The Libgcrypt Reference Manual

Version 1.9.4
22 August 2021

Werner Koch (wk@gnupg.org)
Moritz Schulte (mo@g10code.com)

This manual is for Libgcrypt version 1.9.4 and was last updated 22 August 2021. Libgcrypt is GNU's library of cryptographic building blocks.

Short Contents

Table of Contents

1 Introduction

Libgcrypt is a library providing cryptographic building blocks.

1.1 Getting Started

This manual documents the Libgcrypt library application programming interface (API). All functions and data types provided by the library are explained.

The reader is assumed to possess basic knowledge about applied cryptography.

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.

1.2 Features

Libgcrypt might have a couple of advantages over other libraries doing a similar job.

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 115). Note, that some parts (which are in general not needed by applications) are subject to the terms of the GNU General Public License (see [Copying], page 125); please see the README file of the distribution for of list of these parts.

It encapsulates the low level cryptography
: Libgcrypt provides a high level interface to cryptographic building blocks using an extensible and flexible API.

1.3 Overview

The Libgcrypt library is fully thread-safe, where it makes sense to be thread-safe. Not thread-safe are some cryptographic functions that modify a certain context stored in handles. If the user really intents to use such functions from different threads on the same handle, he has to take care of the serialization of such functions himself. If not described otherwise, every function is thread-safe.

Libgcrypt depends on the library 'libgpg-error', which contains some common code used by other GnuPG components.

2 Preparation

To use Libgcrypt, 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 `gcrypt.h`. You must include this in all source files using the library, either directly or through some other header file, like this:

```
#include <gcrypt.h>
```

The name space of Libgcrypt is `gcry_*` for function and type names and `GCRY*` for other symbols. In addition the same name prefixes with one prepended underscore are reserved for internal use and should never be used by an application. Note that Libgcrypt uses libgpg-error, which uses `gpg_*` as name space for function and type names and `GPG_*` for other symbols, including all the error codes.

Certain parts of gcrypt.h may be excluded by defining these macros:

`GCRYPT_NO_MPI_MACROS`
: Do not define the shorthand macros `mpi_*` for `gcry_mpi_*`.

`GCRYPT_NO_DEPRECATED`
: Do not include definitions for deprecated features. This is useful to make sure that no deprecated features are used.

2.2 Building sources

If you want to compile a source file including the 'gcrypt.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, Libgcrypt ships with a small helper program `libgcrypt-config` that knows the path to the include file and other configuration options. The options that need to be added to the compiler invocation at compile time are output by the `--cflags` option to `libgcrypt-config`. The following example shows how it can be used at the command line:

```
gcc -c foo.c `libgcrypt-config --cflags`
```

Adding the output of '`libgcrypt-config --cflags`' to the compiler's command line will ensure that the compiler can find the Libgcrypt 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 `libgcrypt-config` can be used. For convenience, this option also outputs all other options that are required to link the program with the Libgcrypt libraries (in particular, the

'-lgcrypt' option). The example shows how to link foo.o with the Libgcrypt library to a program foo.

```
gcc -o foo foo.o `libgcrypt-config --libs`
```

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

```
gcc -o foo foo.c `libgcrypt-config --cflags --libs`
```

2.3 Building sources using Automake

It is much easier if you use GNU Automake instead of writing your own Makefiles. If you do that, you do not have to worry about finding and invoking the libgcrypt-config script at all. Libgcrypt provides an extension to Automake that does all the work for you.

AM_PATH_LIBGCRYPT ([*minimum-version*], [*action-if-found*], [*action-if-not-found*]) [Macro]

Check whether Libgcrypt (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.

Additionally, the function defines LIBGCRYPT_CFLAGS to the flags needed for compilation of the program to find the gcrypt.h header file, and LIBGCRYPT_LIBS to the linker flags needed to link the program to the Libgcrypt library. If the used helper script does not match the target type you are building for a warning is printed and the string libgcrypt is appended to the variable gpg_config_script_warn.

This macro searches for libgcrypt-config along the PATH. If you are cross-compiling, it is useful to set the environment variable SYSROOT to the top directory of your target. The macro will then first look for the helper program in the bin directory below that top directory. An absolute directory name must be used for SYSROOT. Finally, if the configure command line option --with-libgcrypt-prefix is used, only its value is used for the top directory below which the helper script is expected.

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

```
AM_CPPFLAGS = $(LIBGCRYPT_CFLAGS)
LDADD = $(LIBGCRYPT_LIBS)
```

2.4 Initializing the library

Before the library can be used, it must initialize itself. This is achieved by invoking the function gcry_check_version described below.

Also, it is often desirable to check that the version of Libgcrypt used is indeed one which fits all requirements. Even with binary compatibility, new features may have been introduced, but due to problem with the dynamic linker an old version may actually be used. So you may want to check that the version is okay right after program startup.

const char * gcry_check_version (*const char *req_version*) [Function]

The function gcry_check_version initializes some subsystems used by Libgcrypt and must be invoked before any other function in the library. See Section 2.5 [Multi-Threading], page 6.

Furthermore, this function returns the version number of the library. It can also verify that the version number is higher than a certain required version number *req_version*, if this value is not a null pointer.

Libgcrypt uses a concept known as secure memory, which is a region of memory set aside for storing sensitive data. Because such memory is a scarce resource, it needs to be setup in advanced to a fixed size. Further, most operating systems have special requirements on how that secure memory can be used. For example, it might be required to install an application as "setuid(root)" to allow allocating such memory. Libgcrypt requires a sequence of initialization steps to make sure that this works correctly. The following examples show the necessary steps.

