  Since: 1.8.0
  Reserved for the time of the last update of this key.

__gpgme_subkey_t__
  [Data type]
  Since: 1.5.0
  The __gpgme_subkey_t__ type is a pointer to a subkey structure. Subkeys are one component of a __gpgme_key_t__ object. In fact, subkeys are those parts that contains the real information about the individual cryptographic keys that belong to the same key object. One __gpgme_key_t__ can contain several subkeys. The first subkey in the linked list is also called the primary key.
  The subkey structure has the following members:

  __gpgme_subkey_t next__
  This is a pointer to the next subkey structure in the linked list, or **NULL** if this is the last element.

  unsigned int revoked : 1
  This is true if the subkey is revoked.

  unsigned int expired : 1
  This is true if the subkey is expired.

  unsigned int disabled : 1
  This is true if the subkey is disabled.

  unsigned int invalid : 1
  This is true if the subkey is invalid.

  unsigned int can_encrypt : 1
  This is true if the subkey can be used for encryption.

  unsigned int can_sign : 1
  This is true if the subkey can be used to create data signatures.

  unsigned int can_certify : 1
  This is true if the subkey can be used to create key certificates.

  unsigned int can_authenticate : 1
    Since: 0.4.5
    This is true if the subkey can be used for authentication.

  unsigned int is_qualified : 1
    Since: 1.1.0
    This is true if the subkey can be used for qualified signatures according to local government regulations.

  unsigned int is_de_vs : 1
    Since: 1.8.0
    This is true if the subkey complies with the rules for classified information in Germany at the restricted level (VS-NfD). This are currently RSA keys of at least 2048 bits or ECDH/ECDSA keys using a Brainpool curve.
unsigned int secret : 1
This is true if the subkey is a secret key. Note that it will be false if the key is actually a stub key; i.e., a secret key operation is currently not possible (offline-key). This is only set if a listing of secret keys has been requested or if GPGME_KEYLIST_MODE_WITH_SECRET is active.

gpgme_pubkey_algo_t pubkey_algo
This is the public key algorithm supported by this subkey.

unsigned int length
This is the length of the subkey (in bits).

char *keyid
This is the key ID of the subkey in hexadecimal digits.

char *fpr
This is the fingerprint of the subkey in hexadecimal digits, if available.

char *v5fpr
For a v4 OpenPGP key this is its v5 style fingerprint of the subkey in hexadecimal digits, if available.

char *keygrip
Since: 1.7.0
The keygrip of the subkey in hex digit form or NULL if not available.

long int timestamp
This is the creation timestamp of the subkey. This is -1 if the timestamp is invalid, and 0 if it is not available.

long int expires
This is the expiration timestamp of the subkey, or 0 if the subkey does not expire.

unsigned int is_cardkey : 1
Since: 1.2.0
True if the secret key is stored on a smart card.

char *card_number
Since: 1.2.0
The serial number of a smart card holding this key or NULL.

char *curve
For ECC algorithms the name of the curve.

gpgme_user_id_t [Data type]
A user ID is a component of a gpgme_key_t object. One key can have many user IDs. The first one in the list is the main (or primary) user ID.
The user ID structure has the following members.

gpgme_user_id_t next
This is a pointer to the next user ID structure in the linked list, or NULL if this is the last element.
unsigned int revoked : 1
This is true if the user ID is revoked.

unsigned int invalid : 1
This is true if the user ID is invalid.

gpgme_validity_t validity
This specifies the validity of the user ID.

char *uid
This is the user ID string.

cchar *name
This is the name component of uid, if available.

cchar *comment
This is the comment component of uid, if available.

cchar *email
This is the email component of uid, if available.

cchar *address;
The mail address (addr-spec from RFC-5322) of the user ID string. This is general the same as the email part of this structure but might be slightly different. If no mail address is available NULL is stored.

gpgme_tofu_info_t tofu
SINCE: 1.7.0
If not NULL information from the TOFU database pertaining to this user id.

gpgme_key_sig_t signatures
This is a linked list with the signatures on this user ID.

unsigned int origin : 5
SINCE: 1.8.0
Reserved for the origin of this user ID.

unsigned long last_update
SINCE: 1.8.0
Reserved for the time of the last update of this user ID.

cchar *uidhash;
A string used by gpg to identify a user ID. This string can be used at certain prompts of gpgme_op_edit to select a user ID. Users must be prepared to see a NULL value here. The format of the value is not specified and may depend on the GPGME or GnuPG version.

gpgme_tofu_info_t
SINCE: 1.7.0
The gpgme_tofu_info_t type is a pointer to a tofu info structure. Tofu info structures are one component of a gpgme_user_id_t object, and provide information from the TOFU database pertaining to the user ID.
The tofu info structure has the following members:
gpgme_tofu_info_t next
This is a pointer to the next tofu info structure in the linked list, or NULL if this is the last element.

unsigned int validity : 3
This is the TOFU validity. It can have the following values:
0 The value 0 indicates a conflict.
1 The value 1 indicates a key without history.
2 The value 2 indicates a key with too little history.
3 The value 3 indicates a key with enough history for basic trust.
4 The value 4 indicates a key with a lot of history.

unsigned int policy : 4
This is the TOFU policy, see gpgme_tofu_policy_t.

unsigned short signcount
This is the number of signatures seen for this binding (or USHRT_MAX if there are more than that).

unsigned short encrcount
This is the number of encryptions done with this binding (or USHRT_MAX if there are more than that).

unsigned long signfirst
Number of seconds since Epoch when the first signature was seen with this binding.

unsigned long signlast
Number of seconds since Epoch when the last signature was seen with this binding.

unsigned long encrfirst
Number of seconds since Epoch when the first encryption was done with this binding.

unsigned long encrlast
Number of seconds since Epoch when the last encryption was done with this binding.

char *description
A human-readable string summarizing the TOFU data (or NULL).

gpgme_key_sig_t
The gpgme_key_sig_t type is a pointer to a key signature structure. Key signatures are one component of a gpgme_key_t object, and validate user IDs on the key in the OpenPGP protocol.

The signatures on a key are only available if the key was retrieved via a listing operation with the GPGME_KEYLIST_MODE_SIGS mode enabled, because it can be expensive to retrieve all signatures of a key.
The signature notations on a key signature are only available if the key was retrieved via a listing operation with the `GPGME_KEYLIST_MODE_SIG_NOTATIONS` mode enabled, because it can be expensive to retrieve all signature notations.

The key signature structure has the following members:

```c
struct gpgme_key_sig_t {
    struct gpgme_key_sig_t *next; /* Pointer to the next key signature in the linked list, or NULL if this is the last element. */
    unsigned int revoked; /* Indicates if the key signature is a revocation signature. */
    unsigned int expired; /* Indicates if the key signature is expired. */
    unsigned int invalid; /* Indicates if the key signature is invalid. */
    unsigned int exportable; /* Indicates if the key signature is exportable. */
    unsigned int trust_depth; /* Depth of a trust signature, or 0 if the key signature is not a trust signature. */
    unsigned int trust_value; /* Trust amount of a trust signature. */
    enum gpgme_pubkey_algo_t pubkey_algo; /* Public key algorithm used to create the signature. */
    const char *keyid; /* Key ID of the key (in hexadecimal digits) used to create the signature. */
    long int timestamp; /* Creation timestamp of the key signature. */
    long int expires; /* Expiration timestamp of the key signature. */
    const char *trust_scope; /* Regular expression that limits the scope of a trust signature. */
    gpgme_error_t status; /* Status of the signature. */
    unsigned int sig_class; /* Signature class of the key signature. */
};
```

This structure contains members that describe the state and properties of a key signature, including whether it is revoked, expired, or invalid, whether it is exportable, the depth and value of trust signatures, the public key algorithm used, the key ID, creation and expiration timestamps, a trusted scope, and the status of the signature. The `sig_class` member specifies the signature class, which has a specific meaning dependent on the crypto engine being used.
char *uid  This is the main user ID of the key used to create the signature.

char *name
This is the name component of uid, if available.

char *comment
This is the comment component of uid, if available.

char *email
This is the email component of uid, if available.

gpgme_sig_notation_t notations
This is a linked list with the notation data and policy URLs.

7.5.2 Listing Keys

gpgme_error_t gpgme_op_keylist_start (gpgme_ctx_t ctx,
const char *pattern, int secret_only)
The function gpgme_op_keylist_start initiates a key listing operation inside the
context ctx. It sets everything up so that subsequent invocations of gpgme_op_keylist_next
return the keys in the list.

If pattern is NULL, all available keys are returned. Otherwise, pattern contains an
engine specific expression that is used to limit the list to all keys matching the pattern.
Note that the total length of the pattern is restricted to an engine-specific maximum
(a couple of hundred characters are usually accepted). The pattern should be used to
restrict the search to a certain common name or user, not to list many specific keys
at once by listing their fingerprints or key IDs.

If secret_only is not 0, the list is restricted to secret keys only.

The context will be busy until either all keys are received (and gpgme_op_keylist_next returns GPG_ERR_EOF), or gpgme_op_keylist_end is called to finish the operation.

The function returns the error code GPG_ERR_INV_VALUE if ctx is not a valid pointer,
and passes through any errors that are reported by the crypto engine support routines.

gpgme_error_t gpgme_op_keylist_ext_start (gpgme_ctx_t ctx,
const char *pattern[], int secret_only, int reserved)
The function gpgme_op_keylist_ext_start initiates an extended key listing operation inside the context ctx. It sets everything up so that subsequent invocations of gpgme_op_keylist_next return the keys in the list.

If pattern or *pattern is NULL, all available keys are returned. Otherwise, pattern is
a NULL terminated array of strings that are used to limit the list to all keys matching
at least one of the patterns verbatim. Note that the total length of all patterns is
restricted to an engine-specific maximum (the exact limit also depends on the number
of patterns and amount of quoting required, but a couple of hundred characters are
usually accepted). Patterns should be used to restrict the search to a certain common
name or user, not to list many specific keys at once by listing their fingerprints or key
IDs.

If secret_only is not 0, the list is restricted to secret keys only.
The value of reserved must be 0.
The context will be busy until either all keys are received (and \texttt{gpgme\_op\_keylist\_next} returns \texttt{GPG\_ERR\_EOF}), or \texttt{gpgme\_op\_keylist\_end} is called to finish the operation.
The function returns the error code \texttt{GPG\_ERR\_INV\_VALUE} if \texttt{ctx} is not a valid pointer, and passes through any errors that are reported by the crypto engine support routines.

\begin{Verbatim}
\texttt{gpgme\_error\_t gpgme\_op\_keylist\_from\_data\_start} \hfill \text {[Function]}
\texttt{(gpgme\_ctx\_t ctx, gpgme\_data\_t data, int reserved)}
\end{Verbatim}
\textit{Since}: 1.8.0
The function \texttt{gpgme\_op\_keylist\_from\_data\_start} initiates a key listing operation inside the context \texttt{ctx}. In contrast to the other key listing operation the keys are read from the supplied \texttt{data} and not from the local key database. The keys are also not imported into the local key database. The function sets everything up so that subsequent invocations of \texttt{gpgme\_op\_keylist\_next} return the keys from \texttt{data}.
The value of reserved must be 0.
This function requires at least GnuPG version 2.1.14 and currently works only with OpenPGP keys.
The context will be busy until either all keys are received (and \texttt{gpgme\_op\_keylist\_next} returns \texttt{GPG\_ERR\_EOF}), or \texttt{gpgme\_op\_keylist\_end} is called to finish the operation. While the context is busy \texttt{data} may not be released.
The function returns the error code \texttt{GPG\_ERR\_INV\_VALUE} if \texttt{ctx} is not a valid pointer, and passes through any errors that are reported by the crypto engine support routines.

\begin{Verbatim}
\texttt{gpgme\_error\_t gpgme\_op\_keylist\_next} \hfill \text {[Function]}
\texttt{(gpgme\_ctx\_t ctx, gpgme\_key\_t \*r\_key)}
\end{Verbatim}
The function \texttt{gpgme\_op\_keylist\_next} returns the next key in the list created by a previous \texttt{gpgme\_op\_keylist\_start} operation in the context \texttt{ctx}. The key will have one reference for the user. See Section 7.5.4 [Manipulating Keys], page 57.
This is the only way to get at \texttt{gpgme\_key\_t} objects in GPGME.
If the last key in the list has already been returned, \texttt{gpgme\_op\_keylist\_next} returns \texttt{GPG\_ERR\_EOF}.
The function returns the error code \texttt{GPG\_ERR\_INV\_VALUE} if \texttt{ctx} or \texttt{r\_key} is not a valid pointer, and \texttt{GPG\_ERR\_ENOMEM} if there is not enough memory for the operation.

\begin{Verbatim}
\texttt{gpgme\_error\_t gpgme\_op\_keylist\_end} \hfill \text {[Function]}
\texttt{(gpgme\_ctx\_t ctx)}
\end{Verbatim}
The function \texttt{gpgme\_op\_keylist\_end} ends a pending key list operation in the context \texttt{ctx}.
After the operation completed successfully, the result of the key listing operation can be retrieved with \texttt{gpgme\_op\_keylist\_result}.
The function returns the error code \texttt{GPG\_ERR\_INV\_VALUE} if \texttt{ctx} is not a valid pointer, and \texttt{GPG\_ERR\_ENOMEM} if at some time during the operation there was not enough memory available.

The following example illustrates how all keys containing a certain string (\texttt{g10code}) can be listed with their key ID and the name and email address of the main user ID:
gpgme_ctx_t ctx;
gpgme_key_t key;
gpgme_error_t err = gpgme_new (&ctx);

if (!err)
{
    err = gpgme_op_keylist_start (ctx, "g10code", 0);
    while (!err)
    {
        err = gpgme_op_keylist_next (ctx, &key);
        if (err)
            break;
        printf ("%s:", key->subkeys->keyid);
        if (key->uids && key->uids->name)
            printf (" %s", key->uids->name);
        if (key->uids && key->uids->email)
            printf (" <%s>", key->uids->email);
        putchar (‘\n’);
        gpgme_key_release (key);
    }
    gpgme_release (ctx);
}
if (gpg_err_code (err) != GPG_ERR_EOF)
{
    fprintf (stderr, "can not list keys: %s\n", gpgme_strerror (err));
    exit (1);
}

gpgme_keylist_result_t [Data type]

This is a pointer to a structure used to store the result of a gpgme_op_keylist_* operation. After successfully ending a key listing operation, you can retrieve the pointer to the result with gpgme_op_keylist_result. The structure contains the following member:

unsigned int truncated : 1

This is true if the crypto backend had to truncate the result, and less than the desired keys could be listed.

gpgme_keylist_result_t gpgme_op_keylist_result [Function]

(gpgme_ctx_t ctx)

The function gpgme_op_keylist_result returns a gpgme_keylist_result_t pointer to a structure holding the result of a gpgme_op_keylist_* operation. The pointer is only valid if the last operation on the context was a key listing operation, and if this operation finished successfully. The returned pointer is only valid until the next operation is started on the context.

In a simple program, for which a blocking operation is acceptable, the following function can be used to retrieve a single key.
The function `gpgme_get_key` gets the key with the fingerprint (or key ID) `fpr` from the crypto backend and return it in `r_key`. If `secret` is true, get the secret key. The currently active keylist mode is used to retrieve the key. The key will have one reference for the user.

If the key is not found in the keyring, `gpgme_get_key` returns the error code `GPG_ERR_EOF` and `*r_key` will be set to `NULL`.

The function returns the error code `GPG_ERR_INV_VALUE` if `ctx` or `r_key` is not a valid pointer or `fpr` is not a fingerprint or key ID, `GPG_ERR_AMBIGUOUS_NAME` if the key ID was not a unique specifier for a key, and `GPG_ERR_ENOMEM` if at some time during the operation there was not enough memory available.

### 7.5.3 Information About Keys

Please see the beginning of this section for more information about `gpgme_key_t` objects.

The `gpgme_validity_t` type is used to specify the validity of a user ID in a key. The following validities are defined:

- **GPGME_VALIDITY_UNKNOWN**
  The user ID is of unknown validity. The string representation of this validity is “?”.

- **GPGME_VALIDITY_UNDEFINED**
  The validity of the user ID is undefined. The string representation of this validity is “q”.

- **GPGME_VALIDITY_NEVER**
  The user ID is never valid. The string representation of this validity is “n”.

- **GPGME_VALIDITY_MARGINAL**
  The user ID is marginally valid. The string representation of this validity is “m”.

- **GPGME_VALIDITY_FULL**
  The user ID is fully valid. The string representation of this validity is “f”.

- **GPGME_VALIDITY_ULTIMATE**
  The user ID is ultimately valid. The string representation of this validity is “u”.

### 7.5.4 Manipulating Keys

The function `gpgme_key_ref` acquires an additional reference for the key `key`. 
void gpgme_key_unref (gpgme_key_t key)  
The function gpgme_key_unref releases a reference for the key key. If this was the last reference, the key will be destroyed and all resources associated to it will be released.

gpgme_error_t gpgme_op_setexpire (gpgme_ctx_t ctx,  
gpgme_key_t key, unsigned long expires, const char *subfprs,  
unsigned int reserved);  
SINCE: 1.14.1  
The function gpgme_op_setexpire sets the expiration time of the key key or of the specified subkeys. This function requires at least version 2.1.22 of GnuPG.  
key specifies the key to operate on.  
expires specifies the expiration time in seconds from now. To be similar to other usages where expiration times are provided in unsigned long this is similar to the key creation date and so it is in seconds from NOW.  
The common case is to use 0 to not set an expiration time. Note that this parameter takes an unsigned long value and not a time_t to avoid problems on systems which use a signed 32 bit time_t. Note further that the OpenPGP protocol uses 32 bit values for timestamps and thus can only encode dates up to the year 2106.  
subfprs selects the subkey(s) for which the expiration time should be set. If subfprs is set to NULL, then the expiration time of the primary key is set. If subfprs is an asterisk (*), then the expiration times of all non-revoked and not yet expired subkeys are set. To select more than one subkey put all subkey fingerprints into one string separated by linefeeds characters (\n).  
reserved is reserved for later use and must be 0.

gpgme_error_t gpgme_op_setexpire_start (gpgme_ctx_t ctx,  
gpgme_key_t key, unsigned long expires, const char *subfprs,  
unsigned int flags);  
SINCE: 1.14.1  
The function gpgme_op_setexpire_start initiates a gpgme_op_setexpire operation; see there for details. It must be completed by calling gpgme_wait on the context. See Section 7.8.1 [Waiting For Completion], page 100.

7.5.5 Generating Keys
GPGME provides a set of functions to create public key pairs. Most of these functions require the use of GnuPG 2.1 and later; for older GnuPG versions the gpgme_op_genkey function can be used. Existing code which wants to update to the new functions or new code which shall supports older GnuPG versions may try the new functions first and provide a fallback to the old function if the error code GPG_ERR_NOT_SUPPORTED is received.

gpgme_error_t gpgme_op_createkey (gpgme_ctx_t ctx,  
const char *userid, const char *algo, unsigned long reserved,  
unsigned long expires, gpgme_key_t extrakey, unsigned int flags);  
SINCE: 1.7.0
The function `gpgme_op_createkey` generates a new key for the protocol active in the context `ctx`. As of now this function does only work for OpenPGP and requires at least version 2.1.13 of GnuPG.

`userid` is commonly the mail address associated with the key. GPGME does not require a specific syntax but if more than a mail address is given, RFC-822 style format is suggested. The value is expected to be in UTF-8 encoding (i.e., no IDN encoding for mail addresses). This is a required parameter.

`algo` specifies the algorithm for the new key (actually a keypair of public and private key). For a list of supported algorithms, see the GnuPG manual. If `algo` is `NULL` or the string "default", the key is generated using the default algorithm of the engine. If the string "future-default" is used the engine may use an algorithm which is planned to be the default in a future release of the engine; however existing implementation of the protocol may not be able to already handle such future algorithms. For the OpenPGP protocol, the specification of a default algorithm, without requesting a non-default usage via `flags`, triggers the creation of a primary key plus a secondary key (subkey).

`reserved` must be set to zero.

`expires` specifies the expiration time in seconds. If you supply 0, a reasonable expiration time is chosen. Use the flag `GPGME_CREATE_NOEXPRIE` to create keys that do not expire. Note that this parameter takes an unsigned long value and not a `time_t` to avoid problems on systems which use a signed 32 bit `time_t`. Note further that the OpenPGP protocol uses 32 bit values for timestamps and thus can only encode dates up to the year 2106.

`extrakey` is currently not used and must be set to `NULL`. A future version of GPGME may use this parameter to create X.509 keys.

`flags` can be set to the bit-wise OR of the following flags:

- `GPGME_CREATE_SIGN`  
- `GPGME_CREATE_ENCR`  
- `GPGME_CREATE_CERT`  
- `GPGME_CREATE_AUTH`  

Since: 1.7.0

Do not create the key with the default capabilities (key usage) of the requested algorithm but use those explicitly given by these flags: “signing”, “encryption”, “certification”, or “authentication”. The allowed combinations depend on the algorithm.

If any of these flags are set and a default algorithm has been selected only one key is created in the case of the OpenPGP protocol.

- `GPGME_CREATE_NOPASSWD`  
  Since: 1.7.0

Request generation of the key without password protection.

- `GPGME_CREATE_SELFSIGNED`  
  Since: 1.7.0

For an X.509 key do not create a CSR but a self-signed certificate. This has not yet been implemented.
GPGME_CREATE_NOSTORE
Since: 1.7.0
Do not store the created key in the local key database. This has not yet been implemented.

GPGME_CREATE_WANTPUB
GPGME_CREATE_WANTSEC
Since: 1.7.0
Return the public or secret key as part of the result structure. This has not yet been implemented.

GPGME_CREATE_FORCE
Since: 1.7.0
The engine does not allow the creation of a key with a user ID already existing in the local key database. This flag can be used to override this check.

GPGME_CREATE_NOEXPIRE
Since: 1.9.0
Request generation of keys that do not expire.

After the operation completed successfully, information about the created key can be retrieved with gpgme_op_genkey_result.

The function returns zero on success, GPG_ERR_NOT_SUPPORTED if the engine does not support the command, or a bunch of other error codes.

gpgme_error_t gpgme_op_createkey_start (gpgme_ctx_t ctx,       [Function]
                      const char *userid, const char *algo, unsigned long reserved,
                      unsigned long expires, gpgme_key_t extrakey, unsigned int flags);
Since: 1.7.0

The function gpgme_op_createkey_start initiates a gpgme_op_createkey operation; see there for details. It must be completed by calling gpgme_wait on the context. See Section 7.8.1 [Waiting For Completion], page 100.

gpgme_error_t gpgme_op_create_subkey (gpgme_ctx_t ctx,       [Function]
                      gpgme_key_t key, const char *algo, unsigned long reserved,
                      unsigned long expires, unsigned int flags);
Since: 1.7.0

The function gpgme_op_create_subkey creates and adds a new subkey to the primary OpenPGP key given by KEY. The only allowed protocol in ctx is GPGME_PROTOCOL_OPENPGP. Subkeys (aka secondary keys) are a concept in the OpenPGP protocol to bind several keys to a primary key. As of now this function requires at least version 2.1.13 of GnuPG.

key specifies the key to operate on.

algo specifies the algorithm for the new subkey. For a list of supported algorithms, see the GnuPG manual. If algo is NULL or the string "default", the subkey is generated using the default algorithm for an encryption subkey of the engine. If the string "future-default" is used the engine may use an encryption algorithm which is planned
to be the default in a future release of the engine; however existing implementation of the protocol may not be able to already handle such future algorithms. "reserved" must be set to zero.

expires specifies the expiration time in seconds. If you supply 0, a reasonable expiration time is chosen. Use the flag GPGME_CREATE_NOEXPRIE to create keys that do not expire. Note that this parameter takes an unsigned long value and not a time_t to avoid problems on systems which use a signed 32 bit time_t. Note further that the OpenPGP protocol uses 32 bit values for timestamps and thus can only encode dates up to the year 2106.

flags takes the same values as described above for gpgme_op_createkey.

After the operation completed successfully, information about the created key can be retrieved with gpgme_op_genkey_result.