If you don't have a need for secure memory, for example if your application does not use secret keys or other confidential data or it runs in a controlled environment where key material floating around in memory is not a problem, you should initialize Libgcrypt this way:

```
/* Version check should be the very first call because it
   makes sure that important subsystems are initialized.
   #define NEED_LIBGCRYPT_VERSION to the minimum required version. */
if (!gcry_check_version (NEED_LIBGCRYPT_VERSION))
  {
    fprintf (stderr, "libgcrypt is too old (need %s, have %s)\n",
       NEED_LIBGCRYPT_VERSION, gcry_check_version (NULL));
    exit (2);
  }

/* Disable secure memory.  */
gcry_control (GCRYCTL_DISABLE_SECMEM, 0);

/* ... If required, other initialization goes here.  */

/* Tell Libgcrypt that initialization has completed. */
gcry_control (GCRYCTL_INITIALIZATION_FINISHED, 0);
```

If you have to protect your keys or other information in memory against being swapped out to disk and to enable an automatic overwrite of used and freed memory, you need to initialize Libgcrypt this way:

```
/* Version check should be the very first call because it
   makes sure that important subsystems are initialized.
   #define NEED_LIBGCRYPT_VERSION to the minimum required version. */
if (!gcry_check_version (NEED_LIBGCRYPT_VERSION))
  {
    fprintf (stderr, "libgcrypt is too old (need %s, have %s)\n",
       NEED_LIBGCRYPT_VERSION, gcry_check_version (NULL));
    exit (2);
  }
```

```
/* We don't want to see any warnings, e.g. because we have not yet
   parsed program options which might be used to suppress such
   warnings. */
gcry_control (GCRYCTL_SUSPEND_SECMEM_WARN);

/* ... If required, other initialization goes here.  Note that the
   process might still be running with increased privileges and that
   the secure memory has not been initialized.  */

/* Allocate a pool of 16k secure memory.  This makes the secure memory
   available and also drops privileges where needed.  Note that by
   using functions like gcry_xmalloc_secure and gcry_mpi_snew Libgcrypt
   may expand the secure memory pool with memory which lacks the
   property of not being swapped out to disk.   */
gcry_control (GCRYCTL_INIT_SECMEM, 16384, 0);


/* It is now okay to let Libgcrypt complain when there was/is
   a problem with the secure memory. */
gcry_control (GCRYCTL_RESUME_SECMEM_WARN);

/* ... If required, other initialization goes here.  */

/* Tell Libgcrypt that initialization has completed. */
gcry_control (GCRYCTL_INITIALIZATION_FINISHED, 0);
```

It is important that these initialization steps are not done by a library but by the actual application. A library using Libgcrypt might want to check for finished initialization using:

```
if (!gcry_control (GCRYCTL_INITIALIZATION_FINISHED_P))
  {
    fputs ("libgcrypt has not been initialized\n", stderr);
    abort ();
  }
```

Instead of terminating the process, the library may instead print a warning and try to initialize Libgcrypt itself. See also the section on multi-threading below for more pitfalls.

2.5 Multi-Threading

As mentioned earlier, the Libgcrypt library is thread-safe if you adhere to the following requirements:

- If you use pthread and your applications forks and does not directly call exec (even calling stdio functions), all kind of problems may occur. Future versions of Libgcrypt will try to cleanup using pthread_atfork but even that may lead to problems. This is a common problem with almost all applications using pthread and fork.
- The function `gcry_check_version` must be called before any other function in the library. To achieve this in multi-threaded programs, you must synchronize the memory

with respect to other threads that also want to use Libgcrypt. For this, it is sufficient to call gcry_check_version before creating the other threads using Libgcrypt[1].

- Just like the function gpg_strerror, the function gcry_strerror is not thread safe. You have to use gpg_strerror_r instead.

2.6 How to enable the FIPS mode

Libgcrypt may be used in a FIPS 140-2 mode. Note, that this does not necessary mean that Libcgrypt is an appoved FIPS 140-2 module. Check the NIST database at http://csrc.nist.gov/groups/STM/cmvp/ to see what versions of Libgcrypt are approved.

Because FIPS 140 has certain restrictions on the use of cryptography which are not always wanted, Libgcrypt needs to be put into FIPS mode explicitly. Three alternative mechanisms are provided to switch Libgcrypt into this mode:

- If the file /proc/sys/crypto/fips_enabled exists and contains a numeric value other than 0, Libgcrypt is put into FIPS mode at initialization time. Obviously this works only on systems with a proc file system (i.e. GNU/Linux).
- If the file /etc/gcrypt/fips_enabled exists, Libgcrypt is put into FIPS mode at initialization time. Note that this filename is hardwired and does not depend on any configuration options.
- If the application requests FIPS mode using the control command GCRYCTL_FORCE_FIPS_MODE. This must be done prior to any initialization (i.e. before gcry_check_version).

In addition to the standard FIPS mode, Libgcrypt may also be put into an Enforced FIPS mode by writing a non-zero value into the file /etc/gcrypt/fips_enabled or by using the control command GCRYCTL_SET_ENFORCED_FIPS_FLAG before any other calls to libgcrypt. The Enforced FIPS mode helps to detect applications which don't fulfill all requirements for using Libgcrypt in FIPS mode (see Appendix B [FIPS Mode], page 109).