The function returns zero on success, GPG_ERR_NOT_SUPPORTED if the engine does not support the command, or a bunch of other error codes.

```
gpgme_error_t gpgme_opcreatesubkey_start (gpgme_ctx_t ctx,     
gpgme_key_t key, const char *algo, unsigned long reserved, 
unsigned long expires, unsigned int flags);
```

Since: 1.7.0

The function gpgme_opcreatesubkey_start initiates a gpgme_opcreatesubkey operation; see there for details. It must be completed by calling gpgme_wait on the context. See Section 7.8.1 [Waiting For Completion], page 100.

```
gpgme_error_t gpgme_opadduid (gpgme_ctx_t ctx,     
gpgme_key_t key, const char *userid, unsigned int flags);
```

Since: 1.7.0

The function gpgme_opadduid adds a new user ID to the OpenPGP key given by KEY. Adding additional user IDs after key creation is a feature of the OpenPGP protocol and thus the protocol for the context ctx must be set to OpenPGP. As of now this function requires at least version 2.1.13 of GnuPG.

key specifies the key to operate on.

 userid is the user ID to add to the key. A user ID is commonly the mail address to be associated with the key. GPGME does not require a specific syntax but if more than a mail address is given, RFC-822 style format is suggested. The value is expected to be in UTF-8 encoding (i.e., no IDN encoding for mail addresses). This is a required parameter.

flags are currently not used and must be set to zero.

The function returns zero on success, GPG_ERR_NOT_SUPPORTED if the engine does not support the command, or a bunch of other error codes.

```
gpgme_error_t gpgme_opadduid_start (gpgme_ctx_t ctx,     
gpgme_key_t key, const char *userid, unsigned int flags);
```

Since: 1.7.0

The function gpgme_opadduid_start initiates a gpgme_opadduid operation; see there for details. It must be completed by calling gpgme_wait on the context. See Section 7.8.1 [Waiting For Completion], page 100.
**gpgme_error_t gpgme_op_revuid** (gpgme_ctx_t ctx, gpgme_key_t key, const char *userid, unsigned int flags);

**Since:** 1.7.0

The function `gpgme_op_revuid` revokes a user ID from the OpenPGP key given by `KEY`. Revoking user IDs after key creation is a feature of the OpenPGP protocol and thus the protocol for the context `ctx` must be set to OpenPGP. As of now this function requires at least version 2.1.13 of GnuPG.

`key` specifies the key to operate on.

`userid` is the user ID to be revoked from the key. The user ID must be given verbatim because the engine does an exact and case sensitive match. Thus the `uid` field from the user ID object (`gpgme_user_id_t`) is to be used. This is a required parameter.

`flags` are currently not used and must be set to zero.

Note that the engine won’t allow to revoke the last valid user ID. To change a user ID is better to first add the new user ID, then revoke the old one, and finally publish the key.

The function returns zero on success, `GPG_ERR_NOT_SUPPORTED` if the engine does not support the command, or a bunch of other error codes.

**gpgme_error_t gpgme_op_revuid_start** (gpgme_ctx_t ctx, gpgme_key_t key, const char *userid, unsigned int flags);

**Since:** 1.7.0

The function `gpgme_op_revuid_start` initiates a `gpgme_op_revuid` operation; see there for details. It must be completed by calling `gpgme_wait` on the context. See Section 7.8.1 [Waiting For Completion], page 100.

**gpgme_error_t gpgme_op_set_ui_flag** (gpgme_ctx_t ctx, gpgme_key_t key, const char *userid, const char *name, const char *value);

**Since:** 1.8.0

The function `gpgme_op_set_ui_flag` is used to set flags on a user ID from the OpenPGP key given by `KEY`. Setting flags on user IDs after key creation is a feature of the OpenPGP protocol and thus the protocol for the context `ctx` must be set to OpenPGP.

`key` specifies the key to operate on. This parameters is required.

`userid` is the user ID of the key to be manipulated. This user ID must be given verbatim because the engine does an exact and case sensitive match. Thus the `uid` field from the user ID object (`gpgme_user_id_t`) is to be used. This is a required parameter.

`name` names the flag which is to be changed. The only currently supported flag is:

**primary**  
This sets the primary key flag on the given user ID. All other primary key flag on other user IDs are removed. `value` must be given as NULL. For technical reasons this function bumps the creation timestamp of all affected self-signatures up by one second. At least GnuPG version 2.1.20 is required.

The function returns zero on success, `GPG_ERR_NOT_SUPPORTED` if the engine does not support the command, or a bunch of other error codes.
gpgme_error_t gpgme_op_set_uid_flag_start (gpgme_ctx_t ctx, 
    gpgme_key_t key, const char *userid, cons char *name, cons char *value);

Since: 1.8.0

The function `gpgme_op_set_uid_flag_start` initiates a `gpgme_op_set_uid_flag` operation; see there for details. It must be completed by calling `gpgme_wait` on the context. See Section 7.8.1 [Waiting For Completion], page 100.

```c

Section 7.8.1 [Waiting For Completion], page 100.

The function `gpgme_op_genkey` generates a new key pair in the context `ctx`. The meaning of `public` and `secret` depends on the crypto backend.

GPG does not support `public` and `secret`, they should be NULL. GnuPG will generate a key pair and add it to the standard key ring. The fingerprint of the generated key is available with `gpgme_op_genkey_result`.

GpgSM requires `public` to be a writable data object. GpgSM will generate a secret key (which will be stored by `gpg-agent`, and return a certificate request in `public`, which then needs to be signed by the certification authority and imported before it can be used. GpgSM does not make the fingerprint available.

The argument `parms` specifies parameters for the key in an string that looks something like XML. The details about the format of `parms` are specific to the crypto engine used by `ctx`. The first line of the parameters must be `<GnupgKeyParams format="internal">` and the last line must be `</GnupgKeyParams>`. Every line in between the first and last lines is treated as a Header: Value pair. In particular, no XML escaping is necessary if you need to include the characters `<`, `>`, or `&`.

Here is an example for GnuPG as the crypto engine (all parameters of OpenPGP key generation are documented in the GPG manual):

```xml
<GnupgKeyParms format="internal">
  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
</GnupgKeyParms>
```

Here is an example for GpgSM as the crypto engine (all parameters of OpenPGP key generation are documented in the GPGSM manual):

```xml
<GnupgKeyParms format="internal">
  Key-Type: RSA
  Key-Length: 1024
  Name-DN: C=de,O=g10 code,OU=Testlab,CN=Joe 2 Tester
  Name-Email: joe@foo.bar
</GnupgKeyParms>
```

Strings should be given in UTF-8 encoding. The only format supported for now is “internal”. The content of the `GnupgKeyParms` container is passed verbatim to the crypto backend. Control statements are not allowed.
After the operation completed successfully, the result can be retrieved with `gpgme_op_genkey_result`.

The function returns the error code `GPG_ERR_NO_ERROR` if the operation could be started successfully, `GPG_ERR_INV_VALUE` if `parms` is not a well-formed string (e.g., does not have the expected tag-like headers and footers), `GPG_ERR_NOT_SUPPORTED` if `public` or `secret` is not valid, and `GPG_ERR_GENERAL` if no key was created by the backend.

```c
[Function]
gpgme_error_t gpgme_op_genkey_start (gpgme_ctx_t ctx, const char *parms, gpgme_data_t public, gpgme_data_t secret)
```

The function `gpgme_op_genkey_start` initiates a `gpgme_op_genkey` operation. It can be completed by calling `gpgme_wait` on the context. See Section 7.8.1 [Waiting For Completion], page 100.

The function returns the error code `GPG_ERR_NO_ERROR` if the operation could be started successfully, `GPG_ERR_INV_VALUE` if `parms` is not a valid XML string, and `GPG_ERR_NOT_SUPPORTED` if `public` or `secret` is not NULL.

```c
[Data type]
gpgme_genkey_result_t
```

This is a pointer to a structure used to store the result of a `gpgme_op_genkey` operation. After successfully generating a key, you can retrieve the pointer to the result with `gpgme_op_genkey_result`. The structure contains the following members:

- `unsigned int primary : 1`
  This flag is set to 1 if a primary key was created and to 0 if not.

- `unsigned int sub : 1`
  This flag is set to 1 if a subkey was created and to 0 if not.

- `unsigned int uid : 1`
  This flag is set to 1 if a user ID was created and to 0 if not.

- `char *fpr`  This is the fingerprint of the key that was created. If both a primary and a subkey were generated, the fingerprint of the primary key will be returned. If the crypto engine does not provide the fingerprint, `fpr` will be a null pointer.

- `gpgme_data_t pubkey`
  **Since:** 1.7.0
  This will eventually be used to return the public key. It is currently not used.

- `gpgme_data_t seckey`
  **Since:** 1.7.0
  This will eventually be used to return the secret key. It is currently not used.

```c
[Function]
gpgme_genkey_result_t gpgme_op_genkey_result (gpgme_ctx_t ctx)
```

The function `gpgme_op_genkey_result` returns a `gpgme_genkey_result_t` pointer to a structure holding the result of a `gpgme_op_genkey` operation. The pointer is
only valid if the last operation on the context was a `gpgme_op_genkey` or `gpgme_op_genkey_start` operation, and if this operation finished successfully. The returned pointer is only valid until the next operation is started on the context.

### 7.5.6 Signing Keys

Key signatures are a unique concept of the OpenPGP protocol. They can be used to certify the validity of a key and are used to create the Web-of-Trust (WoT). Instead of using the `gpgme_op_interact` function along with a finite state machine, GPGME provides a convenient function to create key signatures when using modern GnuPG versions.

```c
void gpgme_error_t gpgme_op_keysign (gpgme_ctx_t ctx,
    gpgme_key_t key, const char *userid, unsigned long expires,
    unsigned int flags);
```

**Function**

Since: 1.7.0

The function `gpgme_op_keysign` adds a new key signature to the public key `KEY`. This function requires at least version 2.1.12 of GnuPG.

`CTX` is the usual context which describes the protocol to use (which must be OpenPGP) and has also the list of signer keys to be used for the signature. The common case is to use the default key for signing other keys. If another key or more than one key shall be used for a key signature, `gpgme_signers_add` can be used. See Section 7.6.4.1 [Selecting Signers], page 86.

`key` specifies the key to operate on.

`userid` selects the user ID or user IDs to be signed. If `userid` is set to `NULL` all valid user IDs are signed. The user ID must be given verbatim because the engine does an exact and case sensitive match. Thus the `uid` field from the user ID object (`gpgme_user_id_t`) is to be used. To select more than one user ID put them all into one string separated by linefeed characters (`\n`) and set the flag `GPGME_KEYSIGN_LFSEP`.

`expires` specifies the expiration time of the new signature in seconds. The common case is to use 0 to not set an expiration date. However, if the configuration of the engine defines a default expiration for key signatures, that is still used unless the flag `GPGME_KEYSIGN_NOEXPIRE` is used. Note that this parameter takes an unsigned long value and not a `time_t` to avoid problems on systems which use a signed 32-bit `time_t`. Note further that the OpenPGP protocol uses 32-bit values for timestamps and thus can only encode dates up to the year 2106.

`flags` can be set to the bit-wise OR of the following flags:

**GPGME_KEYSIGN_LOCAL**

Since: 1.7.0

Instead of creating an exportable key signature, create a key signature which is is marked as non-exportable.

**GPGME_KEYSIGN_LFSEP**

Since: 1.7.0

Although linefeeds are uncommon in user IDs this flag is required to explicitly declare that `userid` may contain several linefeed separated user IDs.
GPGME_KEYSIGN_NOEXPIRE

Force the creation of a key signature without an expiration date. This overrides expire and any local configuration of the engine.

GPGME_KEYSIGN_FORCE

Force the creation of a new signature even if one already exists. This flag has an effect only if the gpg version is at least 2.2.28 but won’t return an error with older versions.

The function returns zero on success, GPG_ERR_NOT_SUPPORTED if the engine does not support the command, or a bunch of other error codes.

```
[Function]
gpgme_error_t gpgme_op_keysign_start (gpgme_ctx_t ctx,  
gpgme_key_t key, const char *userid, unsigned long expires,  
unsigned int flags);
```

Since: 1.7.0

The function `gpgme_op_keysign_start` initiates a `gpgme_op_keysign` operation; see there for details. It must be completed by calling `gpgme_wait` on the context. See Section 7.8.1 [Waiting For Completion], page 100.

```
[Function]
gpgme_error_t gpgme_op_revsig (gpgme_ctx_t ctx,  
gpgme_key_t key, gpgme_key_t signing_key, const char *userid,  
unsigned int flags);
```

Since: 1.14.1

The function `gpgme_op_revsig` revokes key signatures of the public key `key` made with the key `signing_key`. This function requires at least version 2.2.24 of GnuPG.

- `key` specifies the key to operate on.
- `signing_key` specifies the key whose signatures shall be revoked.
- `userid` selects the user ID or user IDs whose signatures shall be revoked. If `userid` is set to `NULL` the signatures on all user IDs are revoked. The user ID must be given verbatim because the engine does an exact and case sensitive match. Thus the `uid` field from the user ID object (`gpgme_user_id_t`) is to be used. To select more than one user ID put them all into one string separated by linefeeds characters (`\n`) and set the flag `GPGME_REVSIG_LFSEP`.

- `flags` can be set to the bit-wise OR of the following flags:
  - **GPGME_REVSIG_LFSEP**

Although linefeeds are uncommon in user IDs this flag is required to explicitly declare that `userid` may contain several linefeed separated user IDs.

The function returns zero on success, GPG_ERR_NOT_SUPPORTED if the engine does not support the command, or a bunch of other error codes.

```
[Function]
gpgme_error_t gpgme_op_revsig_start (gpgme_ctx_t ctx,  
gpgme_key_t key, gpgme_key_t signing_key, const char *userid,  
unsigned int flags);
```

Since: 1.14.1
The function `gpgme_op_revsig_start` initiates a `gpgme_op_revsig` operation; see there for details. It must be completed by calling `gpgme_wait` on the context. See Section 7.8.1 [Waiting For Completion], page 100.

### 7.5.7 Exporting Keys

Exporting keys means the same as running `gpg` with the command `--export`. However, a mode flag can be used to change the way the export works. The available mode flags are described below, they may be or-ed together.

**GPGME_EXPORT_MODE_EXTERN**

If this bit is set, the output is send directly to the default keyserver. This is currently only allowed for OpenPGP keys. It is good practise to not send more than a few dozens key to a keyserver at one time. Using this flag requires that the `keydata` argument of the export function is set to `NULL`.

**GPGME_EXPORT_MODE_MINIMAL**

Since: 1.3.1

If this bit is set, the smallest possible key is exported. For OpenPGP keys it removes all signatures except for the latest self-signatures. For X.509 keys it has no effect.

**GPGME_EXPORT_MODE_SSH**

Since: 1.4.0

If this bit is set, the latest authentication key of the requested OpenPGP key is exported in the OpenSSH public key format. This accepts just a single key; to force the export of a specific subkey a fingerprint pattern with an appended exclamation mark may be used.

**GPGME_EXPORT_MODE_SECRET**

Since: 1.6.0

Instead of exporting the public key, the secret key is exported. This may not be combined with `GPGME_EXPORT_MODE_EXTERN`. For X.509 the export format is PKCS#8.

**GPGME_EXPORT_MODE_SECRET_SUBKEY**

Since: 1.17.0

If this bit is set, then a secret subkey is exported. The subkey to export must be specified with fingerprint pattern with an appended exclamation mark. This is currently only allowed for OpenPGP keys. This flag may not be combined with `GPGME_EXPORT_MODE_EXTERN`. This flag is not supported by the export functions that take an array of keys.

**GPGME_EXPORT_MODE_RAW**

Since: 1.6.0

If this flag is used with `GPGME_EXPORT_MODE_SECRET` for an X.509 key the export format will be changed to PKCS#1. This flag may not be used with OpenPGP.

**GPGME_EXPORT_MODE_PKCS12**

Since: 1.6.0
If this flag is used with `GPGME_EXPORT_MODE_SECRET` for an X.509 key the export format will be changed to PKCS#12 which also includes the certificate. This flag may not be used with OpenPGP.

```
gpgme_error_t gpgme_op_export (gpgme_ctx_t ctx, const char *pattern, gpgme_export_mode_t mode, gpgme_data_t keydata)
```
The function `gpgme_op_export` extracts public keys and returns them in the data buffer `keydata`. The output format of the key data returned is determined by the ASCII armor attribute set for the context `ctx`, or, if that is not set, by the encoding specified for `keydata`.

If `pattern` is NULL, all available keys are returned. Otherwise, `pattern` contains an engine specific expression that is used to limit the list to all keys matching the pattern. `mode` is usually 0; other values are described above.

The function returns the error code `GPG_ERR_NO_ERROR` if the operation completed successfully, `GPG_ERR_INV_VALUE` if `keydata` is not a valid empty data buffer, and passes through any errors that are reported by the crypto engine support routines.

```
gpgme_error_t gpgme_op_export_start (gpgme_ctx_t ctx, const char *pattern, gpgme_export_mode_t mode, gpgme_data_t keydata)
```
The function `gpgme_op_export_start` initiates a `gpgme_op_export` operation. It can be completed by calling `gpgme_wait` on the context. See Section 7.8.1 [Waiting For Completion], page 100.

The function returns the error code `GPG_ERR_NO_ERROR` if the operation could be started successfully, and `GPG_ERR_INV_VALUE` if `keydata` is not a valid empty data buffer.

```
gpgme_error_t gpgme_op_export_ext (gpgme_ctx_t ctx, const char *pattern[], gpgme_export_mode_t mode, gpgme_data_t keydata)
```
The function `gpgme_op_export_ext` extracts public keys and returns them in the data buffer `keydata`. The output format of the key data returned is determined by the ASCII armor attribute set for the context `ctx`, or, if that is not set, by the encoding specified for `keydata`.

If `pattern` or `*pattern` is NULL, all available keys are returned. Otherwise, `pattern` is a NULL terminated array of strings that are used to limit the list to all keys matching at least one of the patterns verbatim.

`mode` is usually 0; other values are described above.

The function returns the error code `GPG_ERR_NO_ERROR` if the operation completed successfully, `GPG_ERR_INV_VALUE` if `keydata` is not a valid empty data buffer, and passes through any errors that are reported by the crypto engine support routines.

```
gpgme_error_t gpgme_op_export_ext_start (gpgme_ctx_t ctx, const char *pattern[], gpgme_export_mode_t mode, gpgme_data_t keydata)
```
The function `gpgme_op_export_ext_start` initiates a `gpgme_op_export_ext` operation. It can be completed by calling `gpgme_wait` on the context. See Section 7.8.1 [Waiting For Completion], page 100.
The function returns the error code `GPG_ERR_NO_ERROR` if the operation could be started successfully, and `GPG_ERR_INV_VALUE` if `keydata` is not a valid empty data buffer.

```
gpgme_error_t gpgme_op_export_keys (gpgme_ctx_t ctx, [Function] 
gpgme_key_t keys[], gpgme_export_mode_t mode, gpgme_data_t keydata) 
```

**Since:** 1.2.0

The function `gpgme_op_export_keys` extracts public keys and returns them in the data buffer `keydata`. The output format of the key data returned is determined by the ASCII armor attribute set for the context `ctx`, or, if that is not set, by the encoding specified for `keydata`.

The keys to export are taken from the NULL terminated array `keys`. Only keys of the currently selected protocol of `ctx` which do have a fingerprint set are considered for export. Other keys specified by the `keys` are ignored. In particular OpenPGP keys retrieved via an external key listing are not included.

`mode` is usually 0; other values are described above.

The function returns the error code `GPG_ERR_NO_ERROR` if the operation completed successfully, `GPG_ERR_INV_VALUE` if `keydata` is not a valid empty data buffer, `GPG_ERR_NO_DATA` if no useful keys are in `keys` and passes through any errors that are reported by the crypto engine support routines.

```
gpgme_error_t gpgme_op_export_keys_start (gpgme_ctx_t ctx, [Function] 
gpgme_key_t keys[], gpgme_export_mode_t mode, gpgme_data_t keydata) 
```

**Since:** 1.2.0

The function `gpgme_op_export_keys_start` initiates a `gpgme_op_export_ext` operation. It can be completed by calling `gpgme_wait` on the context. See Section 7.8.1 [Waiting For Completion], page 100.

The function returns the error code `GPG_ERR_NO_ERROR` if the operation could be started successfully, and `GPG_ERR_INV_VALUE` if `keydata` is not a valid empty data buffer, `GPG_ERR_NO_DATA` if no useful keys are in `keys` and passes through any errors that are reported by the crypto engine support routines.

### 7.5.8 Importing Keys

Importing keys means the same as running `gpg` with the command ‘--import’.

```
gpgme_error_t gpgme_op_import (gpgme_ctx_t ctx, [Function] 
gpgme_data_t keydata) 
```

The function `gpgme_op_import` adds the keys in the data buffer `keydata` to the key ring of the crypto engine used by `ctx`. The format of `keydata` can be ASCII armored, for example, but the details are specific to the crypto engine.

After the operation completed successfully, the result can be retrieved with `gpgme_op_import_result`.

The function returns the error code `GPG_ERR_NO_ERROR` if the import was completed successfully, `GPG_ERR_INV_VALUE` if `ctx` or `keydata` is not a valid pointer, and `GPG_ERR_NO_DATA` if `keydata` is an empty data buffer.
gpgme_error_t gpgme_op_import_start (gpgme_ctx_t ctx,
     gpgme_data_t keydata)  

The function gpgme_op_import_start initiates a gpgme_op_import operation. It can be completed by calling gpgme_wait on the context. See Section 7.8.1 [Waiting For Completion], page 100.

The function returns the error code GPG_ERR_NO_ERROR if the import could be started successfully, GPG_ERR_INV_VALUE if ctx or keydata is not a valid pointer, and GPG_ERR_NO_DATA if keydata is an empty data buffer.

gpgme_error_t gpgme_op_import_keys (gpgme_ctx_t ctx,
     gpgme_key_t *keys)

Since: 1.2.0

The function gpgme_op_import_keys adds the keys described by the NULL terminated array keys to the key ring of the crypto engine used by ctx. It is used to actually import and make keys permanent which have been retrieved from an external source (i.e., using GPGME_KEYLIST_MODE_EXTERN) earlier. The external keylisting must have been made with the same context configuration (in particular the same home directory). Note that for OpenPGP this may require another access to the keyserver over the network.

Only keys of the currently selected protocol of ctx are considered for import. Other keys specified by the keys are ignored. As of now all considered keys must have been retrieved using the same method, i.e., the used key listing mode must be identical.

After the operation completed successfully, the result can be retrieved with gpgme_op_import_result.

To move keys from one home directory to another, export and import the keydata using gpgme_op_export and gpgme_op_import.

The function returns the error code GPG_ERR_NO_ERROR if the import was completed successfully, GPG_ERR_INV_VALUE if ctx is not a valid pointer, GPG_ERR_CONFLICT if the key listing mode does not match, and GPG_ERR_NO_DATA if no keys were considered for import.

gpgme_error_t gpgme_op_import_keys_start (gpgme_ctx_t ctx,
     gpgme_key_t *keys)

Since: 1.2.0

The function gpgme_op_import_keys_start initiates a gpgme_op_import_keys operation. It can be completed by calling gpgme_wait on the context. See Section 7.8.1 [Waiting For Completion], page 100.

The function returns the error code GPG_ERR_NO_ERROR if the import was started successfully, GPG_ERR_INV_VALUE if ctx is not a valid pointer, GPG_ERR_CONFLICT if the key listing mode does not match, and GPG_ERR_NO_DATA if no keys were considered for import.

---

Thus it is a replacement for the usual workaround of exporting and then importing a key to make an X.509 key permanent.
gpgme_error_t gpgme_op_receive_keys (gpgme_ctx_t ctx, const char *keyids[])

Since: 1.17.0

The function `gpgme_op_receive_keys` adds the keys described by the NULL terminated array `keyids` to the key ring of the crypto engine used by `ctx`. It is used to retrieve and import keys from an external source. This function currently works only for OpenPGP.

After the operation completed successfully, the result can be retrieved with `gpgme_op_import_result`.

The function returns the error code `GPG_ERR_NO_ERROR` if the import was completed successfully, `GPG_ERR_INV_VALUE` if `ctx` is not a valid pointer, and `GPG_ERR_NO_DATA` if no keys were considered for import.

```
gpgme_error_t gpgme_op_receive_keys_start (gpgme_ctx_t ctx, const char *keyids[])

Since: 1.17.0

The function `gpgme_op_receive_keys_start` initiates a `gpgme_op_receive_keys` operation. It can be completed by calling `gpgme_wait` on the context. See Section 7.8.1 [Waiting For Completion], page 100.

The function returns the error code `GPG_ERR_NO_ERROR` if the import was started successfully, `GPG_ERR_INV_VALUE` if `ctx` is not a valid pointer, and `GPG_ERR_NO_DATA` if no keys were considered for import.
```

```
gpgme_import_status_t

This is a pointer to a structure used to store a part of the result of a `gpgme_op_import` operation. For each considered key one status is added that contains information about the result of the import. The structure contains the following members:

- gpgme_import_status_t next
  - This is a pointer to the next status structure in the linked list, or `NULL` if this is the last element.

- char *fpr
  - This is the fingerprint of the key that was considered, or `NULL` if the fingerprint of the key is not known, e.g., because the key to import was encrypted and decryption failed.

- gpgme_error_t result
  - If the import was not successful, this is the error value that caused the import to fail. Otherwise the error code is `GPG_ERR_NO_ERROR`.

- unsigned int status
  - This is a bit-wise OR of the following flags that give more information about what part of the key was imported. If the key was already known, this might be 0.
    - `GPGME_IMPORT_NEW`
      - The key was new.
    - `GPGME_IMPORT_UID`
      - The key contained new user IDs.
```
**GPGME_IMPORT_SIG**
The key contained new signatures.

**GPGME_IMPORT_SUBKEY**
The key contained new sub keys.

**GPGME_IMPORT_SECRET**
The key contained a secret key.

gpgme_import_result_t [Data type]
This is a pointer to a structure used to store the result of a gpgme_op_import operation. After a successful import operation, you can retrieve the pointer to the result with gpgme_op_import_result. The structure contains the following members:

- **int considered**
  The total number of considered keys.

- **int no_user_id**
  The number of keys without user ID.

- **int imported**
  The total number of imported keys.

- **int imported_rsa**
  The number of imported RSA keys.

- **int unchanged**
  The number of unchanged keys.

- **int new_user_ids**
  The number of new user IDs.

- **int new_sub_keys**
  The number of new sub keys.

- **int new_signatures**
  The number of new signatures.

- **int new_revocations**
  The number of new revocations.

- **int secret_read**
  The total number of secret keys read.

- **int secret_imported**
  The number of imported secret keys.

- **int secret_unchanged**
  The number of unchanged secret keys.

- **int not_imported**
  The number of keys not imported.

gpgme_import_status_t imports
A list of gpgme_import_status_t objects which contain more information about the keys for which an import was attempted.
int skipped_v3_keys

For security reasons modern versions of GnuPG do not anymore support v3 keys (created with PGP 2.x) and ignores them on import. This counter provides the number of such skipped v3 keys.

\[\text{Function}\]
\[
gpgme\_import\_result\_t\ gpgme\_op\_import\_result\ (gpgme\_ctx\_t\ ctx)\]

The function \texttt{gpgme\_op\_import\_result} returns a \texttt{gpgme\_import\_result\_t} pointer to a structure holding the result of a \texttt{gpgme\_op\_import} operation. The pointer is only valid if the last operation on the context was a \texttt{gpgme\_op\_import} or \texttt{gpgme\_op\_import\_start} operation, and if this operation finished successfully. The returned pointer is only valid until the next operation is started on the context.

\section*{7.5.9 Deleting Keys}

\[\text{Function}\]
\[
gpgme\_error\_t\ gpgme\_op\_delete\_ext\ (gpgme\_ctx\_t\ ctx,\ const\ gpgme\_key\_t\ key,\ unsigned\ int\ flags)\]
\textbf{Since: 1.9.1}

The function \texttt{gpgme\_op\_delete\_ext} deletes the key \texttt{key} from the key ring of the crypto engine used by \texttt{ctx}.

\textit{flags} can be set to the bit-wise OR of the following flags:

- **GPGME\_DELETE\_ALLOW\_SECRET**
  \textbf{Since: 1.9.1}
  If not set, only public keys are deleted. If set, secret keys are deleted as well, if that is supported.

- **GPGME\_DELETE\_FORCE**
  \textbf{Since: 1.9.1}
  If set, the user is not asked to confirm the deletion.

The function returns the error code \texttt{GPG\_ERR\_NO\_ERROR} if the key was deleted successfully, \texttt{GPG\_ERR\_INV\_VALUE} if \texttt{ctx} or \texttt{key} is not a valid pointer, \texttt{GPG\_ERR\_NO\_PUBKEY} if \texttt{key} could not be found in the keyring, \texttt{GPG\_ERR\_AMBIGUOUS\_NAME} if the key was not specified unambiguously, and \texttt{GPG\_ERR\_CONFLICT} if the secret key for \texttt{key} is available, but \texttt{allow\_secret} is zero.

\[\text{Function}\]
\[
gpgme\_error\_t\ gpgme\_op\_delete\_ext\_start\ (gpgme\_ctx\_t\ ctx,\ const\ gpgme\_key\_t\ key,\ unsigned\ int\ flags)\]
\textbf{Since: 1.9.1}

The function \texttt{gpgme\_op\_delete\_ext\_start} initiates a \texttt{gpgme\_op\_delete} operation. It can be completed by calling \texttt{gpgme\_wait} on the context. See \textsection{7.8.1 \textit{Waiting For Completion}}, page 100.

The function returns the error code \texttt{GPG\_ERR\_NO\_ERROR} if the operation was started successfully, and \texttt{GPG\_ERR\_INV\_VALUE} if \texttt{ctx} or \texttt{key} is not a valid pointer.

The following functions allow only to use one particular flag. Their use is thus deprecated.
gpgme_error_t gpgme_op_delete (gpgme_ctx_t ctx, const gpgme_key_t key, int allow_secret)  
[Function]  
Similar to gpgme_op_delete_ext, but only the flag GPGME_DELETE_ALLOW_SECRET can be provided. Actually all true values are mapped to this flag.

gpgme_error_t gpgme_op_delete_start (gpgme_ctx_t ctx, const gpgme_key_t key, int allow_secret)  
[Function]  
Similar to gpgme_op_delete_ext_start, but only the flag GPGME_DELETE_ALLOW_SECRET can be provided. Actually all true values are mapped to this flag.

7.5.10 Changing Passphrases

gpgme_error_t gpgme_op_passwd (gpgme_ctx_t ctx, const gpgme_key_t key, unsigned int flags)  
SINCE: 1.3.0  
The function gpgme_op_passwd changes the passphrase of the private key associated with key. The only allowed value for flags is 0. The backend engine will usually popup a window to ask for the old and the new passphrase. Thus this function is not useful in a server application (where passphrases are not required anyway).
Note that old gpg engines (before version 2.0.15) do not support this command and will silently ignore it.

gpgme_error_t gpgme_op_passwd_start (gpgme_ctx_t ctx, const gpgme_key_t key, unsigned int flags)  
SINCE: 1.3.0  
The function gpgme_op_passwd_start initiates a gpgme_op_passwd operation. It can be completed by calling gpgme_wait on the context. See Section 7.8.1 [Waiting For Completion], page 100.
The function returns 0 if the operation was started successfully, and an error code if one of the arguments is not valid or the operation could not be started.

7.5.11 Changing TOFU Data

The OpenPGP engine features a Trust-On-First-Use (TOFU) key validation model. For resolving conflicts it is necessary to declare the policy for a key. See the GnuPG manual for details on the TOFU implementation.

enum gpgme_tofu_policy_t  
SINCE: 1.7.0  
The gpgme_tofu_policy_t type specifies the set of possible policy values that are supported by GPGME:
GPGME_TOFU_POLICY_AUTO  
Set the policy to “auto”.
GPGME_TOFU_POLICY_GOOD  
Set the policy to “good”.
GPGME_TOFU_POLICY_BAD  
Set the policy to “bad”.

GPGME_TOFU_POLICY_ASK
Set the policy to “ask”.

GPGME_TOFU_POLICY_UNKNOWN
Set the policy to “unknown”.

To change the policy for a key the following functions can be used:

```c
#include <gpgme.h>

int set_policy_to_ask(gpgme_ctx_t ctx, const gpgme_key_t key, gpgme_tofu_policy_t policy); 
```

**GPGME_TOFU_POLICY_UNKNOWN**

Set the policy to “unknown”.

To change the policy for a key the following functions can be used:

```c
#include <gpgme.h>

int set_policy_to_unknown(gpgme_ctx_t ctx, const gpgme_key_t key, gpgme_tofu_policy_t policy); 
```

Since: 1.7.0

The function `gpgme_op_tofu_policy` changes the TOFU policy of key. The valid values for `policy` are listed above. As of now this function does only work for OpenPGP and requires at least version 2.1.10 of GnuPG.

The function returns zero on success, **GPG_ERR_NOT_SUPPORTED** if the engine does not support the command, or a bunch of other error codes.

```c
#include <gpgme.h>

int start_set_policy_to_ask(gpgme_ctx_t ctx, const gpgme_key_t key, gpgme_tofu_policy_t policy); 
```

Since: 1.7.0

The function `gpgme_op_tofu_policy_start` initiates a `gpgme_op_tofu_policy` operation. It can be completed by calling `gpgme_wait` on the context. See Section 7.8.1 [Waiting For Completion], page 100.

The function returns 0 if the operation was started successfully, and an error code if one of the arguments is not valid or the operation could not be started.

### 7.5.12 Advanced Key Editing

```c
#include <gpgme.h>

int key_interact_operations(gpgme_ctx_t ctx, gpgme_key_t key, unsigned int flags, gpgme_interact_cb_t fnc, void *handle, gpgme_data_t out); 
```

Since: 1.7.0

The function `gpgme_op_interact` processes the key `KEY` interactively, using the interact callback function `FNC` with the handle `HANDLE`. The callback is invoked for every status and command request from the crypto engine. The output of the crypto engine is written to the data object `out`. 
Note that the protocol between the callback function and the crypto engine is specific to the crypto engine and no further support in implementing this protocol correctly is provided by GPGME.

flags modifies the behaviour of the function; the only defined bit value is:

\texttt{GPGME\_INTERACT\_CARD}

\texttt{Since: 1.7.0}

This is used for smartcard based keys and uses gpg's \texttt{--card-edit} command.

The function returns 0 if the edit operation completes successfully, \texttt{GPG\_ERR\_INV\_VALUE} if \texttt{ctx} or \texttt{key} is not a valid pointer, and any error returned by the crypto engine or the edit callback handler.

\begin{verbatim}
[gpgme_error_t] gpgme_op_interact_start (gpgme_ctx_t ctx, gpgme_key_t key, unsigned int flags, gpgme_interact_cb_t fnc, void *handle, gpgme_data_t out)
[Since: 1.7.0]
\end{verbatim}

The function \texttt{gpgme_op_interact_start} initiates a \texttt{gpgme_op_interact} operation. It can be completed by calling \texttt{gpgme_wait} on the context. See \texttt{Section 7.8.1 [Waiting For Completion]}, page 100.

The function returns 0 if the operation was started successfully, and \texttt{GPG\_ERR\_INV\_VALUE} if \texttt{ctx} or \texttt{key} is not a valid pointer.

### 7.6 Crypto Operations

Sometimes, the result of a crypto operation returns a list of invalid keys encountered in processing the request. The following structure is used to hold information about such a key.

\begin{verbatim}
[gpgme_invalid_key_t] This is a pointer to a structure used to store a part of the result of a crypto operation which takes user IDs as one input parameter. The structure contains the following members:

\texttt{gpgme_invalid_key_t next}

This is a pointer to the next invalid key structure in the linked list, or \texttt{NULL} if this is the last element.

\texttt{char *fpr}

The fingerprint or key ID of the invalid key encountered.

\texttt{gpgme_error_t reason}

An error code describing the reason why the key was found invalid.
\end{verbatim}

### 7.6.1 Decrypt

\begin{verbatim}
[gpgme_error_t] gpgme_op_decrypt (gpgme_ctx_t ctx, gpgme_data_t cipher, gpgme_data_t plain)
\end{verbatim}

The function \texttt{gpgme_op_decrypt} decrypts the ciphertext in the data object \texttt{cipher} or, if a file name is set on the data object, the ciphertext stored in the corresponding
The decrypted ciphertext is stored into the data object `plain` or written to the file set with `gpgme_data_set_file_name` for the data object `plain`.

The function returns the error code `GPG_ERR_NO_ERROR` if the ciphertext could be decrypted successfully, `GPG_ERR_INV_VALUE` if `ctx`, `cipher` or `plain` is not a valid pointer, `GPG_ERR_NO_DATA` if `cipher` does not contain any data to decrypt, `GPG_ERR_DECRYPT_FAILED` if `cipher` is not a valid cipher text, `GPG_ERR_BAD_PASSPHRASE` if the passphrase for the secret key could not be retrieved, and passes through some errors that are reported by the crypto engine support routines.

```c
int gpgme_error_t gpgme_op_decrypt_start (gpgme_ctx_t ctx,
                               gpgme_data_t cipher, gpgme_data_t plain)
```

The function `gpgme_op_decrypt_start` initiates a `gpgme_op_decrypt` operation. It can be completed by calling `gpgme_wait` on the context. See Section 7.8.1 [Waiting For Completion], page 100.

The function returns the error code `GPG_ERR_NO_ERROR` if the operation could be started successfully, and `GPG_ERR_INV_VALUE` if `cipher` or `plain` is not a valid pointer.

```c
int gpgme_error_t gpgme_op_decrypt_ext ( gpgme_ctx_t ctx,
                               gpgme_decrypt_flags_t flags, gpgme_data_t cipher, gpgme_data_t plain)
```

Since: 1.8.0

The function `gpgme_op_decrypt_ext` is the same as `gpgme_op_decrypt` but has an additional argument `flags`. If `flags` is 0 both function behave identically.

If the flag `GPGME_DECRYPT_ARCHIVE` is set, then an encrypted archive in the data object `cipher` is decrypted and extracted. The content of the archive is extracted into a directory named `GPGARCH_n_` (where `n` is a number) or into the directory set with `gpgme_data_set_file_name` for the data object `plain`.

The value in `flags` is a bitwise-or combination of one or multiple of the following bit values:

- **GPGME_DECRYPT_VERIFY**
  Since: 1.8.0
  The `GPGME_DECRYPT_VERIFY` symbol specifies that this function shall exactly act as `gpgme_op_decrypt_verify`.

- **GPGME_DECRYPT_ARCHIVE**
  Since: 1.19.0
  The `GPGME_DECRYPT_ARCHIVE` symbol specifies that the input is an encrypted archive that shall be decrypted and extracted. This feature is currently only supported for the OpenPGP crypto engine and requires GnuPG 2.4.1.

- **GPGME_DECRYPT_UNWRAP**
  Since: 1.8.0
  The `GPGME_DECRYPT_UNWRAP` symbol specifies that the output shall be an OpenPGP message with only the encryption layer removed. This requires GnuPG 2.1.12 and works only for OpenPGP. This is the counterpart to `GPGME_ENCRYPT_WRAP`.

The function returns the error codes as described for `gpgme_op_decrypt`. 
The function `gpgme_op_decrypt_ext_start` initiates a `gpgme_op_decrypt_ext` operation. It can be completed by calling `gpgme_wait` on the context. See Section 7.8.1 [Waiting For Completion], page 100.

The function returns the error code `GPG_ERR_NO_ERROR` if the operation could be started successfully, and `GPG_ERR_INV_VALUE` if `cipher` or `plain` is not a valid pointer.

This is a pointer to a structure used to store information about the recipient of an encrypted text which is decrypted in a `gpgme_op_decrypt` operation. This information (except for the status field) is even available before the operation finished successfully, for example in a passphrase callback. The structure contains the following members:

- `next`: This is a pointer to the next recipient structure in the linked list, or `NULL` if this is the last element.
- `pubkey_algo_t`: The public key algorithm used in the encryption.
- `char *keyid`: This is the key ID of the key (in hexadecimal digits) used as recipient.
- `error_t status`: This is an error number with the error code `GPG_ERR_NO_SECKEY` if the secret key for this recipient is not available, and 0 otherwise.

This is a pointer to a structure used to store the result of a `gpgme_op_decrypt` operation. After successfully decrypting data, you can retrieve the pointer to the result with `gpgme_op_decrypt_result`. As with all result structures, it this structure shall be considered read-only and an application must not allocate such a structure on its own. The structure contains the following members:

- `unsupported_algorithm`: If an unsupported algorithm was encountered, this string describes the algorithm that is not supported.
- `wrong_key_usage`: This is true if the key was not used according to its policy.
- `legacy_cipher_nomdc`: Since: 1.11.2 The message was made by a legacy algorithm without any integrity protection. This might be an old but legitimate message.
- `is_mime`: Since: 1.11.0 The message claims that the content is a MIME object.
unsigned int is_de_vs : 1;
Since: 1.10.0 The message was encrypted in a VS-NfD compliant way.
This is a specification in Germany for a restricted communication level.

gpgme_recipient_t recipients
Since: 1.1.0
This is a linked list of recipients to which this message was encrypted.

char *file_name
This is the filename of the original plaintext message file if it is known,
otherwise this is a null pointer.

char *session_key
Since: 1.8.0
A textual representation (null-terminated string) of the session key used in
symmetric encryption of the message, if the context has been set to ex-
port session keys (see gpgme_set_ctx_flag, "export-session-key"),
and a session key was available for the most recent decryption operation.
Otherwise, this is a null pointer.
You must not try to access this member of the struct unless
gpgme_set_ctx_flag (ctx, "export-session-key") returns success
or gpgme_get_ctx_flag (ctx, "export-session-key") returns true
(non-empty string).

char *symkey_algo
Since: 1.11.0
A string with the symmetric encryption algorithm and mode using the
format "<algo>.<mode>". Note that the deprecated non-MDC encryption
mode of OpenPGP is given as "PGPCFB".

[gFunction]

```
gpgme_decrypt_result_t gpgme_op_decrypt_result (gpgme_ctx_t ctx)
```

The function gpgme_op_decrypt_result returns a gpgme_decrypt_result_t
pointer to a structure holding the result of a gpgme_op_decrypt operation. The
pointer is only valid if the last operation on the context was a gpgme_op_decrypt or
gpgme_op_decrypt_start operation. If the operation failed this might be a NULL
pointer. The returned pointer is only valid until the next operation is started on the
context.

7.6.2 Verify

```
gpgme_error_t gpgme_op_verify (gpgme_ctx_t ctx,
gpgme_data_t sig, gpgme_data_t signed_text, gpgme_data_t plain)
```

The function gpgme_op_verify verifies that the signature in the data object sig is
a valid signature. If sig is a detached signature, then the signed text should be
provided in signed_text and plain should be a null pointer. Otherwise, if sig is a
normal (or cleartext) signature, signed_text should be a null pointer and plain should
be a writable data object that will contain the plaintext after successful verification.
If a file name is set on the data object sig (or on the data object signed_text), then
the data of the signature (resp. the data of the signed text) is not read from the data object but from the file with the given file name. If a file name is set on the data object plain then the plaintext is not stored in the data object but it is written to a file with the given file name.

The results of the individual signature verifications can be retrieved with gpgme_op_verify_result.

The function returns the error code GPG_ERR_NO_ERROR if the operation could be completed successfully, GPG_ERR_INV_VALUE if ctx, sig or plain is not a valid pointer, GPG_ERR_NO_DATA if sig does not contain any data to verify, and passes through any errors that are reported by the crypto engine support routines.

GPGME_VERIFY_ARCHIVE

Since: 1.19.0

The GPGME_VERIFY_ARCHIVE symbol specifies that the input is a signed archive that shall be verified and extracted. This feature is currently only supported for the OpenPGP crypto engine and requires GnuPG 2.4.1.

The function returns the error codes as described for gpgme_op_decrypt respective gpgme_op_encrypt.

The function gpgme_op_verify_start initiates a gpgme_op_verify operation. It can be completed by calling gpgme_wait on the context. See Section 7.8.1 [Waiting For Completion], page 100.

The function returns the error code GPG_ERR_NO_ERROR if the operation could be started successfully, GPG_ERR_INV_VALUE if ctx, sig or plain is not a valid pointer, and GPG_ERR_NO_DATA if sig or plain does not contain any data to verify.

The function gpgme_op_verify_ext is the same as gpgme_op_verify but has an additional argument flags. If flags is 0 both function behave identically.

If the flag GPGME_VERIFY_ARCHIVE is set, then a signed archive in the data object sig is verified and extracted. The content of the archive is extracted into a directory named GPGARCH_n_ (where n is a number) or into the directory set with gpgme_data_set_file_name for the data object plain.

The value in flags is a bitwise-or combination of one or multiple of the following bit values:

GPGME_VERIFY_ARCHIVE

The function gpgme_op_verify_ext_start initiates a gpgme_op_verify_ext operation. It can be completed by calling gpgme_wait on the context. See Section 7.8.1 [Waiting For Completion], page 100.
The function returns the error code \texttt{GPG\_ERR\_NO\_ERROR} if the operation could be started successfully, \texttt{GPG\_ERR\_INV\_VALUE} if \texttt{ctx}, \texttt{sig} or \texttt{plain} is not a valid pointer, and \texttt{GPG\_ERR\_NO\_DATA} if \texttt{sig} or \texttt{plain} does not contain any data to verify.

\texttt{gpgme\_sig\_notation\_t} \hspace{1cm} [Data type]
This is a pointer to a structure used to store a part of the result of a \texttt{gpgme\_op\_verify} operation. The structure contains the following members:

\texttt{gpgme\_sig\_notation\_t next}
This is a pointer to the next new signature notation structure in the linked list, or \texttt{NULL} if this is the last element.

\texttt{char *name}
The name of the notation field. If this is \texttt{NULL}, then the member \texttt{value} will contain a policy URL.

\texttt{int name\_len}
The length of the \texttt{name} field. For strings the length is counted without the trailing binary zero.

\texttt{char *value}
The value of the notation field. If \texttt{name} is \texttt{NULL}, then this is a policy URL.

\texttt{int value\_len}
The length of the \texttt{value} field. For strings the length is counted without the trailing binary zero.

\texttt{gpgme\_sig\_notation\_flags\_t flags}
The accumulated flags field. This field contains the flags associated with the notation data in an accumulated form which can be used as an argument to the function \texttt{gpgme\_sig\_notation\_add}. The value \texttt{flags} is a bitwise-or combination of one or multiple of the following bit values:

\texttt{GPGME\_SIG\_NOTATION\_HUMAN\_READABLE}
\texttt{Since: 1.1.0}
The \texttt{GPGME\_SIG\_NOTATION\_HUMAN\_READABLE} symbol specifies that the notation data is in human readable form.

\texttt{GPGME\_SIG\_NOTATION\_CRITICAL}
\texttt{Since: 1.1.0}
The \texttt{GPGME\_SIG\_NOTATION\_CRITICAL} symbol specifies that the notation data is critical.

\texttt{unsigned int human\_readable : 1}
This is true if the \texttt{GPGME\_SIG\_NOTATION\_HUMAN\_READABLE} flag is set and false otherwise. This flag is only valid for notation data, not for policy URLs.

\texttt{unsigned int critical : 1}
This is true if the \texttt{GPGME\_SIG\_NOTATION\_CRITICAL} flag is set and false otherwise. This flag is valid for notation data and policy URLs.
gpgme_signature_t[Data type]
This is a pointer to a structure used to store a part of the result of a gpgme_op_verify operation. The structure contains the following members:

gpgme_signature_t next
This is a pointer to the next new signature structure in the linked list, or NULL if this is the last element.

summary
This is a bit vector giving a summary of the signature status. It provides an easy interface to a defined semantic of the signature status. Checking just one bit is sufficient to see whether a signature is valid without any restrictions. This means that you can check for GPGME_SIGSUM_VALID like this:

```c
if ((sig.summary & GPGME_SIGSUM_VALID))
{
    ..do stuff if valid..
}
else
{
    ..do stuff if not fully valid..
}
```

The defined bits are:

- **GPGME_SIGSUM_VALID**
  The signature is fully valid.

- **GPGME_SIGSUM_GREEN**
  The signature is good but one might want to display some extra information. Check the other bits.

- **GPGME_SIGSUM_RED**
  The signature is bad. It might be useful to check other bits and display more information, i.e., a revoked certificate might not render a signature invalid when the message was received prior to the cause for the revocation.

- **GPGME_SIGSUM_KEY_REVOKED**
  The key or at least one certificate has been revoked.

- **GPGME_SIGSUM_KEY_EXPIRED**
  The key or one of the certificates has expired. It is probably a good idea to display the date of the expiration.

- **GPGME_SIGSUM_SIG_EXPIRED**
  The signature has expired.

- **GPGME_SIGSUM_KEY_MISSING**
  Can’t verify due to a missing key or certificate.

- **GPGME_SIGSUM_CRL_MISSING**
  The CRL (or an equivalent mechanism) is not available.
GPGME_SIGSUM_CRL_TOO_OLD
Available CRL is too old.

GPGME_SIGSUM_BAD_POLICY
A policy requirement was not met.

GPGME_SIGSUM_SYS_ERROR
A system error occurred.

GPGME_SIGSUM_TOFU_CONFLICT
A TOFU conflict was detected.

cchar *fpr
This is the fingerprint or key ID of the signature.

gpgme_error_t status
This is the status of the signature. In particular, the following status
codes are of interest:

GPG_ERR_NO_ERROR
This status indicates that the signature could be verified or
that there is no signature. For the combined result this status
means that all signatures could be verified.
Note: This does not mean that a valid signature could be
found. Check the summary field for that.
For example a gpgme_op_decrypt_verify returns a verifica-
tion result with GPG_ERR_NO_ERROR for encrypted but
unsigned data.

GPG_ERR_SIG_EXPIRED
This status indicates that the signature is valid but expired.
For the combined result this status means that all signatures
are valid and expired.

GPG_ERR_KEY_EXPIRED
This status indicates that the signature is valid but the key
used to verify the signature has expired. For the combined
result this status means that all signatures are valid and all
keys are expired.

GPG_ERR_CERT_REVOKED
This status indicates that the signature is valid but the key
used to verify the signature has been revoked. For the com-
bined result this status means that all signatures are valid
and all keys are revoked.

GPG_ERR_BAD_SIGNATURE
This status indicates that the signature is invalid. For the
combined result this status means that all signatures are in-
valid.

GPG_ERR_NO_PUBKEY
This status indicates that the signature could not be verified
due to a missing key. For the combined result this status
means that all signatures could not be checked due to missing keys.

**GPG_ERR_GENERAL**
This status indicates that there was some other error which prevented the signature verification.

**gpgme_sig_notation_t notations**
This is a linked list with the notation data and policy URLs.

**unsigned long timestamp**
The creation timestamp of this signature.

**unsigned long exp_timestamp**
The expiration timestamp of this signature, or 0 if the signature does not expire.

**unsigned int wrong_key_usage : 1**
This is true if the key was not used according to its policy.

**unsigned int pka_trust : 2**
This is set to the trust information gained by means of the PKA system. Values are:

0 No PKA information available or verification not possible.
1 PKA verification failed.
2 PKA verification succeeded.
3 Reserved for future use.

Depending on the configuration of the engine, this metric may also be reflected by the validity of the signature.

**unsigned int chain_model : 1**
**Since**: 1.1.6
This is true if the validity of the signature has been checked using the chain model. In the chain model the time the signature has been created must be within the validity period of the certificate and the time the certificate itself has been created must be within the validity period of the issuing certificate. In contrast the default validation model checks the validity of signature as well at the entire certificate chain at the current time.

**gpgme_validity_t validity**
The validity of the signature.

**gpgme_error_t validity_reason**
If a signature is not valid, this provides a reason why.

**gpgme_pubkey_algo_t**
The public key algorithm used to create this signature.

**gpgme_hash_algo_t**
The hash algorithm used to create this signature.
char *pka_address
The mailbox from the PKA information or NULL.

gpgme_key_t key
SINCE: 1.7.0
An object describing the key used to create the signature. This key object
may be incomplete in that it only conveys information available directly
with a signature. It may also be NULL if such information is not readily
available.

gpgme_verify_result_t
[Data type]
This is a pointer to a structure used to store the result of a gpgme_op_verify op-
eration. After verifying a signature, you can retrieve the pointer to the result with
gpgme_op_verify_result. If the operation failed this might be a NULL pointer. The
structure contains the following member:

gpgme_signature_t signatures
A linked list with information about all signatures for which a verification
was attempted.

char *file_name
This is the filename of the original plaintext message file if it is known,
otherwise this is a null pointer. Warning: The filename is not covered by
the signature.

unsigned int is_mime : 1;
SINCE: 1.11.0
The message claims that the content is a MIME object. Warning: This
flag is not covered by the signature.

7.6.3 Decrypt and Verify

gpgme_error_t gpgme_op_decrypt_verify (gpgme_ctx_t ctx,
  gpgme_data_t cipher, gpgme_data_t plain)
The function gpgme_op_decrypt_verify decrypts the ciphertext in the data object
cipher and stores it into the data object plain. If cipher contains signatures, they will
be verified.

After the operation completed, gpgme_op_decrypt_result and gpgme_op_verify_
result can be used to retrieve more information about the signatures.
Chapter 7: Contexts

If the error code `GPG_ERR_NO_DATA` is returned, `cipher` does not contain any data to decrypt. However, it might still be signed. The information about detected signatures is available with `gpgme_op_verify_result` in this case.

The function returns the error code `GPG_ERR_NO_ERROR` if the ciphertext could be decrypted successfully, `GPG_ERR_INV_VALUE` if `ctx`, `cipher` or `plain` is not a valid pointer, `GPG_ERR_NO_DATA` if `cipher` does not contain any data to decrypt, `GPG_ERR_DECRYPT_FAILED` if `cipher` is not a valid cipher text, `GPG_ERR_BAD_PASSPHRASE` if the passphrase for the secret key could not be retrieved, and passes through any errors that are reported by the crypto engine support routines.

```c
void gpgme_signers_clear (gpgme_ctx_t ctx)
```

The function `gpgme_signers_clear` releases a reference for each key on the signers list and removes the list of signers from the context `ctx`.

Every context starts with an empty list.

---

7.6.4 Sign

A signature can contain signatures by one or more keys. The set of keys used to create a signatures is contained in a context, and is applied to all following signing operations in this context (until the set is changed).

7.6.4.1 Selecting Signers

The key or the keys used to create a signature are stored in the context. The following functions can be used to manipulate this list. If no signer has been set into the context a default key is used for signing.

```c
void gpgme_signers_clear (gpgme_ctx_t ctx)
```

The function `gpgme_signers_clear` releases a reference for each key on the signers list and removes the list of signers from the context `ctx`.

Every context starts with an empty list.

---

When processing mails it is sometimes useful to extract the actual mail address (the “addr-spec”) from a string. GPGME provides this helper function which uses the same semantics as the internal functions in GPGME and GnuPG:

```c
char * gpgme_addrspec_from_uid (const char *uid)
```

Since: 1.7.1

Return the mail address (called “addr-spec” in RFC-5322) from the string `uid` which is assumed to be a user id (called “address” in RFC-5322). All plain ASCII characters (i.e., those with bit 7 cleared) in the result are converted to lowercase. Caller must free the result using `gpgme_free`. Returns NULL if no valid address was found (in which case `ERRNO` is set to `EINVAL`) or for other errors.
gpgme_error_t gpgme_signers_add (gpgme_ctx_t ctx,
    const gpgme_key_t key)

The function gpgme_signers_add adds the key key to the list of signers in the context ctx.

Calling this function acquires an additional reference for the key.

unsigned int gpgme_signers_count (const gpgme_ctx_t ctx)  
SINCE: 1.4.3

The function gpgme_signers_count returns the number of signer keys in the context ctx.

gpgme_key_t gpgme_signers_enum (const gpgme_ctx_t ctx, int seq)

The function gpgme_signers_enum returns the seqth key in the list of signers in the context ctx. An additional reference is acquired for the user.

If seq is out of range, NULL is returned.

7.6.4.2 Creating a Signature

data type
gpgmeSigMode_t

The gpgmeSigMode_t type is used to specify the desired type of a signature. The following modes are available:

GPGME_SIG_MODE_NORMAL

A normal signature is made, the output includes the plaintext and the signature.

GPGME_SIG_MODE_DETACH

A detached signature is made.

GPGME_SIG_MODE_CLEAR

A clear text signature is made. The ASCII armor and text mode settings of the context are ignored.

GPGME_SIG_MODE_ARCHIVE
SINCE: 1.19.0

A signed archive is created from the given files and directories. This feature is currently only supported for the OpenPGP crypto engine and requires GnuPG 2.4.1.

GPGME_SIG_MODE_FILE
SINCE: 1.24.0

The filename set with gpgme_data_set_file_name for the data object plain is passed to gpg, so that gpg reads the plaintext directly from this file instead of from the data object plain. This flag can be combined with GPGME_SIG_MODE_NORMAL, GPGME_SIG_MODE_DETACH, and GPGME_SIG_MODE_CLEAR, but not with GPGME_SIG_MODE_ARCHIVE. This feature is currently only supported for the OpenPGP crypto engine.
The function `gpgme_op_sign` creates a signature for the text in the data object `plain` and returns it in the data object `sig` or writes it directly to the file set with `gpgme_data_set_file_name` for the data object `sig`. The type of the signature created is determined by the ASCII armor (or, if that is not set, by the encoding specified for `sig`), the text mode attributes set for the context `ctx` and the requested signature mode `mode`.

If the signature mode flag `GPGME_SIG_MODE_FILE` is set and a filename has been set with `gpgme_data_set_file_name` for the data object `plain`, then this filename is passed to gpg, so that gpg reads the plaintext directly from this file instead of from the data object `plain`.

If signature mode `GPGME_SIG_MODE_ARCHIVE` is requested then a signed archive is created from the files and directories given as NUL-separated list in the data object `plain`. The paths of the files and directories have to be given as paths relative to the current working directory or relative to the base directory set with `gpgme_data_set_file_name` for the data object `plain`.

After the operation completed successfully, the result can be retrieved with `gpgme_op_sign_result`.

If an S/MIME signed message is created using the CMS crypto engine, the number of certificates to include in the message can be specified with `gpgme_set_include_certs`. See Section 7.4.8 [Included Certificates], page 37.

The function returns the error code `GPG_ERR_NO_ERROR` if the signature could be created successfully, `GPG_ERR_INV_VALUE` if `ctx`, `plain` or `sig` is not a valid pointer, `GPG_ERR_NO_DATA` if the signature could not be created, `GPG_ERR_BAD_PASSPHRASE` if the passphrase for the secret key could not be retrieved, `GPG_ERR_UNUSABLE_SECKEY` if there are invalid signers, and passes through any errors that are reported by the crypto engine support routines.

The function `gpgme_op_sign_start` initiates a `gpgme_op_sign` operation. It can be completed by calling `gpgme_wait` on the context. See Section 7.8.1 [Waiting For Completion], page 100.

The function returns the error code `GPG_ERR_NO_ERROR` if the operation could be started successfully, and `GPG_ERR_INV_VALUE` if `ctx`, `plain` or `sig` is not a valid pointer.

This is a pointer to a structure used to store a part of the result of a `gpgme_op_sign` operation. The structure contains the following members:

- `gpgme_new_signature_t next`  
  This is a pointer to the next new signature structure in the linked list, or NULL if this is the last element.

- `gpgme_sig_mode_t type`  
  The type of this signature.
gpgme_pubkey_algo_t pubkey_algo
The public key algorithm used to create this signature.

gpgme_hash_algo_t hash_algo
The hash algorithm used to create this signature.

unsigned int sig_class
The signature class of this signature. Note that only the values 0, 1, and 2 are well-defined.

long int timestamp
The creation timestamp of this signature.

char *fpr
The fingerprint of the key which was used to create this signature.

gpgme_sign_result_t
[Data type]
This is a pointer to a structure used to store the result of a gpgme_op_sign operation. After successfully generating a signature, you can retrieve the pointer to the result with gpgme_op_sign_result. The structure contains the following members:

gpgme_invalid_key_t invalid_signers
A linked list with information about all invalid keys for which a signature could not be created.

gpgme_new_signature_t signatures
A linked list with information about all signatures created.

gpgme_sign_result_t gpgme_op_sign_result (gpgme_ctx_t ctx)
[Function]
The function gpgme_op_sign_result returns a gpgme_sign_result_t pointer to a structure holding the result of a gpgme_op_sign operation. The pointer is only valid if the last operation on the context was a gpgme_op_sign, gpgme_op_sign_start, gpgme_op_encrypt_sign or gpgme_op_encrypt_sign_start operation. If that operation failed, the function might return a NULL pointer. The returned pointer is only valid until the next operation is started on the context.

7.6.4.3 Signature Notation Data
Using the following functions, you can attach arbitrary notation data to a signature. This information is then available to the user when the signature is verified.

void gpgme_sig_notation_clear (gpgme_ctx_t ctx)
[Function]
Since: 1.1.0
The function gpgme_sig_notation_clear removes the notation data from the context ctx. Subsequent signing operations from this context will not include any notation data.

Every context starts with an empty notation data list.

gpgme_error_t gpgme_sig_notation_add (gpgme_ctx_t ctx, const char *name, const char *value, gpgme_sig_notation_flags_t flags)
Since: 1.1.0
The function gpgme_sig_notation_add adds the notation data with the name name and the value value to the context ctx.
Subsequent signing operations will include this notation data, as well as any other notation data that was added since the creation of the context or the last \texttt{gpgme_sig_notation_clear} operation.

The arguments \textit{name} and \textit{value} must be NUL-terminated strings in human-readable form. The flag \texttt{GPGME_SIG_NOTATION_HUMAN_READABLE} is implied (non-human-readable notation data is currently not supported). The strings must be in UTF-8 encoding.

If \textit{name} is NULL, then \textit{value} should be a policy URL.

The function \texttt{gpgme_sig_notation_add} returns the error code \texttt{GPG_ERR_NO_ERROR} if the notation data could be added successfully, \texttt{GPG_ERR_INV_VALUE} if \textit{ctx} is not a valid pointer, or if \textit{name}, \textit{value} and \textit{flags} are an invalid combination. The function also passes through any errors that are reported by the crypto engine support routines.

\begin{verbatim}
gpgme_sig_notation_t gpgme_sig_notation_get (const gpgme_ctx_t ctx) 
SINCE: 1.1.0

The function \texttt{gpgme_sig_notation_get} returns the linked list of notation data structures that are contained in the context \textit{ctx}.

If \textit{ctx} is not a valid pointer, or there is no notation data added for this context, \texttt{NULL} is returned.
\end{verbatim}

7.6.5 Encrypt

One plaintext can be encrypted for several recipients at the same time. The list of recipients is created independently of any context, and then passed to the encryption operation.

7.6.5.1 Encrypting a Plaintext

\begin{verbatim}
gpgme_error_t gpgme_op_encrypt (gpgme_ctx_t ctx, 
gpgme_key_t recp[], gpgme_encrypt_flags_t flags, gpgme_data_t plain, 
gpgme_data_t cipher) 

The function \texttt{gpgme_op_encrypt} encrypts the plaintext in the data object \textit{plain} for the recipients \textit{recp} and stores the ciphertext in the data object \textit{cipher} or writes it directly to the file set with \texttt{gpgme_data_set_file_name} for the data object \textit{cipher}.

The type of the ciphertext created is determined by the ASCII armor (or, if that is not set, by the encoding specified for \textit{cipher}) and the text mode attributes set for the context \textit{ctx}. If a filename has been set with \texttt{gpgme_data_set_file_name} for the data object \textit{plain} then this filename is stored in the ciphertext.

If the flag \texttt{GPGME_ENCRYPT_FILE} is set and a filename has been set with \texttt{gpgme_data_set_file_name} for the data object \textit{plain}, then this filename is passed to gpg, so that gpg reads the plaintext directly from this file instead of from the data object \textit{plain}.

If the flag \texttt{GPGME_ENCRYPT_ARCHIVE} is set, then an encrypted archive is created from the files and directories given as NUL-separated list in the data object \textit{plain}. The paths of the files and directories have to be given as paths relative to the current working directory or relative to the base directory set with \texttt{gpgme_data_set_file_name} for the data object \textit{plain}.
\end{verbatim}
recp must be a NULL-terminated array of keys. The user must keep references for all keys during the whole duration of the call (but see `gpgme_op_encrypt_start` for the requirements with the asynchronous variant).

The value in flags is a bitwise-or combination of one or multiple of the following bit values:

**GPGME_ENCRYPT_ALWAYS_TRUST**

The `GPGME_ENCRYPT_ALWAYS_TRUST` symbol specifies that all the recipients in recp should be trusted, even if the keys do not have a high enough validity in the keyring. This flag should be used with care; in general it is not a good idea to use any untrusted keys.

For the S/MIME (CMS) protocol this flag allows to encrypt to a certificate without running any checks on the validity of the certificate.

**GPGME_ENCRYPT_NO_ENCRYPT_TO**

Since: 1.2.0

The `GPGME_ENCRYPT_NO_ENCRYPT_TO` symbol specifies that no default or hidden default recipients as configured in the crypto backend should be included. This can be useful for managing different user profiles.

**GPGME_ENCRYPT_NO_COMPRESS**

Since: 1.5.0

The `GPGME_ENCRYPT_NO_COMPRESS` symbol specifies that the plaintext shall not be compressed before it is encrypted. This is in some cases useful if the length of the encrypted message may reveal information about the plaintext.

**GPGME_ENCRYPT_PREPARE**

**GPGME_ENCRYPT_EXPECT_SIGN**

The `GPGME_ENCRYPT_PREPARE` symbol is used with the UI Server protocol to prepare an encryption (i.e., sending the `PREP_ENCRYPT` command). With the `GPGME_ENCRYPT_EXPECT_SIGN` symbol the UI Server is advised to also expect a sign command.

**GPGME_ENCRYPT_SYMMETRIC**

Since: 1.7.0

The `GPGME_ENCRYPT_SYMMETRIC` symbol specifies that the output should be additionally encrypted symmetrically even if recipients are provided. This feature is only supported for the OpenPGP crypto engine.

**GPGME_ENCRYPT_THROW_KEYIDS**

Since: 1.8.0

The `GPGME_ENCRYPT_THROW_KEYIDS` symbols requests that the identifiers for the decryption keys are not included in the ciphertext. On the receiving side, the use of this flag may slow down the decryption process because all available secret keys must be tried. This flag is only honored for OpenPGP encryption.

**GPGME_ENCRYPT_WRAP**

Since: 1.8.0
The `GPGME_ENCRYPT_WRAP` symbol specifies that the input is an OpenPGP message and not a plain data. This is the counterpart to `GPGME_DECRYPT_UNWRAP`.

**GPGME_ENCRYPT_WANT_ADDRESS**

*Since: 1.11.0*

The `GPGME_ENCRYPT_WANT_ADDRESS` symbol requests that all supplied keys or key specifications include a syntactically valid mail address. If this is not the case the operation is not even tried and the error code `GPG_ERR_INV_USER_ID` is returned. Only the address part of the key specification is conveyed to the backend. As of now the key must be specified using the `recpstring` argument of the extended encrypt functions. This feature is currently only supported for the OpenPGP crypto engine.

**GPGME_ENCRYPT_ARCHIVE**

*Since: 1.19.0*

The `GPGME_ENCRYPT_ARCHIVE` symbol specifies that the input is a NUL-separated list of file paths and directory paths that shall be encrypted into an archive. This feature is currently only supported for the OpenPGP crypto engine and requires GnuPG 2.4.1.

**GPGME_ENCRYPT_FILE**

*Since: 1.24.0*

The `GPGME_ENCRYPT_FILE` symbol specifies that the filename set with `gpgme_data_set_file_name` for the data object `plain` is passed to gpg, so that gpg reads the plaintext directly from this file instead of from the data object `plain`. This feature is currently only supported for the OpenPGP crypto engine.

If `GPG_ERR_UNUSABLE_PUBKEY` is returned, some recipients in `recp` are invalid, but not all. In this case the plaintext might be encrypted for all valid recipients and returned in `cipher` (if this happens depends on the crypto engine). More information about the invalid recipients is available with `gpgme_op_encrypt_result`.

If `recp` is `NULL`, symmetric rather than public key encryption is performed. Symmetrically encrypted cipher text can be deciphered with `gpgme_op_decrypt`. Note that in this case the crypto backend needs to retrieve a passphrase from the user. Symmetric encryption is currently only supported for the OpenPGP crypto backend.

The function returns the error code `GPG_ERR_NO_ERROR` if the ciphertext could be created successfully, `GPG_ERR_INV_VALUE` if `ctx`, `recp`, `plain` or `cipher` is not a valid pointer, `GPG_ERR_UNUSABLE_PUBKEY` if `recp` contains some invalid recipients, `GPG_ERR_BAD_PASSPHRASE` if the passphrase for the symmetric key could not be retrieved, and passes through any errors that are reported by the crypto engine support routines.

```c
void gpgme_op_encrypt_start (gpgme_ctx_t ctx, gpgme_key_t recp[], gpgme_encrypt_flags_t flags, gpgme_data_t plain, gpgme_data_t cipher)
```

The function `gpgme_op_encrypt_start` initiates a `gpgme_op_encrypt` operation. It can be completed by calling `gpgme_wait` on the context. See Section 7.8.1 [Waiting For Completion], page 100.
References to the keys only need to be held for the duration of this call. The user can release its references to the keys after this function returns, even if the operation is not yet finished.

The function returns the error code `GPG_ERR_NO_ERROR` if the operation could be started successfully, `GPG_ERR_INV_VALUE` if `ctx`, `rset`, `plain` or `cipher` is not a valid pointer, and `GPG_ERR_UNUSABLE_PUBKEY` if `rset` does not contain any valid recipients.

```c
[gpgme_error_t] gpgme_op_encrypt_ext (gpgme_ctx_t ctx,
gpgme_key_t recp[], const char *recpstring, gpgme_encrypt_flags_t flags,
gpgme_data_t plain, gpgme_data_t cipher)
```

Since: 1.11.0

This is an extended version of `gpgme_op_encrypt` with `recpstring` as additional parameter. If `recp` is NULL and `recpstring` is not NULL, the latter is expected to be a linefeed delimited string with the set of key specifications. In contrast to `recp` the keys are given directly as strings and there is no need to first create key objects. Leading and trailing white space is remove from each line in `recpstring`. The keys are then passed verbatim to the backend engine.

For the OpenPGP backend several special keywords are supported to modify the operation. These keywords are given instead of a key specification. The currently supported keywords are:

--hidden
--no-hidden

These keywords toggle between normal and hidden recipients for all following key specifications. When a hidden recipient is requested the gpg option `-R` (or `-F` in file mode) is used instead of `-r` (`-f` in file mode).

--file
--no-file

These keywords toggle between regular and file mode for all following key specification. In file mode the option `-f` or `-F` is passed to gpg. At least GnuPG version 2.1.14 is required to handle these options. The `GPGME_ENCRYPT_WANT_ADDRESS` flag is ignored in file mode.

--

This keyword disables all keyword detection up to the end of the string. All keywords are treated as verbatim arguments.

To create a `recpstring` it is often useful to employ a `strconcat` style function. For example this function creates a string to encrypt to two keys:

```c
char *xbuild_recpstring (const char *key1, const char *key2)
{
    char *result = gpgrt_strconcat ("--\n", key1, \"\n\", key2, NULL);
    if (!result)
        { perror ("strconcat failed"); exit (2); }
    return result;
}
```

Note the use of the double dash here; unless you want to specify a keyword, it is a good idea to avoid any possible trouble with key specifications starting with a
double dash. The used strconcat function is available in Libgpg-error 1.28 and later; Libgpg-error (aka Gpgrt) is a dependency of GPGME. The number of arguments to `gpgrt_strconcat` is limited to 47 but that should always be sufficient. In case a larger and non-fixed number of keys are to be supplied the following code can be used:

```c
char *
xbuild_long_recpstring (void)
{
  gpgrt_stream_t memfp;
  const char *s;
  void *result;

  memfp = gpgrt_fopenmem (0, "w+b");
  if (!memfp)
    { perror ("fopenmem failed"); exit (2); }
  gpgrt_fputs ("--", memfp);
  while ((s = get_next_keyspec ()))
    {
      gpgrt_fputc ('
', memfp);
      gpgrt_fputs (s, memfp);
    }
  gpgrt_fputc (0, memfp);
  if (gpgrt_ferror (memfp))
    { perror ("writing to memstream failed"); exit (2); }
  if (gpgrt_fclose_snatch (memfp, &result, NULL))
    { perror ("fclose_snatch failed"); exit (2); }
  return result;
}
```

In this example `get_next_keyspec` is expected to return the next key to be added to the string. Please take care: Encrypting to a large number of recipients is often questionable due to security reasons and also for the technicality that all keys are currently passed on the command line to `gpg` which has as a platform specific length limitation.