Once Libgcrypt has been put into FIPS mode, it is not possible to switch back to standard mode without terminating the process first. If the logging verbosity level of Libgcrypt has been set to at least 2, the state transitions and the self-tests are logged.

2.7 How to disable hardware features

Libgcrypt makes use of certain hardware features. If the use of a feature is not desired it may be either be disabled by a program or globally using a configuration file. The currently supported features are

[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, more relaxed or more strict rules may apply.

padlock-rng
padlock-aes
padlock-sha
padlock-mmul
intel-cpu
intel-fast-shld
intel-bmi2
intel-ssse3
intel-sse4.1
intel-pclmul
intel-aesni
intel-rdrand
intel-avx
intel-avx2
intel-fast-vpgather
intel-rdtsc
intel-shaext
intel-vaes-vpclmul
arm-neon

arm-aes

arm-sha1

arm-sha2

arm-pmull
ppc-vcrypto
ppc-arch_3_00
ppc-arch_2_07
s390x-msa
s390x-msa-4
s390x-msa-8
s390x-vx

To disable a feature for all processes using Libgcrypt 1.6 or newer, create the file `/etc/gcrypt/hwf.deny` and put each feature not to be used on a single line. Empty lines, white space, and lines prefixed with a hash mark are ignored. The file should be world readable.

To disable a feature specifically for a program that program must tell it Libgcrypt before before calling `gcry_check_version`. Example:[2]

```
gcry_control (GCRYCTL_DISABLE_HWF, "intel-rdrand", NULL);
```

To print the list of active features you may use this command:

```
mpicalc --print-config | grep ^hwflist: | tr : '\n' | tail -n +2
```

[2] NB. Libgcrypt uses the RDRAND feature only as one source of entropy. A CPU with a broken RDRAND will thus not compromise of the random number generator

3 Generalities

3.1 Controlling the library

`gcry_error_t gcry_control` (*enum gcry_ctl_cmds* `cmd`, *...*) [Function]

This function can be used to influence the general behavior of Libgcrypt in several ways. Depending on *cmd*, more arguments can or have to be provided.

`GCRYCTL_ENABLE_M_GUARD; Arguments: none`
: This command enables the built-in memory guard. It must not be used to activate the memory guard after the memory management has already been used; therefore it can ONLY be used before `gcry_check_version`. Note that the memory guard is NOT used when the user of the library has set his own memory management callbacks.

`GCRYCTL_ENABLE_QUICK_RANDOM; Arguments: none`
: This command inhibits the use the very secure random quality level (`GCRY_VERY_STRONG_RANDOM`) and degrades all request down to `GCRY_STRONG_RANDOM`. In general this is not recommended. However, for some applications the extra quality random Libgcrypt tries to create is not justified and this option may help to get better performance. Please check with a crypto expert whether this option can be used for your application.

  This option can only be used at initialization time.

`GCRYCTL_DUMP_RANDOM_STATS; Arguments: none`
: This command dumps random number generator related statistics to the library's logging stream.

`GCRYCTL_DUMP_MEMORY_STATS; Arguments: none`
: This command dumps memory management related statistics to the library's logging stream.

`GCRYCTL_DUMP_SECMEM_STATS; Arguments: none`
: This command dumps secure memory management related statistics to the library's logging stream.

`GCRYCTL_DROP_PRIVS; Arguments: none`
: This command disables the use of secure memory and drops the privileges of the current process. This command has not much use; the suggested way to disable secure memory is to use `GCRYCTL_DISABLE_SECMEM` right after initialization.

`GCRYCTL_DISABLE_SECMEM; Arguments: none`
: This command disables the use of secure memory. If this command is used in FIPS mode, FIPS mode will be disabled and the function `gcry_fips_mode_active` returns false. However, in Enforced FIPS mode this command has no effect at all.

  Many applications do not require secure memory, so they should disable it right away. This command should be executed right after `gcry_check_version`.

`GCRYCTL_DISABLE_LOCKED_SECMEM; Arguments: none`
: This command disables the use of the mlock call for secure memory. Disabling the use of mlock may for example be done if an encrypted swap space is in use. This command should be executed right after `gcry_check_version`. Note that by using functions like gcry_xmalloc_secure and gcry_mpi_snew Libgcrypt may expand the secure memory pool with memory which lacks the property of not being swapped out to disk (but will still be zeroed out on free).

`GCRYCTL_DISABLE_PRIV_DROP; Arguments: none`
: This command sets a global flag to tell the secure memory subsystem that it shall not drop privileges after secure memory has been allocated. This command is commonly used right after `gcry_check_version` but may also be used right away at program startup. It won't have an effect after the secure memory pool has been initialized. WARNING: A process running setuid(root) is a severe security risk. Processes making use of Libgcrypt or other complex code should drop these extra privileges as soon as possible. If this command has been used the caller is responsible for dropping the privileges.

`GCRYCTL_INIT_SECMEM; Arguments: unsigned int nbytes`
: This command is used to allocate a pool of secure memory and thus enabling the use of secure memory. It also drops all extra privileges the process has (i.e. if it is run as setuid (root)). If the argument *nbytes* is 0, secure memory will be disabled. The minimum amount of secure memory allocated is currently 16384 bytes; you may thus use a value of 1 to request that default size.