```c
gpgme_error_t gpgme_op_encrypt_ext_start (gpgme_ctx_t ctx, [Function]
gpgme_key_t recp[]], const char *recpstring, gpgme_encrypt_flags_t flags,
gpgme_data_t plain, gpgme_data_t cipher)
SINCE: 1.11.0
```

This is an extended version of `gpgme_op_encrypt_start` with `recpstring` as additional parameter. If `recp` is `NULL` and `recpstring` is not `NULL`, the latter is expected to be a linefeed delimited string with the set of key specifications. In contrast to `recp` the keys are given directly as strings and there is no need to first create key objects. The keys are passed verbatim to the backend engine.
gpgme_encrypt_result_t

This is a pointer to a structure used to store the result of a gpgme_op_encrypt operation. After successfully encrypting data, you can retrieve the pointer to the result with gpgme_op_encrypt_result. The structure contains the following members:

gpgme_invalid_key_t invalid_recipients

A linked list with information about all invalid keys for which the data could not be encrypted.

gpgme_encrypt_result_t gpgme_op_encrypt_result

This function returns a gpgme_encrypt_result_t pointer to a structure holding the result of a gpgme_op_encrypt operation. The pointer is only valid if the last operation on the context was a gpgme_op_encrypt, gpgme_op_encrypt_start, gpgme_op_sign or gpgme_op_sign_start operation. If this operation failed, this might be a NULL pointer. The returned pointer is only valid until the next operation is started on the context.

gpgme_op_encrypt_sign (gpgme_ctx_t ctx, gpgme_key_t recp[], gpgme_encrypt_flags_t flags, gpgme_data_t plain, gpgme_data_t cipher)

The function gpgme_op_encrypt_sign does a combined encrypt and sign operation. It is used like gpgme_op_encrypt, but the ciphertext also contains signatures for the signers listed in ctx.

The combined encrypt and sign operation is currently only available for the OpenPGP crypto engine.

gpgme_op_encrypt_sign_start (gpgme_ctx_t ctx, gpgme_key_t recp[], gpgme_encrypt_flags_t flags, gpgme_data_t plain, gpgme_data_t cipher)

The function gpgme_op_encrypt_sign_start initiates a gpgme_op_encrypt_sign operation. It can be completed by calling gpgme_wait on the context. See Section 7.8.1 [Waiting For Completion], page 100.

The function returns the error code GPG_ERR_NO_ERROR if the operation could be started successfully, and GPG_ERR_INV_VALUE if ctx, rset, plain or cipher is not a valid pointer.

gpgme_op_encrypt_sign_ext (gpgme_ctx_t ctx, gpgme_key_t recp[], const char *recpstring, gpgme_encrypt_flags_t flags, gpgme_data_t plain, gpgme_data_t cipher)

Since: 1.11.0

This is an extended version of gpgme_op_encrypt_sign with recpstring as additional parameter. If recp is NULL and recpstring is not NULL, the latter is expected to be a linefeed delimited string with the set of key specifications. In contrast to recp the keys are given directly as strings and there is no need to first create the key objects. The keys are passed verbatim to the backend engine.
Chapter 7: Contexts

---

**gpgme_error_t gpgme_op_encrypt_sign_ext_start**  
(Function)  

gpgme_ctx_t ctx, gpgme_key_t recp[], const char *recpstring,  
gpgme_encrypt_flags_t flags, gpgme_data_t plain, gpgme_data_t cipher)  

Since: 1.11.0  

This is an extended version of **gpgme_op_encrypt_sign_start** with recpstring as additional parameter. If recp is NULL and recpstring is not NULL, the latter is expected to be a linefeed delimited string with the set of key specifications. In contrast to recp the keys are given directly as strings and there is no need to first create the key objects. The keys are passed verbatim to the backend engine.

### 7.7 Miscellaneous operations

Here are some support functions which are sometimes useful.

#### 7.7.1 Running other Programs

GPGME features an internal subsystem to run the actual backend engines. Along with data abstraction object this subsystem can be used to run arbitrary simple programs which even need not be related to cryptographic features. It may for example be used to run tools which are part of the GnuPG system but are not directly accessible with the GPGME API.

**gpgme_error_t gpgme_op_spawn**  
(Function)  

gpgme_ctx_t ctx, const char *file, const char *argv[], gpgme_data_t datain, gpgme_data_t dataout,  
gpgme_data_t dataerr, unsigned int flags)  

Since: 1.5.0  

The function **gpgme_op_spawn** runs the program file with the arguments taken from the NULL terminated array argv. If no arguments are required argv may be given as NULL. In the latter case or if argv[0] is the empty string, GPGME uses the basename of file for argv[0]. The file descriptors stdin, stdout, and stderr are connected to the data objects datain, dataout, and dataerr. If NULL is passed for one of these data objects the corresponding file descriptor is connected to ‘/dev/null’.

The value in flags is a bitwise-or combination of one or multiple of the following bit values:

- **GPGME_SPAWN_DETACHED**  
  Since: 1.5.0  
  Under Windows this flag inhibits the allocation of a new console for the program. This is useful for a GUI application which needs to call a command line helper tool.

- **GPGME_SPAWN_ALLOW_SET_FG**  
  Since: 1.5.0  
  Under Windows this flag allows the called program to put itself into the foreground.

**gpgme_error_t gpgme_op_spawn_start**  
(Function)  

gpgme_ctx_t ctx, const char *file, const char *argv[], gpgme_data_t datain, gpgme_data_t dataout,  
gpgme_data_t dataerr, unsigned int flags)  

Since: 1.5.0  

This is the asynchronous variant of **gpgme_op_spawn**.
7.7.2 Using the Assuan protocol

The Assuan protocol can be used to talk to arbitrary Assuan servers. By default it is connected to the GnuPG agent, but it may be connected to arbitrary servers by using `gpgme_ctx_set_engine_info`, passing the location of the servers socket as `file_name` argument, and an empty string as `home_dir` argument.

The Assuan protocol functions use three kinds of callbacks to transfer data:

```c
typedef gpgme_error_t (*gpgme_assuan_data_cb_t) (void *opaque, const void *data, size_t datalen);
```

Since: 1.2.0

This callback receives any data sent by the server. `opaque` is the pointer passed to `gpgme_op_assuan_transact_start`, `data` of length `datalen` refers to the data sent.

```c
typedef gpgme_error_t (*gpgme_assuan_inquire_cb_t) (void *opaque, const char *name, const char *args, gpgme_data_t *r_data);
```

Since: 1.2.0

This callback is used to provide additional data to the Assuan server. `opaque` is the pointer passed to `gpgme_op_assuan_transact_start`, `name` and `args` specify what kind of data the server requested, and `r_data` is used to return the actual data.

Note: Returning data is currently not implemented in GPGME.

```c
typedef gpgme_error_t (*gpgme_assuan_status_cb_t) (void *opaque, const char *status, const char *args);
```

Since: 1.2.0

This callback receives any status lines sent by the server. `opaque` is the pointer passed to `gpgme_op_assuan_transact_start`, `status` and `args` denote the status update sent.

```c
gpgme_error_t gpgme_op_assuan_transact_start (gpgme_ctx_t ctx, const char *command, gpgme_assuan_data_cb_t data_cb,
    void *data_cb_value, gpgme_assuan_inquire_cb_t inquire_cb,
    void *inquire_cb_value, gpgme_assuan_status_cb_t status_cb,
    void *status_cb_value);
```

Since: 1.2.0

Send the Assuan `command` and return results via the callbacks. Any callback may be `NULL`. The result of the operation may be retrieved using `gpgme_wait_ext`.

Asynchronous variant.

```c
gpgme_error_t gpgme_op_assuan_transact_ext (gpgme_ctx_t ctx, const char *command, gpgme_assuan_data_cb_t data_cb,
    void *data_cb_value, gpgme_assuan_inquire_cb_t inquire_cb,
    void *inquire_cb_value, gpgme_assuan_status_cb_t status_cb,
    void *status_cb_value, gpgme_error_t *op_err);
```

Since: 1.2.0

Send the Assuan `command` and return results via the callbacks. The result of the operation is returned in `op_err`.

Synchronous variant.
7.7.3 How to check for software updates

The GnuPG Project operates a server to query the current versions of software packages related to GnuPG. GPGME can be used to access this online database and check whether a new version of a software package is available.

`gpgme_query_swdb_result_t`  
**Data type**

This is a pointer to a structure used to store the result of a `gpgme_op_query_swdb` operation. After a successful call to that function, you can retrieve the pointer to the result with `gpgme_op_query_swdb_result`. The structure contains the following member:

- `name` This is the name of the package.
- `iversion` The currently installed version or an empty string. This value is either a copy of the argument given to `gpgme_op_query_swdb` or the version of the installed software as figured out by GPGME or GnuPG.
- `created` This gives the date the file with the list of version numbers has originally been created by the GnuPG project.
- `retrieved` This gives the date the file was downloaded.
- `warning` If this flag is set either an error has occurred or some of the information in this structure are not properly set. For example if the version number of the installed software could not be figured out, the `update` flag may not reflect a required update status.
- `update` If this flag is set an update of the software is available.
- `urgent` If this flag is set an available update is important.
- `noinfo` If this flag is set, no valid information could be retrieved.
- `unknown` If this flag is set the given `name` is not known.
- `tooold` If this flag is set the available information is not fresh enough.
- `error` If this flag is set some other error has occurred.
- `version` The version string of the latest released version.
- `reldate` The release date of the latest released version.

`gpgme_error_t gpgme_op_query_swdb (gpgme_ctx_t ctx, const char *name, const char *iversion, gpgme_data_t reserved)`  
**Function**

Query the software version database for software package `name` and check against the installed version given by `iversion`. If `iversion` is given as `NULL` a check is only done if GPGME can figure out the version by itself (for example when using "gpgme" or "gnupg"). If `NULL` is used for `name` the current gpgme version is checked. `reserved` must be set to 0.
The function `gpgme_op_query_swdb_result` returns a `gpgme_query_swdb_result_t` pointer to a structure holding the result of a `gpgme_op_query_swdb` operation. The pointer is only valid if the last operation on the context was a successful call to `gpgme_op_query_swdb`. If that call failed, the result might be a NULL pointer. The returned pointer is only valid until the next operation is started on the context `ctx`.

Here is an example on how to check whether GnuPG is current:

```c
#include <gpgme.h>

int main(void)
{
  gpg_error_t err;
  gpgme_ctx_t ctx;
  gpgme_query_swdb_result_t result;

  gpgme_check_version(NULL);
  err = gpgme_new(&ctx);
  if (err)
    fprintf(stderr, "error creating context: %s\n", gpg_strerror(err));
  else
  {
    gpgme_set_protocol(ctx, GPGME_PROTOCOL_GPGCONF);

    err = gpgme_op_query_swdb(ctx, "gnupg", NULL, 0);
    if (err)
      fprintf(stderr, "error querying swdb: %s\n", gpg_strerror(err));
    else
    {
      result = gpgme_op_query_swdb_result(ctx);
      if (!result)
        fprintf(stderr, "error querying swdb\n");
      else if (!result->warning & !result->update)
        printf("GnuPG version %s is current\n", result->iversion);
      else if (!result->warning & result->update)
        printf("GnuPG version %s can be updated to %s\n", result->iversion, result->version);
      else
        fprintf(stderr, "error finding the update status\n");
    }
  gpgme_release(ctx);
}
return 0;
```
7.8 Run Control

GPGME supports running operations synchronously and asynchronously. You can use asynchronous operation to set up a context up to initiating the desired operation, but delay performing it to a later point.

Furthermore, you can use an external event loop to control exactly when GPGME runs. This ensures that GPGME only runs when necessary and also prevents it from blocking for a long time.

7.8.1 Waiting For Completion

\[
gpgme_ctx_t gpgme_wait (gpgme_ctx_t ctx, gpgme_error_t *status, \text{ [Function]} int hang)\]

The function \textit{gpgme_wait} continues the pending operation within the context \textit{ctx}. In particular, it ensures the data exchange between GPGME and the crypto backend and watches over the run time status of the backend process.

If \textit{hang} is true, the function does not return until the operation is completed or cancelled. Otherwise the function will not block for a long time.

The error status of the finished operation is returned in \textit{status} if \textit{gpgme_wait} does not return \textit{NULL}.

The \textit{ctx} argument can be \textit{NULL}. In that case, \textit{gpgme_wait} waits for any context to complete its operation.

\textit{gpgme_wait} can be used only in conjunction with any context that has a pending operation initiated with one of the \textit{gpgme_op_*_start} functions except \textit{gpgme_op_keylist_start} and \textit{gpgme_op_trustlist_start} (for which you should use the corresponding \textit{gpgme_op_*_next} functions). If \textit{ctx} is \textit{NULL}, all of such contexts are waited upon and possibly returned. Synchronous operations running in parallel, as well as key and trust item list operations, do not affect \textit{gpgme_wait}.

In a multi-threaded environment, only one thread should ever call \textit{gpgme_wait} at any time, regardless of whether \textit{ctx} is specified or not. This means that all calls to this function should be fully synchronized by locking primitives. It is safe to start asynchronous operations while a thread is running in \textit{gpgme_wait}.

The function returns the \textit{ctx} of the context which has finished the operation. If \textit{hang} is false, and the timeout expires, \textit{NULL} is returned and \textit{*status} will be set to 0. If an error occurs, \textit{NULL} is returned and the error is returned in \textit{*status}.

7.8.2 Using External Event Loops

GPGME hides the complexity of the communication between the library and the crypto engine. The price of this convenience is that the calling thread can block arbitrary long waiting for the data returned by the crypto engine. In single-threaded programs, in particular if they are interactive, this is an unwanted side-effect. OTOH, if \textit{gpgme_wait} is used without the \textit{hang} option being enabled, it might be called unnecessarily often, wasting CPU time that could be used otherwise.
The I/O callback interface described in this section lets the user take control over what happens when. GPGME will provide the user with the file descriptors that should be monitored, and the callback functions that should be invoked when a file descriptor is ready for reading or writing. It is then the user’s responsibility to decide when to check the file descriptors and when to invoke the callback functions. Usually this is done in an event loop, that also checks for events in other parts of the program. If the callback functions are only called when the file descriptors are ready, GPGME will never block. This gives the user more control over the program flow, and allows to perform other tasks when GPGME would block otherwise.

By using this advanced mechanism, GPGME can be integrated smoothly into GUI toolkits like GTK+ even for single-threaded programs.

### 7.8.2.1 I/O Callback Interface

**gpgme_error_t (*gpgme_io_cb_t) (void *data, int fd) [Data type]**

The `gpgme_io_cb_t` type is the type of functions which GPGME wants to register as I/O callback handlers using the `gpgme_register_io_cb_t` functions provided by the user.

`data` and `fd` are provided by GPGME when the I/O callback handler is registered, and should be passed through to the handler when it is invoked by the user because it noticed activity on the file descriptor `fd`.

The callback handler always returns 0, but you should consider the return value to be reserved for later use.

**gpgme_error_t (*gpgme_register_io_cb_t) (void *data, int fd, int dir, gpgme_io_cb_t fnc, void *fnc_data, void **tag) [Data type]**

The `gpgme_register_io_cb_t` type is the type of functions which can be called by GPGME to register an I/O callback function `fnc` for the file descriptor `fd` with the user. `fnc_data` should be passed as the first argument to `fnc` when the handler is invoked (the second argument should be `fd`). If `dir` is 0, `fnc` should be called by the user when `fd` is ready for writing. If `dir` is 1, `fnc` should be called when `fd` is ready for reading.

`data` was provided by the user when registering the `gpgme_register_io_cb_t` function with GPGME and will always be passed as the first argument when registering a callback function. For example, the user can use this to determine the event loop to which the file descriptor should be added.

GPGME will call this function when a crypto operation is initiated in a context for which the user has registered I/O callback handler functions with `gpgme_set_io_cbs`. It can also call this function when it is in an I/O callback handler for a file descriptor associated to this context.

The user should return a unique handle in `tag` identifying this I/O callback registration, which will be passed to the `gpgme_register_io_cb_t` function without interpretation when the file descriptor should not be monitored anymore.
void (*gpgme_remove_io_cb_t) (void *tag)  

The gpgme_remove_io_cb_t type is the type of functions which can be called by GPGME to remove an I/O callback handler that was registered before. tag is the handle that was returned by the gpgme_register_io_cb_t for this I/O callback.

GPGME can call this function when a crypto operation is in an I/O callback. It will also call this function when the context is destroyed while an operation is pending.

enum gpgme_event_io_t  

The gpgme_event_io_t type specifies the type of an event that is reported to the user by GPGME as a consequence of an I/O operation. The following events are defined:

GPGME_EVENT_START
The operation is fully initialized now, and you can start to run the registered I/O callback handlers now. Note that registered I/O callback handlers must not be run before this event is signalled. type_data is NULL and reserved for later use.

GPGME_EVENT_DONE
The operation is finished, the last I/O callback for this operation was removed. The accompanying type_data points to a struct gpgme_io_event_done_data variable that contains the status of the operation that finished. This event is signalled after the last I/O callback has been removed.

GPGME_EVENT_NEXT_KEY
In a gpgme_op_keylist_start operation, the next key was received from the crypto engine. The accompanying type_data is a gpgme_key_t variable that contains the key with one reference for the user.

void (*gpgme_event_io_cb_t) (void *data, gpgme_event_io_t type, void *type_data)  

The gpgme_event_io_cb_t type is the type of functions which can be called by GPGME to signal an event for an operation running in a context which has I/O callback functions registered by the user.

data was provided by the user when registering the gpgme_event_io_cb_t function with GPGME and will always be passed as the first argument when registering a callback function. For example, the user can use this to determine the context in which this event has occurred.

type will specify the type of event that has occurred. type_data specifies the event further, as described in the above list of possible gpgme_event_io_t types.

GPGME can call this function in an I/O callback handler.

7.8.2.2 Registering I/O Callbacks

struct gpgme_io_cbs  

This structure is used to store the I/O callback interface functions described in the previous section. It has the following members:
gpgme_register_io_cb_t add
This is the function called by GPGME to register an I/O callback handler. It must be specified.

void *add_priv
This is passed as the first argument to the add function when it is called by GPGME. For example, it can be used to determine the event loop to which the file descriptor should be added.

gpgme_remove_io_cb_t remove
This is the function called by GPGME to remove an I/O callback handler. It must be specified.

gpgme_event_io_cb_t event
This is the function called by GPGME to signal an event for an operation. It must be specified, because at least the start event must be processed.

void *event_priv
This is passed as the first argument to the event function when it is called by GPGME. For example, it can be used to determine the context in which the event has occurred.

void gpgme_set_io_cbs (gpgme_ctx_t ctx, [Function]
 struct gpgme_io_cbs *io_cbs)
The function gpgme_set_io_cbs enables the I/O callback interface for the context ctx. The I/O callback functions are specified by io_cbs.
If io_cbs->add is NULL, the I/O callback interface is disabled for the context, and normal operation is restored.

void gpgme_get_io_cbs (gpgme_ctx_t ctx, [Function]
 struct gpgme_io_cbs *io_cbs)
The function gpgme_get_io_cbs returns the I/O callback functions set with gpgme_set_io_cbs in io_cbs.

7.8.2.3 I/O Callback Example
To actually use an external event loop, you have to implement the I/O callback functions that are used by GPGME to register and unregister file descriptors. Furthermore, you have to actually monitor these file descriptors for activity and call the appropriate I/O callbacks.

The following example illustrates how to do that. The example uses locking to show in which way the callbacks and the event loop can run concurrently. For the event loop, we use a fixed array. For a real-world implementation, you should use a dynamically sized structure because the number of file descriptors needed for a crypto operation in GPGME is not predictable.

#include <assert.h>
#include <errno.h>
#include <stdlib.h>
#include <pthread.h>
#include <sys/types.h>
#include <gpgme.h>
/* The following structure holds the result of a crypto operation. */
struct op_result
{
    int done;
    gpgme_error_t err;
};

/* The following structure holds the data associated with one I/O callback. */
struct one_fd
{
    int fd;
    int dir;
    gpgme_io_cb_t fnc;
    void *fnc_data;
    void *loop;
};

struct event_loop
{
    pthread_mutex_t lock;
#define MAX_FDS 32
    /* Unused slots are marked with FD being -1. */
    struct one_fd fds[MAX_FDS];
};

The following functions implement the I/O callback interface.

gpgme_error_t
add_io_cb (void *data, int fd, int dir, gpgme_io_cb_t fnc, void *fnc_data,
    void **r_tag)
{
    struct event_loop *loop = data;
    struct one_fd *fds = loop->fds;
    int i;

    pthread_mutex_lock (&loop->lock);
    for (i = 0; i < MAX_FDS; i++)
    {
        if (fds[i].fd == -1)
        {
            fds[i].fd = fd;
            fds[i].dir = dir;
            fds[i].fnc = fnc;
            fds[i].fnc_data = fnc_data;
            fds[i].loop = loop;
            break;
        
    
}
The final missing piece is the event loop, which will be presented next. We only support waiting for the success of a single operation.

```c
int
do_select (struct event_loop *loop)
{
    fd_set rfds;
    fd_set wfds;
    int i, n;
    int any = 0;
    struct timeval tv;
    struct one_fd *fdlist = loop->fds;

    pthread_mutex_lock (&loop->lock);
    pthread_mutex_unlock (&loop->lock);

    FD_ZERO (&rfds);
    FD_ZERO (&wfds);

    pthread_mutex_lock (&loop->lock);
    return gpg_error (GPG_ERR_GENERAL);
}

void
remove_io_cb (void *tag)
{
    struct one_fd *fd = tag;
    struct event_loop *loop = fd->loop;

    pthread_mutex_lock (&loop->lock);
    fd->fd = -1;
    pthread_mutex_unlock (&loop->lock);
}

void
event_io_cb (void *data, gpgme_event_io_t type, void *type_data)
{
    struct op_result *result = data;

    /* We don’t support list operations here. */
    if (type == GPGME_EVENT_DONE)
    {
        result->done = 1;
        result->err = *type_data;
    }
}
```

```c
The final missing piece is the event loop, which will be presented next. We only support waiting for the success of a single operation.

```
FD_ZERO (&wfds);
for (i = 0; i < MAX_FDS; i++)
    if (fdlist[i].fd != -1)
        FD_SET(fdlist[i].fd, fdlist[i].dir ? &rfds : &wfds);
pthread_mutex_unlock(&loop->lock);

tv.tv_sec = 0;
tv.tv_usec = 1000;

do {
    n = select(FD_SETSIZE, &rfds, &wfds, NULL, &tv);
} while (n < 0 && errno == EINTR);

if (n < 0)
    return n; /* Error or timeout. */

pthread_mutex_lock(&loop->lock);
for (i = 0; i < MAX_FDS && n; i++)
{
    if (fdlist[i].fd != -1)
    {
        if (FD_ISSET(fdlist[i].fd, fdlist[i].dir ? &rfds : &wfds))
        {
            assert(n);
            n--;
            any = 1;
            /* The I/O callback handler can register/remove callbacks,
             * so we have to unlock the file descriptor list. */
            pthread_mutex_unlock(&loop->lock);
            (*fdlist[i].fnc)(fdlist[i].fnc_data, fdlist[i].fd);
            pthread_mutex_lock(&loop->lock);
        }
    }
}
pthread_mutex_unlock(&loop->lock);
return any;
}

void
wait_for_op(struct event_loop *loop, struct op_result *result)
{
    int ret;

do {

The main function shows how to put it all together.