`GCRYCTL_AUTO_EXPAND_SECMEM; Arguments: unsigned int chunksize`
: This command enables on-the-fly expanding of the secure memory area. Note that by using functions like `gcry_xmalloc_secure` and `gcry_mpi_snew` will do this auto expanding anyway. The argument to this option is the suggested size for new secure memory areas. A larger size improves performance of all memory allocation and releasing functions. The given chunksize is rounded up to the next 32KiB. The drawback of auto expanding is that memory might be swapped out to disk; this can be fixed by configuring the system to use an encrypted swap space.

`GCRYCTL_TERM_SECMEM; Arguments: none`
: This command zeroises the secure memory and destroys the handler. The secure memory pool may not be used anymore after running this command. If the secure memory pool as already been destroyed, this command has no effect. Applications might want to run this command from their exit handler to make sure that the secure memory gets properly destroyed. This command is not necessarily thread-safe but that should not be needed in cleanup code. It may be called from a signal handler.

`GCRYCTL_DISABLE_SECMEM_WARN; Arguments: none`
: Disable warning messages about problems with the secure memory subsystem. This command should be run right after `gcry_check_version`.

`GCRYCTL_SUSPEND_SECMEM_WARN; Arguments: none`

Postpone warning messages from the secure memory subsystem. See [the initialization example], page 5, on how to use it.

`GCRYCTL_RESUME_SECMEM_WARN; Arguments: none`

Resume warning messages from the secure memory subsystem. See [the initialization example], page 6, on how to use it.

`GCRYCTL_USE_SECURE_RNDPOOL; Arguments: none`

This command tells the PRNG to store random numbers in secure memory. This command should be run right after `gcry_check_version` and not later than the command GCRYCTL_INIT_SECMEM. Note that in FIPS mode the secure memory is always used.

`GCRYCTL_SET_RANDOM_SEED_FILE; Arguments: const char *filename`

This command specifies the file, which is to be used as seed file for the PRNG. If the seed file is registered prior to initialization of the PRNG, the seed file's content (if it exists and seems to be valid) is fed into the PRNG pool. After the seed file has been registered, the PRNG can be signalled to write out the PRNG pool's content into the seed file with the following command.

`GCRYCTL_UPDATE_RANDOM_SEED_FILE; Arguments: none`

Write out the PRNG pool's content into the registered seed file.

Multiple instances of the applications sharing the same random seed file can be started in parallel, in which case they will read out the same pool and then race for updating it (the last update overwrites earlier updates). They will differentiate only by the weak entropy that is added in read_seed_file based on the PID and clock, and up to 16 bytes of weak random non-blockingly. The consequence is that the output of these different instances is correlated to some extent. In a perfect attack scenario, the attacker can control (or at least guess) the PID and clock of the application, and drain the system's entropy pool to reduce the "up to 16 bytes" above to 0. Then the dependencies of the initial states of the pools are completely known. Note that this is not an issue if random of `GCRY_VERY_STRONG_RANDOM` quality is requested as in this case enough extra entropy gets mixed. It is also not an issue when using Linux (rndlinux driver), because this one guarantees to read full 16 bytes from /dev/urandom and thus there is no way for an attacker without kernel access to control these 16 bytes.

`GCRYCTL_CLOSE_RANDOM_DEVICE; Arguments: none`

Try to close the random device. If on Unix system you call fork(), the child process does no call exec(), and you do not intend to use Libgcrypt in the child, it might be useful to use this control code to close the inherited file descriptors of the random device. If Libgcrypt is later used again by the child, the device will be re-opened. On non-Unix systems this control code is ignored.

`GCRYCTL_SET_VERBOSITY; Arguments: int level`
: This command sets the verbosity of the logging. A level of 0 disables all extra logging whereas positive numbers enable more verbose logging. The level may be changed at any time but be aware that no memory synchronization is done so the effect of this command might not immediately show up in other threads. This command may even be used prior to `gcry_check_version`.

`GCRYCTL_SET_DEBUG_FLAGS; Arguments: unsigned int flags`
: Set the debug flag bits as given by the argument. Be aware that no memory synchronization is done so the effect of this command might not immediately show up in other threads. The debug flags are not considered part of the API and thus may change without notice. As of now bit 0 enables debugging of cipher functions and bit 1 debugging of multi-precision-integers. This command may even be used prior to `gcry_check_version`.

`GCRYCTL_CLEAR_DEBUG_FLAGS; Arguments: unsigned int flags`
: Set the debug flag bits as given by the argument. Be aware that that no memory synchronization is done so the effect of this command might not immediately show up in other threads. This command may even be used prior to `gcry_check_version`.

`GCRYCTL_DISABLE_INTERNAL_LOCKING; Arguments: none`
: This command does nothing. It exists only for backward compatibility.

`GCRYCTL_ANY_INITIALIZATION_P; Arguments: none`
: This command returns true if the library has been basically initialized. Such a basic initialization happens implicitly with many commands to get certain internal subsystems running. The common and suggested way to do this basic initialization is by calling gcry_check_version.

`GCRYCTL_INITIALIZATION_FINISHED; Arguments: none`
: This command tells the library that the application has finished the initialization.

`GCRYCTL_INITIALIZATION_FINISHED_P; Arguments: none`
: This command returns true if the command GCRYCTL_INITIALIZATION_FINISHED has already been run.

`GCRYCTL_SET_THREAD_CBS; Arguments: struct ath_ops *ath_ops`
: This command is obsolete since version 1.6.

`GCRYCTL_FAST_POLL; Arguments: none`
: Run a fast random poll.

`GCRYCTL_SET_RNDEGD_SOCKET; Arguments: const char *filename`
: This command may be used to override the default name of the EGD socket to connect to. It may be used only during initialization as it is not thread safe. Changing the socket name again is not supported. The function may return an error if the given filename is too long for a local socket name.

EGD is an alternative random gatherer, used only on systems lacking a proper random device.

`GCRYCTL_PRINT_CONFIG; Arguments: FILE *stream`
: This command dumps information pertaining to the configuration of the library to the given stream. If NULL is given for *stream*, the log system is used. This command may be used before the initialization has been finished but not before a `gcry_check_version`. Note that the macro `estream_t` can be used instead of `gpgrt_stream_t`.

`GCRYCTL_OPERATIONAL_P; Arguments: none`
: This command returns true if the library is in an operational state. This information makes only sense in FIPS mode. In contrast to other functions, this is a pure test function and won't put the library into FIPS mode or change the internal state. This command may be used before the initialization has been finished but not before a `gcry_check_version`.

`GCRYCTL_FIPS_MODE_P; Arguments: none`
: This command returns true if the library is in FIPS mode. Note, that this is no indication about the current state of the library. This command may be used before the initialization has been finished but not before a `gcry_check_version`. An application may use this command or the convenience macro below to check whether FIPS mode is actually active.

`int gcry_fips_mode_active (void)` [Function]
: Returns true if the FIPS mode is active. Note that this is implemented as a macro.

`GCRYCTL_FORCE_FIPS_MODE; Arguments: none`
: Running this command puts the library into FIPS mode. If the library is already in FIPS mode, a self-test is triggered and thus the library will be put into operational state. This command may be used before a call to `gcry_check_version` and that is actually the recommended way to let an application switch the library into FIPS mode. Note that Libgcrypt will reject an attempt to switch to fips mode during or after the initialization.

`GCRYCTL_SET_ENFORCED_FIPS_FLAG; Arguments: none`
: Running this command sets the internal flag that puts the library into the enforced FIPS mode during the FIPS mode initialization. This command does not affect the library if the library is not put into the FIPS mode and it must be used before any other libgcrypt library calls that initialize the library such as `gcry_check_version`. Note that Libgcrypt will reject an attempt to switch to the enforced fips mode during or after the initialization.

`GCRYCTL_SET_PREFERRED_RNG_TYPE; Arguments: int`
: These are advisory commands to select a certain random number generator. They are only advisory because libraries may not know what an application actually wants or vice versa. Thus Libgcrypt employs a priority check to select the actually used RNG. If an applications selects a lower priority RNG but a library requests a higher priority RNG

Libgcrypt will switch to the higher priority RNG. Applications and libraries should use these control codes before `gcry_check_version`. The available generators are:

`GCRY_RNG_TYPE_STANDARD`
: A conservative standard generator based on the "Continuously Seeded Pseudo Random Number Generator" designed by Peter Gutmann.

`GCRY_RNG_TYPE_FIPS`
: A deterministic random number generator conforming to he document "NIST-Recommended Random Number Generator Based on ANSI X9.31 Appendix A.2.4 Using the 3-Key Triple DES and AES Algorithms" (2005-01-31). This implementation uses the AES variant.

`GCRY_RNG_TYPE_SYSTEM`
: A wrapper around the system's native RNG. On Unix system these are usually the /dev/random and /dev/urandom devices.

The default is `GCRY_RNG_TYPE_STANDARD` unless FIPS mode as been enabled; in which case `GCRY_RNG_TYPE_FIPS` is used and locked against further changes.

`GCRYCTL_GET_CURRENT_RNG_TYPE; Arguments: int *`
: This command stores the type of the currently used RNG as an integer value at the provided address.

`GCRYCTL_SELFTEST; Arguments: none`
: This may be used at anytime to have the library run all implemented self-tests. It works in standard and in FIPS mode. Returns 0 on success or an error code on failure.

`GCRYCTL_DISABLE_HWF; Arguments: const char *name`
: Libgcrypt detects certain features of the CPU at startup time. For performance tests it is sometimes required not to use such a feature. This option may be used to disable a certain feature; i.e. Libgcrypt behaves as if this feature has not been detected. This call can be used several times to disable a set of features, or features may be given as a colon or comma delimited string. The special feature "all" can be used to disable all available features.

  Note that the detection code might be run if the feature has been disabled. This command must be used at initialization time; i.e. before calling `gcry_check_version`.

`GCRYCTL_REINIT_SYSCALL_CLAMP; Arguments: none`
: Libgcrypt wraps blocking system calls with two functions calls ("system call clamp") to give user land threading libraries a hook for re-scheduling. This works by reading the system call clamp from Libgpg-error at initialization time. However sometimes Libgcrypt needs to be initialized before

the user land threading systems and at that point the system call clamp has not been registered with Libgpg-error and in turn Libgcrypt would not use them. The control code can be used to tell Libgcrypt that a system call clamp has now been registered with Libgpg-error and advise Libgcrypt to read the clamp again. Obviously this control code may only be used before a second thread is started in a process.

3.2 Error Handling

Many functions in Libgcrypt 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.