```c
int main (int argc, char *argv[])
{
    struct event_loop loop;
    struct op_result result;
    gpgme_ctx_t ctx;
    gpgme_error_t err;
    gpgme_data_t sig, text;
    int i;
    pthread_mutexattr_t attr;
    struct gpgme_io_cbs io_cbs = {
        add_io_cb, 
        &loop, 
        remove_io_cb, 
        event_io_cb, 
        &result
    };

    init_gpgme ();

    /* Initialize the loop structure. */
    /* The mutex must be recursive, since remove_io_cb (which acquires a
     * lock) can be called while holding a lock acquired in do_select. */
    pthread_mutexattr_init (&attr);
    pthread_mutexattr_settype (&attr, PTHREAD_MUTEX_RECURSIVE);
    pthread_mutex_init (&loop.lock, &attr);
    pthread_mutexattr_destroy (&attr);

    for (i = 0; i < MAX_FDS; i++)
        loop.fds[i].fd = -1;

    /* Initialize the result structure. */
    result.done = 0;

    err = gpgme_data_new_from_file (&sig, "signature", 1);
    if (!err)
        err = gpgme_data_new_from_file (&text, "text", 1);
    if (!err)
        err = gpgme_new (&ctx);
```

```c
ret = do_select (loop);
}
while (ret >= 0 && !result->done);
}
```
if (!err)
{
    gpgme_set_io_cbs (ctx, &io_cbs);
    err = gpgme_op_verify_start (ctx, sig, text, NULL);
}
if (err)
{
    fprintf (stderr, "gpgme error: %s: %s\n",
             gpgme_strsource (err), gpgme_strerror (err));
    exit (1);
}

wait_for_op (&loop, &result);
if (!result.done)
{
    fprintf (stderr, "select error\n");
    exit (1);
}
if (!result.err)
{
    fprintf (stderr, "verification failed: %s: %s\n",
             gpgme_strsource (result.err), gpgme_strerror (result.err));
    exit (1);
}
/* Evaluate verify result. */
...
return 0;
}

7.8.2.4 I/O Callback Example GTK+

The I/O callback interface can be used to integrate GPGME with the GTK+ event loop. The following code snippets shows how this can be done using the appropriate register and remove I/O callback functions. In this example, the private data of the register I/O callback function is unused. The event notifications is missing because it does not require any GTK+ specific setup.

#include <gtk/gtk.h>

struct my_gpgme_io_cb
{
    gpgme_io_cb_t fnc;
    void *fnc_data;
    guint input_handler_id
};

void
my_gpgme_io_cb (gpointer data, gint source, GdkInputCondition condition)
{  
struct my_gpgme_io_cb *iocb = data;  
(*(iocb->fnc)) (iocb->data, source);
}

void
my_gpgme_remove_io_cb (void *data)
{
  struct my_gpgme_io_cb *iocb = data;
  gtk_input_remove (data->input_handler_id);
}

void
my_gpgme_register_io_callback (void *data, int fd, int dir, gpgme_io_cb_t fnc,  
void *fnc_data, void **tag)
{
  struct my_gpgme_io_cb *iocb = g_malloc (sizeof (struct my_gpgme_io_cb));  
iocb->fnc = fnc;
  iocb->data = fnc_data;
  iocb->input_handler_id = gtk_input_add_full (fd, dir
    ? GDK_INPUT_READ
    : GDK_INPUT_WRITE,
    my_gpgme_io_callback,
    0, iocb, NULL);

  *tag = iocb;
  return 0;
}

7.8.2.5 I/O Callback Example GDK

The I/O callback interface can also be used to integrate GPGME with the GDK event loop. The following code snippets shows how this can be done using the appropriate register and remove I/O callback functions. In this example, the private data of the register I/O callback function is unused. The event notifications is missing because it does not require any GDK specific setup.

It is very similar to the GTK+ example in the previous section.

#include <gdk/gdk.h>

struct my_gpgme_io_cb
{
  gpgme_io_cb_t fnc;
  void *fnc_data;
  gint tag;
};

void
my_gpgme_io_cb (gpointer data, gint source, GdkInputCondition condition)

{
    struct my_gpgme_io_cb *iocb = data;
    (*(iocb->fnc)) (iocb->data, source);
}

void
my_gpgme_remove_io_cb (void *data)
{
    struct my_gpgme_io_cb *iocb = data;
    gdk_input_remove (data->tag);
}

void
my_gpgme_register_io_callback (void *data, int fd, int dir, gpgme_io_cb_t fnc,
                                void *fnc_data, void **tag)
{
    struct my_gpgme_io_cb *iocb = g_malloc (sizeof (struct my_gpgme_io_cb));
    iocb->fnc = fnc;
    iocb->data = fnc_data;
    iocb->tag = gtk_input_add_full (fd, dir ? GDK_INPUT_READ : GDK_INPUT_WRITE,
                                     my_gpgme_io_callback, iocb, NULL);
    *tag = iocb;
    return 0;
}

7.8.2.6 I/O Callback Example Qt
The I/O callback interface can also be used to integrate GPGME with the Qt event loop.
The following code snippets show how this can be done using the appropriate register and
remove I/O callback functions. In this example, the private data of the register I/O callback
function is unused. The event notifications is missing because it does not require any Qt
specific setup.

#include <qsocketnotifier.h>
#include <qapplication.h>

struct IOCB {
    IOCB( GpgmeIOCb f, void * d, QSocketNotifier * n )
        : func( f ), data( d ), notifier( n ) {}
    GpgmeIOCb func;
    void * data;
    QSocketNotifier * notifier;
}

class MyApp : public QApplication {

    // ...
static void registerGpgmeIOCallback( void * data, int fd, int dir,  
  GpgmeIOCb func, void * func_data,  
  void ** tag ) {
  QSocketNotifier * n =  
    new QSocketNotifier( fd, dir ? QSocketNotifier::Read  
      : QSocketNotifier::Write );
  connect( n, SIGNAL(activated(int)),  
    qApp, SLOT(slotGpgmeIOCallback(int)) );
  qApp->mIOCBs.push_back( IOCB( func, func_data, n ) );
  *tag = (void*)n;
}

static void removeGpgmeIOCallback( void * tag ) {
  if ( !tag ) return;
  QSocketNotifier * n = static_cast<QSocketNotifier*>( tag );
  for ( QValueList<IOCB>::iterator it = qApp->mIOCBs.begin() ;  
    it != qApp->mIOCBs.end() ; ++it )
    if ( it->notifier == n ) {
      delete it->notifier;
      qApp->mIOCBs.erase( it );
      return;
    }
}

public slots:
  void slotGpgmeIOCallback( int fd ) {  
    for ( QValueList<IOCB>::const_iterator it = mIOCBs.begin() ;  
      it != mIOCBs.end() ; ++it )
      if ( it->notifier && it->notifier->socket() == fd )  
        (*(it->func)) ( it->func_data, fd );
  }

  // ...

private:
  QValueList<IOCB> mIOCBs;
  // ...
};

### 7.8.3 Cancellation

Sometimes you do not want to wait for an operation to finish. GPGME provides two different functions to achieve that. The function `gpgme_cancel` takes effect immediately. When it returns, the operation is effectively canceled. However, it has some limitations and cannot be used with synchronous operations. In contrast, the function `gpgme_cancel_async` can be used with any context and from any thread, but it is not guaranteed to take effect
immediately. Instead, cancellation occurs at the next possible time (typically the next time
I/O occurs in the target context).

`gpgme_ctx_t gpgme_cancel (gpgme_ctx_t ctx)`

Since: 0.4.5
The function `gpgme_cancel` attempts to cancel a pending operation in the context `ctx`. This only works if you use the global event loop or your own event loop.

If you use the global event loop, you must not call `gpgme_wait` during cancellation. After successful cancellation, you can call `gpgme_wait` (optionally waiting on `ctx`), and the context `ctx` will appear as if it had finished with the error code `GPG_ERR_CANCEL`.

If you use an external event loop, you must ensure that no I/O callbacks are invoked for this context (for example by halting the event loop). On successful cancellation, all registered I/O callbacks for this context will be unregistered, and a `GPGME_EVENT_DONE` event with the error code `GPG_ERR_CANCEL` will be signalled.

The function returns an error code if the cancellation failed (in this case the state of `ctx` is not modified).

`gpgme_ctx_t gpgme_cancel_async (gpgme_ctx_t ctx)`

Since: 1.1.7
The function `gpgme_cancel_async` attempts to cancel a pending operation in the context `ctx`. This can be called by any thread at any time after starting an operation on the context, but will not take effect immediately. The actual cancellation happens at the next time GPGME processes I/O in that context.

The function returns an error code if the cancellation failed (in this case the state of `ctx` is not modified).
Appendix A The GnuPG UI Server Protocol

This section specifies the protocol used between clients and a User Interface Server (UI server). This protocol helps to build a system where all cryptographic operations are done by a server and the server is responsible for all dialogs. Although GPGME has no direct support for this protocol it is believed that servers will utilize the GPGME library; thus having the specification included in this manual is an appropriate choice. This protocol should be referenced as ‘The GnuPG UI Server Protocol’.

A server needs to implement these commands:¹

A.1 UI Server: Encrypt a Message

Before encryption can be done the recipients must be set using the command:

RECIPIENT string
[Command]
Set the recipient for the encryption. string is an RFC-2822 recipient name ("mailbox" as per section 3.4). This command may or may not check the recipient for validity right away; if it does not all recipients are expected to be checked at the time of the ENCRYPT command. All RECIPIENT commands are cumulative until a successful ENCRYPT command or until a RESET command. Linefeeds are obviously not allowed in string and should be folded into spaces (which are equivalent).

To tell the server the source and destination of the data, the next two commands are to be used:

INPUT FD=n
[Command]
Set the file descriptor for the message to be encrypted to n. The message send to the server is binary encoded.

GpgOL is a Windows only program, thus n is not a libc file descriptor but a regular system handle. Given that the Assuan connection works over a socket, it is not possible to use regular inheritance to make the file descriptor available to the server. Thus DuplicateHandle needs to be used to duplicate a handle to the server process. This is the reason that the server needs to implement the GETINFO pid command. Sending this command a second time replaces the file descriptor set by the last one.

OUTPUT FD=n [--binary]
[Command]
Set the file descriptor to be used for the output (i.e. the encrypted message) to n. If the option --binary is given the output shall be in binary format; if not given, the output for OpenPGP needs to be ASCII armored and for CMS Base-64 encoded. For details on the file descriptor, see the INPUT command.

The setting of the recipients, the data source and destination may happen in any order, even intermixed. If this has been done the actual encryption operation is called using:

¹ In all examples we assume that the connection has already been established; see the Assuan manual for details.
Appendix A: The GnuPG UI Server Protocol

**ENCRYPT --protocol=name**

[Command]

This command reads the plaintext from the file descriptor set by the INPUT command, encrypts it and writes the ciphertext to the file descriptor set by the OUTPUT command. The server may (and should) overlap reading and writing. The recipients used for the encryption are all the recipients set so far. If any recipient is not usable the server should take appropriate measures to notify the user about the problem and may cancel the operation by returning an error code. The used file descriptors are void after this command; the recipient list is only cleared if the server returns success.

Because GpgOL uses a streaming mode of operation the server is not allowed to auto select the protocol and must obey to the mandatory protocol parameter:

- **OpenPGP** Use the OpenPGP protocol (RFC-2440).
- **CMS** Use the CMS (PKCS#7) protocol (RFC-3852).

To support automagically selection of the protocol depending on the selected keys, the server MAY implement the command:

**PREP_ENCRYPT [--protocol=name] [--expect-sign]**

[Command]

This command considers all recipients set so far and decides whether it is able to take input and start the actual encryption. This is kind of a dry-run ENCRYPT without requiring or using the input and output file descriptors. The server shall cache the result of any user selection to avoid asking this again when the actual ENCRYPT command is send. The ‘--protocol’ option is optional; if it is not given, the server should allow the user to select the protocol to be used based on the recipients given or by any other means.

If ‘--expect-sign’ is given the server should expect that the message will also be signed and use this hint to present a unified recipient and signer selection dialog if possible and desired. A selected signer should then be cached for the expected SIGN command (which is expected in the same session but possible on another connection).

If this command is given again before a successful ENCRYPT command, the second one takes effect.

Before sending the OK response the server shall tell the client the protocol to be used (either the one given by the argument or the one selected by the user) by means of a status line:

**PROTOCOL name**

[Status line]

Advise the client to use the protocol name for the ENCRYPT command. The valid protocol names are listed under the description of the ENCRYPT command. The server shall emit exactly one PROTOCOL status line.

Here is an example of a complete encryption sequence; client lines are indicated by a c:, server responses by C::

```plaintext
ENCRYPT --protocol=name
C: OK
PROTOCOL name
```
A.2 UI Server: Sign a Message

The server needs to implement opaque signing as well as detached signing. Due to the nature of OpenPGP messages it is always required to send the entire message to the server; sending just the hash is not possible. The following two commands are required to set the input and output file descriptors:

**INPUT FD=n**  
Set the file descriptor for the message to be signed to \( n \). The message sent to the server is binary encoded. For details on the file descriptor, see the description of **INPUT** in the **ENCRYPT** section.

**OUTPUT FD=n [-binary]**  
Set the file descriptor to be used for the output. The output is either the complete signed message or in case of a detached signature just that detached signature. If the option --binary is given the output shall be in binary format; if not given, the output for OpenPGP needs to be ASCII armored and for CMS Base-64 encoded. For details on the file descriptor, see the **INPUT** command.

To allow the server the selection of a non-default signing key the client may optionally use the **SENDER** command, see [command SENDER], page 121.

The signing operation is then initiated by:
SIGN --protocol=name [--detached]  
Sign the data set with the INPUT command and write it to the sink set by OUTPUT.  
name is the signing protocol used for the message. For a description of the allowed  
protocols see the ENCRYPT command. With option --detached given, a detached  
signature is created; this is actually the usual way the command is used.

The client expects the server to send at least this status information before the final OK  
response:

MICALG string  
The string represents the hash algorithm used to create the signature. It is used  
with RFC-1847 style signature messages and defined by PGP/MIME (RFC-3156)  
and S/MIME (RFC-3851). The GPGME library has a supporting function gpgme_  
hash_algo_name to return the algorithm name as a string. This string needs to be  
lowercased and for OpenPGP prefixed with "pgp-".

A.3 UI Server: Decrypt a Message

Decryption may include the verification of OpenPGP messages. This is due to the often  
used combined signing/encryption modus of OpenPGP. The client may pass an option to  
the server to inhibit the signature verification. The following two commands are required  
to set the input and output file descriptors:

INPUT FD=n  
Set the file descriptor for the message to be decrypted to n. The message send to the  
server is either binary encoded or — in the case of OpenPGP — ASCII armored. For  
details on the file descriptor, see the description of INPUT in the ENCRYPT section.

OUTPUT FD=n  
Set the file descriptor to be used for the output. The output is binary encoded. For  
details on the file descriptor, see the description of INPUT in the ENCRYPT section.

The decryption is started with the command:

DECRIPT --protocol=name [--no-verify] [--export-session-key]  
name is the encryption protocol used for the message. For a description of the allowed  
protocols see the ENCRYPT command. This argument is mandatory. If the option  
'--no-verify' is given, the server should not try to verify a signature, in case the  
input data is an OpenPGP combined message. If the option '--export-session-key'  
is given and the underlying engine knows how to export the session key, it will appear  
on a status line

A.4 UI Server: Verify a Message

The server needs to support the verification of opaque signatures as well as detached signa-
tures. The kind of input sources controls what kind message is to be verified.

MESSAGE FD=n  
This command is used with detached signatures to set the file descriptor for the  
signed data to n. The data is binary encoded (used verbatim). For details on the file  
descriptor, see the description of INPUT in the ENCRYPT section.
Set the file descriptor for the opaque message or the signature part of a detached signature to \textit{n}. The message send to the server is either binary encoded or – in the case of OpenPGP – ASCII armored. For details on the file descriptor, see the description of \texttt{INPUT} in the \texttt{ENCRYPT} section.

\textbf{OUTPUT \texttt{FD=\textit{n}}}

Set the file descriptor to be used for the output. The output is binary encoded and only used for opaque signatures. For details on the file descriptor, see the description of \texttt{INPUT} in the \texttt{ENCRYPT} section.

The verification is then started using:

\textbf{VERIFY \texttt{--protocol=\textit{name}} \texttt{[\texttt{--silent}]} [Command]}

\textit{name} is the signing protocol used for the message. For a description of the allowed protocols see the \texttt{ENCRYPT} command. This argument is mandatory. Depending on the combination of \texttt{MESSAGE} \texttt{INPUT} and \texttt{OUTPUT} commands, the server needs to select the appropriate verification mode:

\begin{itemize}
  \item \texttt{MESSAGE} and \texttt{INPUT} This indicates a detached signature. Output data is not applicable.
  \item \texttt{INPUT} This indicates an opaque signature. As no output command has been given, the server is only required to check the signature.
  \item \texttt{INPUT} and \texttt{OUTPUT} This indicates an opaque signature. The server shall write the signed data to the file descriptor set by the output command. This data shall even be written if the signatures can’t be verified.
\end{itemize}

With \texttt{\textasciitilde--silent} the server shall not display any dialog; this is for example used by the client to get the content of opaque signed messages. The client expects the server to send at least this status information before the final OK response:

\textbf{SIGSTATUS \textit{flag} \textit{displaystring} [Status line]}

Returns the status for the signature and a short string explaining the status. Valid values for \textit{flag} are:

\begin{itemize}
  \item \texttt{none} The message has a signature but it could not not be verified due to a missing key.
  \item \texttt{green} The signature is fully valid.
  \item \texttt{yellow} The signature is valid but additional information was shown regarding the validity of the key.
  \item \texttt{red} The signature is not valid.
\end{itemize}

\textit{displaystring} is a percent-and-plus-encoded string with a short human readable description of the status. For example

\begin{verbatim}
S SIGSTATUS green Good+signature+from+Keith+Moon+<keith@example.net>
\end{verbatim}

Note that this string needs to fit into an Assuan line and should be short enough to be displayed as short one-liner on the clients window. As usual the encoding of this string is UTF-8 and it should be send in its translated form.

The server shall send one status line for every signature found on the message.
A.5 UI Server: Specifying the input files to operate on.

All file related UI server commands operate on a number of input files or directories, specified by one or more FILE commands:

FILE [-clear] name

Add the file or directory name to the list of pathnames to be processed by the server. The parameter name must be an absolute path name (including the drive letter) and is percent espaced (in particular, the characters %, = and white space characters are always escaped). If the option --clear is given, the list of files is cleared before adding name.

Historical note: The original spec did not define --clear but the keyword --continued after the file name to indicate that more files are to be expected. However, this has never been used and thus removed from the specs.

A.6 UI Server: Encrypting and signing files.

First, the input files need to be specified by one or more FILE commands. Afterwards, the actual operation is requested:

CRYPT_FILES --nohup

SIGN_FILES --nohup

CRYPT_SIGN_FILES --nohup

Request that the files specified by FILE are encrypted and/or signed. The command selects the default action. The UI server may allow the user to change this default afterwards interactively, and even abort the operation or complete it only on some of the selected files and directories.

What it means to encrypt or sign a file or directory is specific to the preferences of the user, the functionality the UI server provides, and the selected protocol. Typically, for each input file a new file is created under the original filename plus a protocol specific extension (like .gpg or .sig), which contain the encrypted/signed file or a detached signature. For directories, the server may offer multiple options to the user (for example ignore or process recursively).

The CRYPT_SIGN_FILES command requests a combined sign and encrypt operation. It may not be available for all protocols (for example, it is available for OpenPGP but not for CMS).

The option --nohup is mandatory. It is currently unspecified what should happen if --nohup is not present. Because --nohup is present, the server always returns OK promptly, and completes the operation asynchronously.

A.7 UI Server: Decrypting and verifying files.

First, the input files need to be specified by one or more FILE commands. Afterwards, the actual operation is requested:

CRYPT_FILES --nohup

VERIFY_FILES --nohup
Appendix A: The GnuPG UI Server Protocol

**DECRYPT_VERIFY_FILES** –nohup

[Command]

Request that the files specified by FILE are decrypted and/or verified. The command selects the default action. The UI server may allow the user to change this default afterwards interactively, and even abort the operation or complete it only on some of the selected files and directories.

What it means to decrypt or verify a file or directory is specific to the preferences of the user, the functionality the UI server provides, and the selected protocol. Typically, for decryption, a new file is created for each input file under the original filename minus a protocol specific extension (like .gpg) which contains the original plaintext. For verification a status is displayed for each signed input file, indicating if it is signed, and if yes, if the signature is valid. For files that are signed and encrypted, the VERIFY command transiently decrypts the file to verify the enclosed signature. For directories, the server may offer multiple options to the user (for example ignore or process recursively).

The option --nohup is mandatory. It is currently unspecified what should happen if --nohup is not present. Because --nohup is present, the server always returns OK promptly, and completes the operation asynchronously.

**A.8 UI Server: Managing certificates.**

First, the input files need to be specified by one or more FILE commands. Afterwards, the actual operation is requested:

**IMPORT_FILES** –nohup

[Command]

Request that the certificates contained in the files specified by FILE are imported into the local certificate databases.

For directories, the server may offer multiple options to the user (for example ignore or process recursively).

The option --nohup is mandatory. It is currently unspecified what should happen if --nohup is not present. Because --nohup is present, the server always returns OK promptly, and completes the operation asynchronously.

FIXME: It may be nice to support an EXPORT command as well, which is enabled by the context menu of the background of a directory.

**A.9 UI Server: Create and verify checksums for files.**

First, the input files need to be specified by one or more FILE commands. Afterwards, the actual operation is requested:

**CHECKSUM_CREATE_FILES** –nohup

[Command]

Request that checksums are created for the files specified by FILE. The choice of checksum algorithm and the destination storage and format for the created checksums depend on the preferences of the user and the functionality provided by the UI server. For directories, the server may offer multiple options to the user (for example ignore or process recursively).

The option --nohup is mandatory. It is currently unspecified what should happen if --nohup is not present. Because --nohup is present, the server always returns OK promptly, and completes the operation asynchronously.
CHECKSUM_VERIFY_FILES –nohup

Request that checksums are created for the files specified by FILE and verified against previously created and stored checksums. The choice of checksum algorithm and the source storage and format for previously created checksums depend on the preferences of the user and the functionality provided by the UI server. For directories, the server may offer multiple options to the user (for example ignore or process recursively).

If the source storage of previously created checksums is available to the user through the Windows shell, this command may also accept such checksum files as FILE arguments. In this case, the UI server should instead verify the checksum of the referenced files as if they were given as INPUT files.

The option --nohup is mandatory. It is currently unspecified what should happen if --nohup is not present. Because --nohup is present, the server always returns OK promptly, and completes the operation asynchronously.

A.10 Miscellaneous UI Server Commands

The server needs to implement the following commands which are not related to a specific command:

GETINFO what

This is a multi purpose command, commonly used to return a variety of information. The required subcommands as described by the what parameter are:

pid Return the process id of the server in decimal notation using an Assuan data line.

To allow the server to pop up the windows in the correct relation to the client, the client is advised to tell the server by sending the option:

window-id number

The number represents the native window ID of the clients current window. On Windows systems this is a windows handle (HWND) and on X11 systems it is the X Window ID. The number needs to be given as a hexadecimal value so that it is easier to convey pointer values (e.g. HWND).

A client may want to fire up the certificate manager of the server. To do this it uses the Assuan command:

START_KEYMANAGER

The server shall pop up the main window of the key manager (aka certificate manager). The client expects that the key manager is brought into the foreground and that this command immediately returns (does not wait until the key manager has been fully brought up).

A client may want to fire up the configuration dialog of the server. To do this it uses the Assuan command:

START_CONF DIALOG

The server shall pop up its configuration dialog. The client expects that this dialog is brought into the foreground and that this command immediately returns (i.e. it does not wait until the dialog has been fully brought up).
When doing an operation on a mail, it is useful to let the server know the address of the sender:

**SENDER** [--info] [--protocol=name] email

email is the plain ASCII encoded address ("addr-spec" as per RFC-2822) enclosed in angle brackets. The address set with this command is valid until a successful completion of the operation or until a **RESET** command. A second command overrides the effect of the first one; if email is not given and ‘--info’ is not used, the server shall use the default signing key.

If option ‘--info’ is not given, the server shall also suggest a protocol to use for signing. The client may use this suggested protocol on its own discretion. The same status line as with PREP,ENCRYPT is used for this.

The option ‘--protocol’ may be used to give the server a hint on which signing protocol should be preferred.

To allow the UI-server to visually identify a running operation or to associate operations the server MAY support the command:

**SESSION** number [string]

The number is an arbitrary value, a server may use to associate simultaneous running sessions. It is a 32 bit unsigned integer with 0 as a special value indicating that no session association shall be done.

If string is given, the server may use this as the title of a window or, in the case of an email operation, to extract the sender’s address. The string may contain spaces; thus no plus-escaping is used.

This command may be used at any time and overrides the effect of the last command. A **RESET** undoes the effect of this command.
Appendix B How to solve problems

Everyone knows that software often does not do what it should do and thus there is a need to track down problems. This is in particular true for applications using a complex library like GPGME and of course also for the library itself. Here we give a few hints on how to solve such problems.

First of all you should make sure that the keys you want to use are installed in the GnuPG engine and are usable. Thus the first test is to run the desired operation using gpg or gpgsm on the command line. If you can’t figure out why things don’t work, you may use GPGME’s built in trace feature. This feature is either enabled using the environment variable GPGME_DEBUG or, if this is not possible, by calling the function gpgme_set_global_flag. The value is the trace level and an optional file name. If no file name is given the trace output is printed to stderr.

For example

```
GPGME_DEBUG=9:/home/user/mygpgme.log
```

(Note that under Windows you use a semicolon in place of the colon to separate the fields.)

A trace level of 9 is pretty verbose and thus you may want to start off with a lower level. The exact definition of the trace levels and the output format may change with any release; you need to check the source code for details. In any case the trace log should be helpful to understand what is going going on. Warning: The trace log may reveal sensitive details like passphrases or other data you use in your application. If you are asked to send a log file, make sure that you run your tests only with play data.

The trace function makes use of gpgrt’s logging function and thus the special ‘socket://’ and ‘tcp://’ files may be used. Because this conflicts with the use of colons to separate fields, the following hack is used: If the file name contains the string ‘//’ all carets are replaced by colons. For example to log to TCP port 42042 this can be used:

```
GPGME_DEBUG=5:tcp^//127.0.0.1^42042
```
Appendix C Deprecated Functions

For backward compatibility GPGME has a number of functions, data types and constants which are deprecated and should not be used anymore. We document here those which are really old to help understanding old code and to allow migration to their modern counterparts.

**Warning:** These interfaces will be removed in a future version of GPGME.

### void gpgme_key_release (gpgme_key_t key)

The function `gpgme_key_release` is equivalent to `gpgme_key_unref`.

### gpgme_error_t gpgme_op_import_ext (gpgme_ctx_t ctx, gpgme_data_t keydata, int *nr)

**Since:** 0.3.9

The function `gpgme_op_import_ext` is equivalent to:

```c
  gpgme_error_t err = gpgme_op_import (ctx, keydata);
  if (!err)
    {
      gpgme_import_result_t result = gpgme_op_import_result (ctx);
      *nr = result->considered;
    }
```

### Data type

```c
  gpgme_error_t (*gpgme_edit_cb_t) (void *handle, gpgme_status_code_t status, const char *args, int fd)
```

The `gpgme_edit_cb_t` type is the type of functions which GPGME calls if it a key edit operation is on-going. The status code `status` and the argument line `args` are passed through by GPGME from the crypto engine. The file descriptor `fd` is -1 for normal status messages. If `status` indicates a command rather than a status message, the response to the command should be written to `fd`. The `handle` is provided by the user at start of operation.

The function should return `GPG_ERR_FALSE` if it did not handle the status code, 0 for success, or any other error value.

### gpgme_error_t gpgme_op_edit (gpgme_ctx_t ctx, gpgme_key_t key, gpgme_edit_cb_t fnc, void *handle, gpgme_data_t out)

**Since:** 0.3.9

Note: This function is deprecated, please use `gpgme_op_interact` instead.

The function `gpgme_op_edit` processes the key `KEY` interactively, using the edit callback function `FNC` with the handle `HANDLE`. The callback is invoked for every status and command request from the crypto engine. The output of the crypto engine is written to the data object `out`.

Note that the protocol between the callback function and the crypto engine is specific to the crypto engine and no further support in implementing this protocol correctly is provided by GPGME.

The function returns the error code `GPG_ERR_NO_ERROR` if the edit operation completes successfully, `GPG_ERR_INV_VALUE` if `ctx` or `key` is not a valid pointer, and any error returned by the crypto engine or the edit callback handler.
Appendix C: Deprecated Functions

Function gpgme_error_t gpgme_op_edit_start (gpgme_ctx_t ctx, gpgme_key_t key, gpgme_edit_cb_t fnc, void *handle, gpgme_data_t out) [Function] Since: 0.3.9
Note: This function is deprecated, please use gpgme_op_interact_start instead.
The function gpgme_op_edit_start initiates a gpgme_op_edit operation. It can be completed by calling gpgme_wait on the context. See Section 7.8.1 [Waiting For Completion], page 100.
The function returns the error code GPG_ERR_NO_ERROR if the operation was started successfully, and GPG_ERR_INV_VALUE if ctx or key is not a valid pointer.

Function gpgme_error_t gpgme_op_card_edit (gpgme_ctx_t ctx, gpgme_key_t key, gpgme_edit_cb_t fnc, void *handle, gpgme_data_t out) [Function] Note: This function is deprecated, please use gpgme_op_interact with the flag GPGME_INTERACT_CARD instead.
The function gpgme_op_card_edit is analogous to gpgme_op_edit, but should be used to process the smart card corresponding to the key key.

Function gpgme_error_t gpgme_op_card_edit_start (gpgme_ctx_t ctx, gpgme_key_t key, gpgme_edit_cb_t fnc, void *handle, gpgme_data_t out) [Function] Note: This function is deprecated, please use gpgme_op_interact_start with the flag GPGME_INTERACT_CARD instead.
The function gpgme_op_card_edit_start initiates a gpgme_op_card_edit operation. It can be completed by calling gpgme_wait on the context. See Section 7.8.1 [Waiting For Completion], page 100.
The function returns the error code GPG_ERR_NO_ERROR if the operation was started successfully, and GPG_ERR_INV_VALUE if ctx or key is not a valid pointer.

Function gpgme_error_t gpgme_data_new_with_read_cb (gpgme_data_t *dh, int (*readfunc)(void *hook, char *buffer, size_t count, size_t *nread), void *hook_value) [Function] The function gpgme_data_new_with_read_cb creates a new gpgme_data_t object and uses the callback function readfunc to retrieve the data on demand. As the callback function can supply the data in any way it wants, this is the most flexible data type GPGME provides. However, it can not be used to write data.
The callback function receives hook_value as its first argument whenever it is invoked. It should return up to count bytes in buffer, and return the number of bytes actually read in nread. It may return 0 in nread if no data is currently available. To indicate EOF the function should return with an error code of -1 and set nread to 0. The callback function may support to reset its internal read pointer if it is invoked with buffer and nread being NULL and count being 0.
The function returns the error code GPG_ERR_NO_ERROR if the data object was successfully created, GPG_ERR_INV_VALUE if dh or readfunc is not a valid pointer, and GPG_ERR_ENOMEM if not enough memory is available.

Function gpgme_error_t gpgme_data_rewind (gpgme_data_t dh) [Function] The function gpgme_data_rewind is equivalent to:
return (gpgme_data_seek (dh, 0, SEEK_SET) == -1) ? gpgme_error_from_errno (errno) : 0;

**gpgme_attr_t**

The `gpgme_attr_t` type is used to specify a key or trust item attribute. The following attributes are defined:

- **GPGME_ATTR_KEYID**
  
  This is the key ID of a sub key. It is representable as a string.

- **GPGME_ATTR_FPR**
  
  This is the fingerprint of a sub key. It is representable as a string.

- **GPGME_ATTR_ALGO**
  
  This is the crypto algorithm for which the sub key can be used. It is representable as a string and as a number. The numbers correspond to the `enum gcry_pk_algos` values in the gcrypt library.

- **GPGME_ATTR_LEN**
  
  This is the key length of a sub key. It is representable as a number.

- **GPGME_ATTR_CREATED**
  
  This is the timestamp at creation time of a sub key. It is representable as a number.

- **GPGME_ATTR_EXPIRE**
  
  This is the expiration time of a sub key. It is representable as a number.

- **GPGME_ATTR_USERID**
  
  This is a user ID. There can be more than one user IDs in a `gpgme_key_t` object. The first one (with index 0) is the primary user ID. The user ID is representable as a number.

- **GPGME_ATTR_NAME**
  
  This is the name belonging to a user ID. It is representable as a string.

- **GPGME_ATTR_EMAIL**
  
  This is the email address belonging to a user ID. It is representable as a string.

- **GPGME_ATTR_COMMENT**
  
  This is the comment belonging to a user ID. It is representable as a string.

- **GPGME_ATTR_VALIDITY**
  
  This is the validity belonging to a user ID. It is representable as a string and as a number. See below for a list of available validities.

- **GPGME_ATTR_UID_REVOKED**
  
  This specifies if a user ID is revoked. It is representable as a number, and is 1 if the user ID is revoked, and 0 otherwise.

- **GPGME_ATTR_UID_INVALID**
  
  This specifies if a user ID is invalid. It is representable as a number, and is 1 if the user ID is invalid, and 0 otherwise.
GPGME_ATTR_TYPE
This returns information about the type of key. For the string function this will either be "PGP" or "X.509". The integer function returns 0 for PGP and 1 for X.509.

GPGME_ATTR_IS_SECRET
This specifies if the key is a secret key. It is representable as a number, and is 1 if the key is revoked, and 0 otherwise.

GPGME_ATTR_KEY_REVOKED
This specifies if a sub key is revoked. It is representable as a number, and is 1 if the key is revoked, and 0 otherwise.

GPGME_ATTR_KEY_INVALID
This specifies if a sub key is invalid. It is representable as a number, and is 1 if the key is invalid, and 0 otherwise.

GPGME_ATTR_KEY_EXPIRED
This specifies if a sub key is expired. It is representable as a number, and is 1 if the key is expired, and 0 otherwise.

GPGME_ATTR_KEY_DISABLED
This specifies if a sub key is disabled. It is representable as a number, and is 1 if the key is disabled, and 0 otherwise.

GPGME_ATTR_KEY_CAPS
This is a description of the capabilities of a sub key. It is representable as a string. The string contains the letter “e” if the key can be used for encryption, “s” if the key can be used for signatures, and “c” if the key can be used for certifications.

GPGME_ATTR_CAN_ENCRYPT
This specifies if a sub key can be used for encryption. It is representable as a number, and is 1 if the sub key can be used for encryption, and 0 otherwise.

GPGME_ATTR_CAN_SIGN
This specifies if a sub key can be used to create data signatures. It is representable as a number, and is 1 if the sub key can be used for signatures, and 0 otherwise.

GPGME_ATTR_CAN_CERTIFY
This specifies if a sub key can be used to create key certificates. It is representable as a number, and is 1 if the sub key can be used for certifications, and 0 otherwise.

GPGME_ATTR_SERIAL
The X.509 issuer serial attribute of the key. It is representable as a string.

GPGME_ATTR_ISSUE
The X.509 issuer name attribute of the key. It is representable as a string.

GPGME_ATTR_CHAINID
The X.509 chain ID can be used to build the certification chain. It is representable as a string.
const char * gpgme_key_get_string_attr (gpgme_key_t key,       [Function]
gpgme_attr_t what, const void *reserved, int idx)
The function gpgme_key_get_string_attr returns the value of the string-
representable attribute what of key key. If the attribute is an attribute of a sub key
or an user ID, idx specifies the sub key or user ID of which the attribute value is
returned. The argument reserved is reserved for later use and should be NULL.
The string returned is only valid as long as the key is valid.
The function returns 0 if an attribute can’t be returned as a string, key is not a valid
pointer, idx out of range, or reserved not NULL.

unsigned long gpgme_key_get_ulong_attr (gpgme_key_t key,      [Function]
gpgme_attr_t what, const void *reserved, int idx)
The function gpgme_key_get_ulong_attr returns the value of the number-
representable attribute what of key key. If the attribute is an attribute of a sub key
or an user ID, idx specifies the sub key or user ID of which the attribute value is
returned. The argument reserved is reserved for later use and should be NULL.
The function returns 0 if the attribute can’t be returned as a number, key is not a valid
pointer, idx out of range, or reserved not NULL.

The signatures on a key are only available if the key was retrieved via a listing operation
with the GPGME_KEYLIST_MODE_SIGS mode enabled, because it is expensive to retrieve all
signatures of a key.

So, before using the below interfaces to retrieve the signatures on a key, you have to
make sure that the key was listed with signatures enabled. One convenient, but blocking,
way to do this is to use the function gpgme_get_key.

gpgme_attr_t

The gpgme_attr_t type is used to specify a key signature attribute. The following
attributes are defined:

GPGME_ATTR_KEYID
This is the key ID of the key which was used for the signature. It is
representable as a string.

GPGME_ATTR_ALGO
This is the crypto algorithm used to create the signature. It is repre-
sentable as a string and as a number. The numbers correspond to the
enum gcry_pk_algos values in the gcrypt library.

GPGME_ATTR_CREATED
This is the timestamp at creation time of the signature. It is representable as a number.

GPGME_ATTR_EXPIRE
This is the expiration time of the signature. It is representable as a number.

GPGME_ATTR_USERID
This is the user ID associated with the signing key. The user ID is re-
presentable as a number.
GPGME_ATTR_NAME
This is the name belonging to a user ID. It is representable as a string.

GPGME_ATTR_EMAIL
This is the email address belonging to a user ID. It is representable as a string.

GPGME_ATTR_COMMENT
This is the comment belonging to a user ID. It is representable as a string.

GPGME_ATTR_KEY_REVOKED
This specifies if a key signature is a revocation signature. It is representable as a number, and is 1 if the key is revoked, and 0 otherwise.

GPGME_ATTR_SIG_CLASS
This specifies the signature class of a key signature. It is representable as a number. The meaning is specific to the crypto engine.

GPGME_ATTR_SIG_CLASS
This specifies the signature class of a key signature. It is representable as a number. The meaning is specific to the crypto engine.

GPGME_ATTR_SIG_STATUS
This is the same value as returned by gpgme_get_sig_status.

const char * gpgme_key_sig_get_string_attr (gpgme_key_t key, int uid_idx, gpgme_attr_t what, const void *reserved, int idx)  
The function gpgme_key_sig_get_string_attr returns the value of the string-representable attribute what of the signature idx on the user ID uid_idx in the key key. The argument reserved is reserved for later use and should be NULL.

The string returned is only valid as long as the key is valid.

The function returns 0 if an attribute can’t be returned as a string, key is not a valid pointer, uid_idx or idx out of range, or reserved not NULL.

unsigned long gpgme_key_sig_get_ulong_attr (gpgme_key_t key, int uid_idx, gpgme_attr_t what, const void *reserved, int idx)  
The function gpgme_key_sig_get_ulong_attr returns the value of the number-representable attribute what of the signature idx on the user ID uid_idx in the key key. The argument reserved is reserved for later use and should be NULL.

The function returns 0 if an attribute can’t be returned as a string, key is not a valid pointer, uid_idx or idx out of range, or reserved not NULL.

enum gpgme_sig_stat_t  
The gpgme_sig_stat_t type holds the result of a signature check, or the combined result of all signatures. The following results are possible:

GPGME_SIG_STAT_NONE
This status should not occur in normal operation.

GPGME_SIG_STAT_GOOD
This status indicates that the signature is valid. For the combined result this status means that all signatures are valid.
GPGME_SIG_STAT_GOOD_EXP
This status indicates that the signature is valid but expired. For the combined result this status means that all signatures are valid and expired.

GPGME_SIG_STAT_GOOD_EXPKEY
This status indicates that the signature is valid but the key used to verify the signature has expired. For the combined result this status means that all signatures are valid and all keys are expired.

GPGME_SIG_STAT_BAD
This status indicates that the signature is invalid. For the combined result this status means that all signatures are invalid.

GPGME_SIG_STAT_NOKEY
This status indicates that the signature could not be verified due to a missing key. For the combined result this status means that all signatures could not be checked due to missing keys.

GPGME_SIG_STAT_NOSIG
This status indicates that the signature data provided was not a real signature.

GPGME_SIG_STAT_ERROR
This status indicates that there was some other error which prevented the signature verification.

GPGME_SIG_STAT_DIFF
For the combined result this status means that at least two signatures have a different status. You can get each key’s status with gpgme_get_sig_status.

const char * gpgme_get_sig_status (gpgme_ctx_t ctx, int idx, gpgme_sig_stat_t *r_stat, time_t *r_created) [Function]

The function gpgme_get_sig_status is equivalent to:

gpgme_verify_result_t result;
gpgme_signature_t sig;

result = gpgme_op_verify_result (ctx);
sig = result->signatures;

while (sig && idx)
{
    sig = sig->next;
    idx--;
}
if (!sig || idx)
    return NULL;

if (r_stat)
{

switch (gpg_err_code (sig->status))
{
  case GPG_ERR_NO_ERROR:
    *r_stat = GPGME_SIG_STAT_GOOD;
    break;
  
  case GPG_ERR_BAD_SIGNATURE:
    *r_stat = GPGME_SIG_STAT_BAD;
    break;
  
  case GPG_ERR_NO_PUBKEY:
    *r_stat = GPGME_SIG_STAT_NOKEY;
    break;
  
  case GPG_ERR_NO_DATA:
    *r_stat = GPGME_SIG_STAT_NOSIG;
    break;
  
  case GPG_ERR_SIG_EXPIRED:
    *r_stat = GPGME_SIG_STAT_GOOD_EXP;
    break;
  
  case GPG_ERR_KEY_EXPIRED:
    *r_stat = GPGME_SIG_STAT_GOOD_EXPKEY;
    break;
  
  default:
    *r_stat = GPGME_SIG_STAT_ERROR;
    break;
}

if (r_created)
  *r_created = sig->timestamp;
return sig->fpr;

const char * gpgme_get_sig_string_attr (gpgme_ctx_t ctx, int idx, gpgme_attr_t what, int whatidx) [Function]
The function gpgme_get_sig_string_attr is equivalent to:

  gpgme_verify_result_t result;
  gpgme_signature_t sig;

  result = gpgme_op_verify_result (ctx);
  sig = result->signatures;

  while (sig && idx)
  {
    sig = sig->next;
Appendix C: Deprecated Functions

idx--;
}
if (!sig || idx)
    return NULL;

switch (what)
{
    case GPGME_ATTR_FPR:
        return sig->fpr;

    case GPGME_ATTR_ERRTOK:
        if (whatidx == 1)
            return sig->wrong_key_usage ? "Wrong_Key_Usage" : "";
        else
            return "";
        default:
            break;
}

return NULL;

const char * gpgme_get_sig_ulong_attr (gpgme_ctx_t ctx, int idx, gpgme_attr_t what, int whatidx)
{
    gpgme_verify_result_t result;
    gpgme_signature_t sig;

    result = gpgme_op_verify_result (ctx);
    sig = result->signatures;

    while (sig && idx)
    {
        sig = sig->next;
        idx--;
    }
    if (!sig || idx)
        return 0;

    switch (what)
    {
        case GPGME_ATTR_CREATED:
            return sig->timestamp;

        case GPGME_ATTR_EXPIRE:
            return sig->exp_timestamp;

        case GPGME_ATTR_VALIDITY:
return (unsigned long) sig->validity;

case GPGME_ATTR_SIG_STATUS:
    switch (sig->status)
    {
    case GPG_ERR_NO_ERROR:
        return GPGME_SIG_STAT_GOOD;
    case GPG_ERR_BAD_SIGNATURE:
        return GPGME_SIG_STAT_BAD;
    case GPG_ERR_NO_PUBKEY:
        return GPGME_SIG_STAT_NOKEY;
    case GPG_ERR_NO_DATA:
        return GPGME_SIG_STAT_NOSIG;
    case GPG_ERR_SIG_EXPIRED:
        return GPGME_SIG_STAT_GOOD_EXP;
    case GPG_ERR_KEY_EXPIRED:
        return GPGME_SIG_STAT_GOOD_EXPKEY;
    default:
        return GPGME_SIG_STAT_ERROR;
    }

    case GPGME_ATTR_SIG_SUMMARY:
        return sig->summary;

    default:
        break;
    }
return 0;

const char * gpgme_get_sig_key (gpgme_ctx_t ctx, int idx, gpgme_key_t *r_key)
{
    The function gpgme_get_sig_key is equivalent to:
    gpgme_verify_result_t result;
    gpgme_signature_t sig;

    result = gpgme_op_verify_result (ctx);
    sig = result->signatures;

    while (sig && idx)
    {
        sig = sig->next;
    }
    return sig;
}
idx--;  
}  
if (!sig || idx)  
return gpg_error (GPG_ERR_EOF);

return gpgme_get_key (ctx, sig->fpr, r_key, 0);
GNU Lesser General Public License

Version 2.1, February 1999
Copyright © 1991, 1999 Free Software Foundation, Inc.
59 Temple Place – Suite 330, Boston, MA 02111-1307, USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

[This is the first released version of the Lesser GPL. It also counts as the successor of the GNU Library Public License, version 2, hence the version number 2.1.]

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public Licenses are intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users.

This license, the Lesser General Public License, applies to some specially designated software—typically libraries—of the Free Software Foundation and other authors who decide to use it. You can use it too, but we suggest you first think carefully about whether this license or the ordinary General Public License is the better strategy to use in any particular case, based on the explanations below.

When we speak of free software, we are referring to freedom of use, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish); that you receive source code or can get it if you want it; that you can change the software and use pieces of it in new free programs; and that you are informed that you can do these things.

To protect your rights, we need to make restrictions that forbid distributors to deny you these rights or to ask you to surrender these rights. These restrictions translate to certain responsibilities for you if you distribute copies of the library or if you modify it.

For example, if you distribute copies of the library, whether gratis or for a fee, you must give the recipients all the rights that we gave you. You must make sure that they, too, receive or can get the source code. If you link other code with the library, you must provide complete object files to the recipients, so that they can relink them with the library after making changes to the library and recompiling it. And you must show them these terms so they know their rights.

We protect your rights with a two-step method: (1) we copyright the library, and (2) we offer you this license, which gives you legal permission to copy, distribute and/or modify the library.

To protect each distributor, we want to make it very clear that there is no warranty for the free library. Also, if the library is modified by someone else and passed on, the recipients should know that what they have is not the original version, so that the original author’s reputation will not be affected by problems that might be introduced by others.

Finally, software patents pose a constant threat to the existence of any free program. We wish to make sure that a company cannot effectively restrict the users of a free program
by obtaining a restrictive license from a patent holder. Therefore, we insist that any patent license obtained for a version of the library must be consistent with the full freedom of use specified in this license.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License. This license, the GNU Lesser General Public License, applies to certain designated libraries, and is quite different from the ordinary General Public License. We use this license for certain libraries in order to permit linking those libraries into non-free programs.

When a program is linked with a library, whether statically or using a shared library, the combination of the two is legally speaking a combined work, a derivative of the original library. The ordinary General Public License therefore permits such linking only if the entire combination fits its criteria of freedom. The Lesser General Public License permits more lax criteria for linking other code with the library.

We call this license the Lesser General Public License because it does Less to protect the user’s freedom than the ordinary General Public License. It also provides other free software developers Less of an advantage over competing non-free programs. These disadvantages are the reason we use the ordinary General Public License for many libraries. However, the Lesser license provides advantages in certain special circumstances.

For example, on rare occasions, there may be a special need to encourage the widest possible use of a certain library, so that it becomes a de-facto standard. To achieve this, non-free programs must be allowed to use the library. A more frequent case is that a free library does the same job as widely used non-free libraries. In this case, there is little to gain by limiting the free library to free software only, so we use the Lesser General Public License.

In other cases, permission to use a particular library in non-free programs enables a greater number of people to use a large body of free software. For example, permission to use the GNU C Library in non-free programs enables many more people to use the whole GNU operating system, as well as its variant, the GNU/Linux operating system.

Although the Lesser General Public License is less protective of the users’ freedom, it does ensure that the user of a program that is linked with the Library has the freedom and the wherewithal to run that program using a modified version of the Library.

The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a “work based on the library” and a “work that uses the library”. The former contains code derived from the library, whereas the latter must be combined with the library in order to run.

**TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION**

0. This License Agreement applies to any software library or other program which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Lesser General Public License (also called “this License”). Each licensee is addressed as “you”.

A “library” means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.
The “Library”, below, refers to any such software library or work which has been distributed under these terms. A “work based on the Library” means either the Library or any derivative work under copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term “modification”.)

“Source code” for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does.

1. You may copy and distribute verbatim copies of the Library’s complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the Library.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:
   a. The modified work must itself be a software library.
   b. You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change.
   c. You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License.
   d. If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful.

(For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply
to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Library.

In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.

Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.

This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

4. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange.

If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.

5. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a “work that uses the Library”. Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.

However, linking a “work that uses the Library” with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a “work that uses the library”. The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.

When a “work that uses the Library” uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.
If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself.

6. As an exception to the Sections above, you may also combine or link a “work that uses the Library” with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer’s own use and reverse engineering for debugging such modifications.

You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

a. Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable “work that uses the Library”, as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)

b. Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (1) uses at run time a copy of the library already present on the user’s computer system, rather than copying library functions into the executable, and (2) will operate properly with a modified version of the library, if the user installs one, as long as the modified version is interface-compatible with the version that the work was made with.

c. Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.

d. If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.

e. Verify that the user has already received a copy of these materials or that you have already sent this user a copy.

For an executable, the required form of the “work that uses the Library” must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the materials to be distributed need not include anything that is normally distributed (in either source or binary form) with the major components
(compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

7. You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:
   a. Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above.
   b. Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

8. You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

9. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.

10. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients’ exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties with this License.

11. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.
If any portion of this section is held invalid or unenforceable under any particular
circumstance, the balance of the section is intended to apply, and the section as a
whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other
property right claims or to contest validity of any such claims; this section has the
sole purpose of protecting the integrity of the free software distribution system which
is implemented by public license practices. Many people have made generous contri-
butions to the wide range of software distributed through that system in reliance on