Libgcrypt uses the `libgpg-error` library. This allows to share the error codes with other components of the GnuPG system, and to 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, Libgcrypt does not use its own identifiers for error codes, but uses those provided by `libgpg-error`. They usually start with `GPG_ERR_`.

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

Most functions in Libgcrypt return an error code in the case of failure. 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 canceling 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.

GnuPG components, including Libgcrypt, use an extra library named libgpg-error to provide a common error handling scheme. For more information on libgpg-error, see the according manual.

3.2.1 Error Values

`gcry_err_code_t` [Data type]

The `gcry_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.

`gcry_err_source_t` [Data type]

The `gcry_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.

`gcry_error_t` [Data type]

The `gcry_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 Libgcrypt, the error source is used purely for diagnostic 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.

`gcry_err_code_t gcry_err_code` (*gcry_error_t err*) [Function]

The static inline function `gcry_err_code` returns the `gcry_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.

`gcry_err_source_t gcry_err_source` (*gcry_error_t err*) [Function]

The static inline function `gcry_err_source` returns the `gcry_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.

`gcry_error_t gcry_err_make` (*gcry_err_source_t source*, *gcry_err_code_t code*) [Function]

The static inline function `gcry_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.

`gcry_error_t gcry_error` (*gcry_err_code_t code*) [Function]

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

For GCRY applications, the default error source is `GPG_ERR_SOURCE_USER_1`. You can define `GCRY_ERR_SOURCE_DEFAULT` before including `gcrypt.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 errno numbers.

`gcry_error_t gcry_err_make_from_errno` (*gcry_err_source_t* `source`, *int* `err`) [Function]

The function `gcry_err_make_from_errno` is like `gcry_err_make`, but it takes a system error like `errno` instead of a `gcry_err_code_t` error code.

`gcry_error_t gcry_error_from_errno` (*int* `err`) [Function]

The function `gcry_error_from_errno` is like `gcry_error`, but it takes a system error like `errno` instead of a `gcry_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.

`gcry_err_code_t gcry_err_code_from_errno` (*int* `err`) [Function]

The function `gcry_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`.

`int gcry_err_code_to_errno` (*gcry_err_code_t* `err`) [Function]

The function `gcry_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`.

3.2.2 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 Libgcrypt 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.

`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 OpenPGP 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 Libgcrypt can use them to mark error values coming from callback handlers. Thus GPG_ERR_SOURCE_USER_1 is the default for errors created with gcry_error and gcry_error_from_errno, unless you define GCRY_ERR_SOURCE_DEFAULT before including gcrypt.h.

3.2.3 Error Codes

The library libgpg-error defines many error values. The following list includes the most important error codes.

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_EFOO 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.

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 data was expected where no data was found.

`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_WRONG_KEY_USAGE`
: This value indicates that a key is not used appropriately.

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

`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_NOT_OPERATIONAL`
: This value means that the library is not yet in state which allows to use this function. This error code is in particular returned if Libgcrypt is operated in FIPS mode and the internal state of the library does not yet or not anymore allow the use of a service.

  This error code is only available with newer libgpg-error versions, thus you might see "invalid error code" when passing this to `gpg_strerror`. The numeric value of this error code is 176.

`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 Libgcrypt 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.

3.2.4 Error Strings

`const char * gcry_strerror` (*gcry_error_t err*) [Function]

The function `gcry_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.

`const char * gcry_strsource` (*gcry_error_t err*) [Function]

The function `gcry_strsource` 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 the functions described above:

```
{
  gcry_cipher_hd_t handle;
  gcry_error_t err = 0;

  err = gcry_cipher_open (&handle, GCRY_CIPHER_AES,
                          GCRY_CIPHER_MODE_CBC, 0);
  if (err)
    {
      fprintf (stderr, "Failure: %s/%s\n",
               gcry_strsource (err),
               gcry_strerror (err));
    }
}
```

4 Handler Functions

Libgcrypt makes it possible to install so called 'handler functions', which get called by Libgcrypt in case of certain events.

4.1 Progress handler

It is often useful to retrieve some feedback while long running operations are performed.

`gcry_handler_progress_t` [Data type]

Progress handler functions have to be of the type `gcry_handler_progress_t`, which is defined as:

```
void (*gcry_handler_progress_t) (void *, const char *, int, int, int)
```

The following function may be used to register a handler function for this purpose.

`void gcry_set_progress_handler` (*gcry_handler_progress_t* `cb`, *void* *`cb_data`) [Function]

This function installs *cb* as the 'Progress handler' function. It may be used only during initialization. *cb* must be defined as follows:

```
void
my_progress_handler (void *cb_data, const char *what,
                     int printchar, int current, int total)
{
  /* Do something.  */
}
```

A description of the arguments of the progress handler function follows.

cb_data The argument provided in the call to `gcry_set_progress_handler`.

what A string identifying the type of the progress output. The following values for *what* are defined:

- `need_entropy`
  Not enough entropy is available. *total* holds the number of required bytes.
- `wait_dev_random`
  Waiting to re-open a random device. *total* gives the number of seconds until the next try.
- `primegen` Values for *printchar*:
  - `\n` Prime generated.
  - `!` Need to refresh the pool of prime numbers.
  - `<, >` Number of bits adjusted.
  - `^` Searching for a generator.
  - `.` Fermat test on 10 candidates failed.
  - `:` Restart with a new random value.
  - `+` Rabin Miller test passed.

4.2 Allocation handler

It is possible to make Libgcrypt use special memory allocation functions instead of the built-in ones.

Memory allocation functions are of the following types:

`gcry_handler_alloc_t` [Data type]

This type is defined as: `void *(*gcry_handler_alloc_t) (size_t n)`.

`gcry_handler_secure_check_t` [Data type]

This type is defined as: `int *(*gcry_handler_secure_check_t) (const void *)`.

`gcry_handler_realloc_t` [Data type]

This type is defined as: `void *(*gcry_handler_realloc_t) (void *p, size_t n)`.

`gcry_handler_free_t` [Data type]

This type is defined as: `void *(*gcry_handler_free_t) (void *)`.

Special memory allocation functions can be installed with the following function:

`void gcry_set_allocation_handler` (*gcry_handler_alloc_t* ***func_alloc***, *gcry_handler_alloc_t* ***func_alloc_secure***, *gcry_handler_secure_check_t* ***func_secure_check***, *gcry_handler_realloc_t* ***func_realloc***, *gcry_handler_free_t* ***func_free***) [Function]

Install the provided functions and use them instead of the built-in functions for doing memory allocation. Using this function is in general not recommended because the standard Libgcrypt allocation functions are guaranteed to zeroize memory if needed.

This function may be used only during initialization and may not be used in fips mode.

4.3 Error handler

The following functions may be used to register handler functions that are called by Libgcrypt in case certain error conditions occur. They may and should be registered prior to calling `gcry_check_version`.

`gcry_handler_no_mem_t` [Data type]

This type is defined as: `int (*gcry_handler_no_mem_t) (void *, size_t, unsigned int)`

`void gcry_set_outofcore_handler` (*gcry_handler_no_mem_t* ***func_no_mem***, *void* ****cb_data***) [Function]

This function registers *func_no_mem* as 'out-of-core handler', which means that it will be called in the case of not having enough memory available. The handler is called with 3 arguments: The first one is the pointer *cb_data* as set with this function, the second is the requested memory size and the last being a flag. If bit 0 of the flag is set, secure memory has been requested. The handler should either return true to indicate that Libgcrypt should try again allocating memory or return false to let Libgcrypt use its default fatal error handler.

`gcry_handler_error_t` [Data type]

This type is defined as: `void (*gcry_handler_error_t) (void *, int, const char *)`

`void gcry_set_fatalerror_handler` (*gcry_handler_error_t* *`func_error`*, *void* ***`cb_data`*) [Function]

This function registers *func_error* as 'error handler', which means that it will be called in error conditions.

4.4 Logging handler

`gcry_handler_log_t` [Data type]

This type is defined as: `void (*gcry_handler_log_t) (void *, int, const char *, va_list)`

`void gcry_set_log_handler` (*gcry_handler_log_t* *`func_log`*, *void* ***`cb_data`*) [Function]

This function registers *func_log* as 'logging handler', which means that it will be called in case Libgcrypt wants to log a message. This function may and should be used prior to calling `gcry_check_version`.

5 Symmetric cryptography

The cipher functions are used for symmetrical cryptography, i.e. cryptography using a shared key. The programming model follows an open/process/close paradigm and is in that similar to other building blocks provided by Libgcrypt.

5.1 Available ciphers

`GCRY_CIPHER_NONE`
: This is not a real algorithm but used by some functions as error return. The value always evaluates to false.

`GCRY_CIPHER_IDEA`
: This is the IDEA algorithm.

`GCRY_CIPHER_3DES`
: Triple-DES with 3 Keys as EDE. The key size of this algorithm is 168 bits but you have to pass 192 bits because the most significant bits of each byte are ignored.

`GCRY_CIPHER_CAST5`
: CAST128-5 block cipher algorithm. The key size is 128 bits.

`GCRY_CIPHER_BLOWFISH`
: The blowfish algorithm. The supported key sizes are 8 to 576 bits in 8 bit increments.

`GCRY_CIPHER_SAFER_SK128`
: Reserved and not currently implemented.

`GCRY_CIPHER_DES_SK`
: Reserved and not currently implemented.

`GCRY_CIPHER_AES`
`GCRY_CIPHER_AES128`
`GCRY_CIPHER_RIJNDAEL`
`GCRY_CIPHER_RIJNDAEL128`
: AES (Rijndael) with a 128 bit key.

`GCRY_CIPHER_AES192`
`GCRY_CIPHER_RIJNDAEL192`
: AES (Rijndael) with a 192 bit key.

`GCRY_CIPHER_AES256`
`GCRY_CIPHER_RIJNDAEL256`
: AES (Rijndael) with a 256 bit key.

`GCRY_CIPHER_TWOFISH`
: The Twofish algorithm with a 256 bit key.

`GCRY_CIPHER_TWOFISH128`
: The Twofish algorithm with a 128 bit key.

`GCRY_CIPHER_ARCFOUR`
: An algorithm which is 100% compatible with RSA Inc.'s RC4 algorithm. Note that this is a stream cipher and must be used very carefully to avoid a couple of weaknesses.

`GCRY_CIPHER_DES`
: Standard DES with a 56 bit key. You need to pass 64 bit but the high bits of each byte are ignored. Note, that this is a weak algorithm which can be broken in reasonable time using a brute force approach.

`GCRY_CIPHER_SERPENT128`
`GCRY_CIPHER_SERPENT192`
`GCRY_CIPHER_SERPENT256`
: The Serpent cipher from the AES contest.