consistent application of that system; it is up to the author/donor to decide if he or
she is willing to distribute software through any other system and a licensee cannot
impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence
of the rest of this License.

12. If the distribution and/or use of the Library is restricted in certain countries either
by patents or by copyrighted interfaces, the original copyright holder who places the
Library under this License may add an explicit geographical distribution limitation
excluding those countries, so that distribution is permitted only in or among countries
not thus excluded. In such case, this License incorporates the limitation as if written
in the body of this License.

13. The Free Software Foundation may publish revised and/or new versions of the Lesser
General Public License from time to time. Such new versions will be similar in spirit
to the present version, but may differ in detail to address new problems or concerns.
Each version is given a distinguishing version number. If the Library specifies a version
number of this License which applies to it and “any later version”, you have the option of
following the terms and conditions either of that version or of any later version published
by the Free Software Foundation. If the Library does not specify a license version
number, you may choose any version ever published by the Free Software Foundation.

14. If you wish to incorporate parts of the Library into other free programs whose distribu-
tion conditions are incompatible with these, write to the author to ask for permission.
For software which is copyrighted by the Free Software Foundation, write to the Free
Software Foundation; we sometimes make exceptions for this. Our decision will be
guided by the two goals of preserving the free status of all derivatives of our free soft-
ware and of promoting the sharing and reuse of software generally.

NO WARRANTY

15. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO
WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLI-
CABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPY-
RIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY “AS IS”
WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED,
INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MER-
CHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE
RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH
YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST
OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN
WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS
How to Apply These Terms to Your New Libraries

If you develop a new library, and you want it to be of the greatest possible use to the public, we recommend making it free software that everyone can redistribute and change. You can do so by permitting redistribution under these terms (or, alternatively, under the terms of the ordinary General Public License).

To apply these terms, attach the following notices to the library. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the “copyright” line and a pointer to where the full notice is found.

one line to give the library’s name and an idea of what it does.
Copyright (C) year name of author

This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this library; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307, USA.

Also add information on how to contact you by electronic and paper mail.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a “copyright disclaimer” for the library, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the library ‘Frob’ (a library for tweaking knobs) written by James Random Hacker.

signature of Ty Coon, 1 April 1990
Ty Coon, President of Vice

That’s all there is to it!
GNU General Public License

Version 3, 29 June 2007

Copyright © 2007 Free Software Foundation, Inc. https://fsf.org/

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program—to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers’ and authors’ protection, the GPL clearly explains that there is no warranty for this free software. For both users’ and authors’ sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users’ freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.
Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS

0. Definitions.

“This License” refers to version 3 of the GNU General Public License.

“Copyright” also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

“The Program” refers to any copyrightable work licensed under this License. Each licensee is addressed as “you”. “Licensees” and “recipients” may be individuals or organizations.

To “modify” a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a “modified version” of the earlier work or a work “based on” the earlier work.

A “covered work” means either the unmodified Program or a work based on the Program.

To “propagate” a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To “convey” a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays “Appropriate Legal Notices” to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.


The “source code” for a work means the preferred form of the work for making modifications to it. “Object code” means any non-source form of a work.

A “Standard Interface” means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.
The “System Libraries” of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A “Major Component”, in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The “Corresponding Source” for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work’s System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.


No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.
When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.


You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

a. The work must carry prominent notices stating that you modified it, and giving a relevant date.

b. The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to “keep intact all notices”.

c. You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.

d. If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an “aggregate” if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation’s users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:
a. Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.

b. Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

c. Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

d. Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

e. Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A “User Product” is either (1) a “consumer product”, which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, “normally used” refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

“Installation Information” for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source.
The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

“Additional permissions” are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

a. Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or

b. Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or

c. Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or
d. Limiting the use for publicity purposes of names of licensors or authors of the material; or

e. Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or

f. Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered “further restrictions” within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.


You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance.
However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An “entity transaction” is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party’s predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

A “contributor” is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor’s “contributor version”.

A contributor’s “essential patent claims” are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, “control” includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor’s essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a “patent license” is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To “grant” such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so
available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. “Knowingly relying” means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient’s use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is “discriminatory” if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.


If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.
The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License “or any later version” applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy’s public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.
END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the “copyright” line and a pointer to where the full notice is found.

one line to give the program’s name and a brief idea of what it does.
Copyright (C) year name of author

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see https://www.gnu.org/licenses/.

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

program Copyright (C) year name of author
This program comes with ABSOLUTELY NO WARRANTY; for details type ‘show w’. This is free software, and you are welcome to redistribute it under certain conditions; type ‘show c’ for details.

The hypothetical commands ‘show w’ and ‘show c’ should show the appropriate parts of the General Public License. Of course, your program’s commands might be different; for a GUI interface, you would use an “about box”.

You should also get your employer (if you work as a programmer) or school, if any, to sign a “copyright disclaimer” for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see https://www.gnu.org/licenses/.

The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read https://www.gnu.org/philosophy/why-not-lgpl.html.
Concept Index

A
aborting operations .................................. 111
algorithms .............................................. 15
algorithms, hash ....................................... 16
algorithms, message digest .......................... 16
algorithms, public key ................................. 15
armor mode ............................................. 35
ASCII armor ........................................... 35
ASSUAN .................................................. 14
attributes, of a key .................................... 57
auditlog .................................................. 46
auditlog, of the engine ............................... 46
autoconf ................................................ 5
automake ............................................... 5
cryptographic operation, signature check ........... 79
cryptographic operation, signing ..................... 86
cryptographic operation, verification ............... 79
cryptographic operation, wait for ................... 100

B
backend ................................................ 10

C
callback, passphrase .................................. 40
callback, progress meter .............................. 41
callback, status message .............................. 42
cancelling operations ................................ 111
canonical text mode ................................... 35
certificates, included ................................ 37
CMS ...................................................... 14
compiler flags ........................................ 3
compiler options ..................................... 3
configuration of crypto backend ...................... 14
core .................................................... 14
context ............................................... 33
context, armor mode .................................. 35
context, attributes ................................... 34
context, configuring engine .......................... 34
context, creation ..................................... 33
context, destruction ................................ 33
context, offline mode ................................ 36
context, pinentry mode ................................ 36
context, result of operation ......................... 33
context, selecting protocol ........................... 34
context, sender ....................................... 35
context, text mode ................................... 35
crypto backend ....................................... 10
crypto engine .......................................... 10
cryptographic message syntax ....................... 14
cryptographic operation .............................. 76
cryptographic operation, aborting .................. 111
cryptographic operation, cancelling ............... 111
cryptographic operation, decryption ............... 76
cryptographic operation, decryption and
verification ........................................... 85
cryptographic operation, encryption .............. 90
cryptographic operation, running .................. 100
data buffer, convenience ............................. 31
data buffer, creation ................................ 24
data buffer, destruction .............................. 28
data buffer, encoding ................................ 29
data buffer, file name ................................ 29
data buffer, I/O operations ........................... 28
data buffer, manipulation ............................. 28
data buffer, meta-data ................................. 29
data buffer, read ...................................... 28
data buffer, seek ..................................... 28
data buffer, write ..................................... 28
data, exchanging ..................................... 24
debug .................................................. 122
decryption .............................................. 76
decryption and verification ............................ 85
deprecated ............................................ 123
e
encryption ............................................. 90
eengine .................................................. 10
eengine, ASSUAN .................................... 14
eengine, configuration of ............................. 14
eengine, configuration per context ................... 34
eengine, GnuPG ...................................... 14
eengine, GpgSM ...................................... 14
eengine, information about ......................... 12
error codes .......................................... 17
error codes, list of .................................. 19, 20
error codes, printing of ............................... 22
error handling ....................................... 17
error sources ........................................ 17
error sources, printing of ............................ 22
error strings ........................................ 22
error values ......................................... 17
error values, printing of ............................. 22
event loop, external ................................. 100

F
flags, of a context ................................... 42
From: .................................................. 35

g
GDK, using GPGME with ......................... 109
GnuPG ............................................. 14
<table>
<thead>
<tr>
<th>Concept Index</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>GPGME_DEBUG</strong></td>
</tr>
<tr>
<td><strong>GpgSM</strong></td>
</tr>
<tr>
<td><strong>GTK+, using GPGME with</strong></td>
</tr>
<tr>
<td><strong>H</strong></td>
</tr>
<tr>
<td>hash algorithms</td>
</tr>
<tr>
<td>header file</td>
</tr>
<tr>
<td><strong>I</strong></td>
</tr>
<tr>
<td>identify</td>
</tr>
<tr>
<td>include file</td>
</tr>
<tr>
<td><strong>K</strong></td>
</tr>
<tr>
<td>key listing</td>
</tr>
<tr>
<td>key listing mode</td>
</tr>
<tr>
<td>key listing, mode of</td>
</tr>
<tr>
<td>key listing, start</td>
</tr>
<tr>
<td>key management</td>
</tr>
<tr>
<td>key ring, add</td>
</tr>
<tr>
<td>key ring, delete from</td>
</tr>
<tr>
<td>key ring, export from</td>
</tr>
<tr>
<td>key ring, import to</td>
</tr>
<tr>
<td>key ring, list</td>
</tr>
<tr>
<td>key ring, search</td>
</tr>
<tr>
<td>key, attributes</td>
</tr>
<tr>
<td>key, creation</td>
</tr>
<tr>
<td>key, delete</td>
</tr>
<tr>
<td>key, edit</td>
</tr>
<tr>
<td>key, export</td>
</tr>
<tr>
<td>key, import</td>
</tr>
<tr>
<td>key, information about</td>
</tr>
<tr>
<td>key, manipulation</td>
</tr>
<tr>
<td>key, signing</td>
</tr>
<tr>
<td><strong>L</strong></td>
</tr>
<tr>
<td>largefile support</td>
</tr>
<tr>
<td>LFS</td>
</tr>
<tr>
<td>LGPL, GNU Lesser General Public License</td>
</tr>
<tr>
<td>libtool</td>
</tr>
<tr>
<td>listing keys</td>
</tr>
<tr>
<td>locale, default</td>
</tr>
<tr>
<td>locale, of a context</td>
</tr>
<tr>
<td><strong>M</strong></td>
</tr>
<tr>
<td>message digest algorithms</td>
</tr>
<tr>
<td>multi-threading</td>
</tr>
<tr>
<td><strong>N</strong></td>
</tr>
<tr>
<td>notation data</td>
</tr>
<tr>
<td><strong>O</strong></td>
</tr>
<tr>
<td>offline mode</td>
</tr>
<tr>
<td>OpenPGP</td>
</tr>
<tr>
<td><strong>P</strong></td>
</tr>
<tr>
<td>passphrase callback</td>
</tr>
<tr>
<td>passphrase, change</td>
</tr>
<tr>
<td>pinentry mode</td>
</tr>
<tr>
<td>policy URL</td>
</tr>
<tr>
<td>progress meter callback</td>
</tr>
<tr>
<td>protocol</td>
</tr>
<tr>
<td>protocol, ASSUAN</td>
</tr>
<tr>
<td>protocol, CMS</td>
</tr>
<tr>
<td>protocol, GnuPG</td>
</tr>
<tr>
<td>protocol, S/MIME</td>
</tr>
<tr>
<td>protocol, selecting</td>
</tr>
<tr>
<td>public key algorithms</td>
</tr>
<tr>
<td><strong>Q</strong></td>
</tr>
<tr>
<td>Qt, using GPGME with</td>
</tr>
<tr>
<td><strong>R</strong></td>
</tr>
<tr>
<td>run control</td>
</tr>
<tr>
<td><strong>S</strong></td>
</tr>
<tr>
<td>S/MIME</td>
</tr>
<tr>
<td>sender</td>
</tr>
<tr>
<td>sign</td>
</tr>
<tr>
<td>signal handling</td>
</tr>
<tr>
<td>signals</td>
</tr>
<tr>
<td>signature check</td>
</tr>
<tr>
<td>signature notation data</td>
</tr>
<tr>
<td>signature, creation</td>
</tr>
<tr>
<td>signature, selecting signers</td>
</tr>
<tr>
<td>signature, verification</td>
</tr>
<tr>
<td>signers, selecting</td>
</tr>
<tr>
<td>status message callback</td>
</tr>
<tr>
<td><strong>T</strong></td>
</tr>
<tr>
<td>text mode</td>
</tr>
<tr>
<td>thread-safeness</td>
</tr>
<tr>
<td>type of data</td>
</tr>
<tr>
<td><strong>U</strong></td>
</tr>
<tr>
<td>UI server</td>
</tr>
<tr>
<td>user interface server</td>
</tr>
<tr>
<td><strong>V</strong></td>
</tr>
<tr>
<td>validity, TOFU</td>
</tr>
<tr>
<td>verification</td>
</tr>
</tbody>
</table>
verification and decryption.................. 85
version check, of the engines ................ 11
version check, of the library ................ 6

W

wait for completion.......................... 100
## Function and Data Index

### A
- AM_PATH_GPGME ............................................ 5

### C
- CHECKSUM_CREATE_FILES ................................. 119
- CHECKSUM_VERIFY_FILES ................................. 120

### D
- DECRYPT .................................................. 116
- DECRYPT_FILES .......................................... 118
- DECRYPT_VERIFY_FILES .................................. 118

### E
- ENCRYPT .................................................. 114
- ENCRYPT_FILES .......................................... 118
- ENCRYPT_SIGN_FILES .................................... 118
- enum gpgme_data_encoding_t ............................ 29
- enum gpgme_data_type_t ................................. 31
- enum gpgme_event_io_t ................................. 102
- enum gpgme_hash_algo_t ................................. 16
- enum gpgme_pinentry_mode_t ......................... 37
- enum gpgme_protocol_t ................................. 10
- enum gpgme_pubkey_algo_t ............................. 15
- enum gpgme_sig_mode_t ................................. 87
- enum gpgme_sig_stat_t ................................. 128
- enum gpgme_tofu_policy_t .............................. 74

### F
- FILE .................................................................. 118

### G
- GETINFO .................................................... 120
- gpgme_addspec_from_uid ................................. 86
- gpgme_attr_t ............................................. 125, 127
- gpgme_cancel ............................................ 112
- gpgme_cancel_async ...................................... 112
- gpgme_check_version .................................... 6
- gpgme_ctx_get_engine_info ............................. 34
- gpgme_ctx_set_engine_info ............................. 34
- gpgme_ctx_t ................................................ 33
- gpgme_data_encoding_t ................................. 29
- gpgme_data_get_encoding .............................. 30
- gpgme_data_get_file_name .............................. 29
- gpgme_data_identify .................................... 32
- gpgme_data_new .......................................... 24
- gpgme_data_new_from_cbs ............................ 27
- gpgme_data_new_from_estream ....................... 26
- gpgme_data_new_from_fd .............................. 25
- gpgme_data_new_from_filepart ..................... 25
- gpgme_data_new_from_mem ............................ 25
- gpgme_data_new_from_stream ....................... 26
- gpgme_data_new_with_read_cb ....................... 124
- gpgme_data_read .................................... 28
- gpgme_data_read_cb_t ................................. 26
- gpgme_data_release .................................... 28
- gpgme_data_release_and_get_mem .................. 28
- gpgme_data_release_cb_t ............................ 27
- gpgme_data_rewind .................................... 124
- gpgme_data_seek ........................................ 29
- gpgme_data_seek_cb_t ................................. 27
- gpgme_data_set_encoding ............................ 30
- gpgme_data_set_file_name ........................... 29
- gpgme_data_set_flag .................................. 31
- gpgme_data_t ............................................. 24
- gpgme_data_type_t ...................................... 31
- gpgme_data_write ...................................... 28
- gpgme_data_write_cb_t ............................... 27
- gpgme_decrypt ........................................... 78
- gpgme_decrypt_result_t .................. 123
- gpgme_decrypt_result_t .............................. 95
- gpgme_engine_check_version ....................... 12
- gpgme_engine_info_t ................................. 12
- gpgme_err_code ......................................... 18
- gpgme_err_code_from_errno ......................... 19
- gpgme_err_code_to_errno ............................. 19
- gpgme_err_make ......................................... 18
- gpgme_err_make_from_errno ......................... 18
- gpgme_err_source ....................................... 18
- gpgme_err_source_t .................................... 17
- gpgme_error .............................................. 18
- gpgme_error_t ........................................... 18
- gpgme_error_t (*gpgme_assuan_data_cb_t) (void *opaque, const void *data, size_t datalen) ..................... 97
- gpgme_error_t (*gpgme_assuan_inquire_cb_t) (void *opaque, const char *name, const char *args, gpgme_data_t *r_data) ..................... 97
- gpgme_error_t (*gpgme_assuan_status_cb_t) (void *opaque, const char *status, const char *args) ..................... 97
- gpgme_error_t (*gpgme_edit_cb_t) (void *handle, gpgme_status_code_t status, const char *args, int fd) ..................... 123
- gpgme_error_t (*gpgme_interact_cb_t) (void *handle, const char *status, const char *args, int fd) ..................... 75
- gpgme_error_t (*gpgme_io_cb_t) (void *data, int fd) ..................... 101

---

The above content is a representation of a function and data index from a document, showing various functions and data types with their associated documentation and usage examples.
Function and Data Index 159

<table>
<thead>
<tr>
<th>Function/Method</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>gpgme_op_query_swdb</td>
<td>98</td>
</tr>
<tr>
<td>gpgme_op_query_swdb_result</td>
<td>99</td>
</tr>
<tr>
<td>gpgme_op_receive_keys</td>
<td>71</td>
</tr>
<tr>
<td>gpgme_op_receive_keys_start</td>
<td>71</td>
</tr>
<tr>
<td>gpgme_op_revsig</td>
<td>66</td>
</tr>
<tr>
<td>gpgme_op_revsig_start</td>
<td>66</td>
</tr>
<tr>
<td>gpgme_op_revuid</td>
<td>62</td>
</tr>
<tr>
<td>gpgme_op_revuid_start</td>
<td>62</td>
</tr>
<tr>
<td>gpgme_op_set_uid_flag_start</td>
<td>63</td>
</tr>
<tr>
<td>gpgme_op_setexpire</td>
<td>58</td>
</tr>
<tr>
<td>gpgme_op_setexpire_start</td>
<td>58</td>
</tr>
<tr>
<td>gpgme_op_sign</td>
<td>88</td>
</tr>
<tr>
<td>gpgme_op_sign_result</td>
<td>89</td>
</tr>
<tr>
<td>gpgme_op_sign_start</td>
<td>88</td>
</tr>
<tr>
<td>gpgme_op_spaawn</td>
<td>96</td>
</tr>
<tr>
<td>gpgme_op_spaawn_start</td>
<td>96</td>
</tr>
<tr>
<td>gpgme_op_tofu_policy</td>
<td>75</td>
</tr>
<tr>
<td>gpgme_op_tofu_policy_start</td>
<td>75</td>
</tr>
<tr>
<td>gpgme_op_verify</td>
<td>79</td>
</tr>
<tr>
<td>gpgme_op_verify_ext</td>
<td>80</td>
</tr>
<tr>
<td>gpgme_op_verify_ext_start</td>
<td>80</td>
</tr>
<tr>
<td>gpgme_op_verify_result</td>
<td>85</td>
</tr>
<tr>
<td>gpgme_op_verify_start</td>
<td>80</td>
</tr>
<tr>
<td>gpgme_passphrase_cb_t</td>
<td>40</td>
</tr>
<tr>
<td>gpgme_pinentry_mode_t</td>
<td>37</td>
</tr>
<tr>
<td>gpgme_progress_cb_t</td>
<td>41</td>
</tr>
<tr>
<td>gpgme_protocol_t</td>
<td>10, 12</td>
</tr>
<tr>
<td>gpgme_pubkey_algo_name</td>
<td>16</td>
</tr>
<tr>
<td>gpgme_pubkey_algo_string</td>
<td>16</td>
</tr>
<tr>
<td>gpgme_pubkey_algo_t</td>
<td>15</td>
</tr>
<tr>
<td>gpgme_query_swdb_result</td>
<td>98</td>
</tr>
<tr>
<td>gpgme_recipient_t</td>
<td>78</td>
</tr>
<tr>
<td>gpgme_register_io_cb_t</td>
<td>101</td>
</tr>
<tr>
<td>gpgme_release</td>
<td>33</td>
</tr>
<tr>
<td>gpgme_result_ref</td>
<td>33</td>
</tr>
<tr>
<td>gpgme_result_unref</td>
<td>33</td>
</tr>
<tr>
<td>gpgme_set_armor</td>
<td>35</td>
</tr>
<tr>
<td>gpgme_set_ctx_flag</td>
<td>42</td>
</tr>
<tr>
<td>gpgme_set_engine_info</td>
<td>14</td>
</tr>
<tr>
<td>gpgme_set_global_flag</td>
<td>6</td>
</tr>
<tr>
<td>gpgme_set_include_certs</td>
<td>37</td>
</tr>
<tr>
<td>gpgme_set_io_cbs</td>
<td>103</td>
</tr>
<tr>
<td>gpgme_set_keylist_mode</td>
<td>38</td>
</tr>
<tr>
<td>gpgme_set_locale</td>
<td>45</td>
</tr>
<tr>
<td>gpgme_set_passphrase_cb</td>
<td>36</td>
</tr>
<tr>
<td>gpgme_set_pinentry_mode</td>
<td>36</td>
</tr>
<tr>
<td>gpgme_set_progress_cb</td>
<td>41</td>
</tr>
<tr>
<td>gpgme_set_protocol</td>
<td>34</td>
</tr>
<tr>
<td>gpgme_set_sender</td>
<td>35</td>
</tr>
<tr>
<td>gpgme_set_status_cb</td>
<td>42</td>
</tr>
<tr>
<td>gpgme_set_textmode</td>
<td>35</td>
</tr>
<tr>
<td>gpgme_sig_mode_t</td>
<td>87</td>
</tr>
<tr>
<td>gpgme_sig_notation_add</td>
<td>89</td>
</tr>
<tr>
<td>gpgme_sig_notation_clear</td>
<td>89</td>
</tr>
<tr>
<td>gpgme_sig_notation_get</td>
<td>90</td>
</tr>
<tr>
<td>gpgme_sig_notation_t</td>
<td>81</td>
</tr>
<tr>
<td>gpgme_sig_stat_t</td>
<td>128</td>
</tr>
<tr>
<td>gpgme_sign_result_t</td>
<td>89</td>
</tr>
<tr>
<td>gpgme_signature_t</td>
<td>82</td>
</tr>
<tr>
<td>gpgme_signers_add</td>
<td>87</td>
</tr>
<tr>
<td>gpgme_signers_clear</td>
<td>86</td>
</tr>
<tr>
<td>gpgme_signers_count</td>
<td>87</td>
</tr>
<tr>
<td>gpgme_signers_enum</td>
<td>87</td>
</tr>
<tr>
<td>gpgme_ssize_t</td>
<td>24</td>
</tr>
<tr>
<td>gpgme_status_cb_t</td>
<td>42</td>
</tr>
<tr>
<td>gpgme_strerror_r</td>
<td>42</td>
</tr>
<tr>
<td>gpgme_strerror_r</td>
<td>22</td>
</tr>
<tr>
<td>gpgme_strerror_r</td>
<td>22</td>
</tr>
<tr>
<td>gpgme_strerror_r</td>
<td>22</td>
</tr>
<tr>
<td>gpgme_subkey_t</td>
<td>49</td>
</tr>
<tr>
<td>gpgme_tofu_info_t</td>
<td>51</td>
</tr>
<tr>
<td>gpgme_tofu_policy_t</td>
<td>74</td>
</tr>
<tr>
<td>gpgme_user_id_t</td>
<td>50</td>
</tr>
<tr>
<td>gpgme_validity_t</td>
<td>57</td>
</tr>
<tr>
<td>gpgme_verify_result_t</td>
<td>85</td>
</tr>
<tr>
<td>gpgme_wait</td>
<td>100</td>
</tr>
</tbody>
</table>

**I**
- IMPORT_FILES: 119

**M**
- MESSAGE: 116
  - MICALG: 116

**O**
- off_t (*gpgme_data_seek_cb_t) (void *handle, off_t offset, int whence): 27
- OUTPUT: 113, 115, 116, 117

**P**
- PREP_ENCRYPT: 114
  - PROTOCOL: 114

**R**
- RECIPIENT: 113

**S**
- SENDER: 121
  - SESSION: 121
  - SIGN: 116
  - SIGN_FILES: 118
  - SIGSTATUS: 117
  - ssize_t (*gpgme_data_read_cb_t) (void *handle, void *buffer, size_t size): 26
  - ssize_t (*gpgme_data_write_cb_t) (void *handle, const void *buffer, size_t size): 27
V
VERIFY ........................................ 117
VERIFY_FILES .............................. 118
void (*gpgme_data_release_cb_t) (void *handle) .................... 27

W
window-id .................................. 120