`GCRY_CIPHER_RFC2268_40`
`GCRY_CIPHER_RFC2268_128`
: Ron's Cipher 2 in the 40 and 128 bit variants.

`GCRY_CIPHER_SEED`
: A 128 bit cipher as described by RFC4269.

`GCRY_CIPHER_CAMELLIA128`
`GCRY_CIPHER_CAMELLIA192`
`GCRY_CIPHER_CAMELLIA256`
: The Camellia cipher by NTT. See `http://info.isl.ntt.co.jp/crypt/eng/camellia/specifications.html`.

`GCRY_CIPHER_SALSA20`
: This is the Salsa20 stream cipher.

`GCRY_CIPHER_SALSA20R12`
: This is the Salsa20/12 - reduced round version of Salsa20 stream cipher.

`GCRY_CIPHER_GOST28147`
: The GOST 28147-89 cipher, defined in the respective GOST standard. Translation of this GOST into English is provided in the RFC-5830.

`GCRY_CIPHER_GOST28147_MESH`
: The GOST 28147-89 cipher, defined in the respective GOST standard. Translation of this GOST into English is provided in the RFC-5830. This cipher will use CryptoPro keymeshing as defined in RFC 4357 if it has to be used for the selected parameter set.

`GCRY_CIPHER_CHACHA20`
: This is the ChaCha20 stream cipher.

`GCRY_CIPHER_SM4`
: A 128 bit cipher by the State Cryptography Administration of China (SCA). See `https://tools.ietf.org/html/draft-ribose-cfrg-sm4-10`.

5.2 Available cipher modes

`GCRY_CIPHER_MODE_NONE`
: No mode specified. This should not be used. The only exception is that if Libgcrypt is not used in FIPS mode and if any debug flag has been set, this mode may be used to bypass the actual encryption.

`GCRY_CIPHER_MODE_ECB`
: Electronic Codebook mode.

`GCRY_CIPHER_MODE_CFB`
`GCRY_CIPHER_MODE_CFB8`
: Cipher Feedback mode. For GCRY_CIPHER_MODE_CFB the shift size equals the block size of the cipher (e.g. for AES it is CFB-128). For GCRY_CIPHER_MODE_CFB8 the shift size is 8 bit but that variant is not yet available.

`GCRY_CIPHER_MODE_CBC`
: Cipher Block Chaining mode.

`GCRY_CIPHER_MODE_STREAM`
: Stream mode, only to be used with stream cipher algorithms.

`GCRY_CIPHER_MODE_OFB`
: Output Feedback mode.

`GCRY_CIPHER_MODE_CTR`
: Counter mode.

`GCRY_CIPHER_MODE_AESWRAP`
: This mode is used to implement the AES-Wrap algorithm according to RFC-3394. It may be used with any 128 bit block length algorithm, however the specs require one of the 3 AES algorithms. These special conditions apply: If `gcry_cipher_setiv` has not been used the standard IV is used; if it has been used the lower 64 bit of the IV are used as the Alternative Initial Value. On encryption the provided output buffer must be 64 bit (8 byte) larger than the input buffer; in-place encryption is still allowed. On decryption the output buffer may be specified 64 bit (8 byte) shorter than then input buffer. As per specs the input length must be at least 128 bits and the length must be a multiple of 64 bits.

`GCRY_CIPHER_MODE_CCM`
: Counter with CBC-MAC mode is an Authenticated Encryption with Associated Data (AEAD) block cipher mode, which is specified in 'NIST Special Publication 800-38C' and RFC 3610.

`GCRY_CIPHER_MODE_GCM`
: Galois/Counter Mode (GCM) is an Authenticated Encryption with Associated Data (AEAD) block cipher mode, which is specified in 'NIST Special Publication 800-38D'.

`GCRY_CIPHER_MODE_POLY1305`
: This mode implements the Poly1305 Authenticated Encryption with Associated Data (AEAD) mode according to RFC-8439. This mode can be used with ChaCha20 stream cipher.

`GCRY_CIPHER_MODE_OCB`
: OCB is an Authenticated Encryption with Associated Data (AEAD) block cipher mode, which is specified in RFC-7253. Supported tag lengths are 128, 96, and 64 bit with the default being 128 bit. To switch to a different tag length `gcry_cipher_ctl` using the command `GCRYCTL_SET_TAGLEN` and the address of an `int` variable set to 12 (for 96 bit) or 8 (for 64 bit) provided for the `buffer` argument and `sizeof(int)` for `buflen`.

  Note that the use of `gcry_cipher_final` is required.

`GCRY_CIPHER_MODE_XTS`
: XEX-based tweaked-codebook mode with ciphertext stealing (XTS) mode is used to implement the AES-XTS as specified in IEEE 1619 Standard Architecture for Encrypted Shared Storage Media and NIST SP800-38E.

  The XTS mode requires doubling key-length, for example, using 512-bit key with AES-256 (`GCRY_CIPHER_AES256`). The 128-bit tweak value is feed to XTS mode as little-endian byte array using `gcry_cipher_setiv` function. When encrypting or decrypting, full-sized data unit buffers needs to be passed to `gcry_cipher_encrypt` or `gcry_cipher_decrypt`. The tweak value is automatically incremented after each call of `gcry_cipher_encrypt` and `gcry_cipher_decrypt`. Auto-increment allows avoiding need of setting IV between processing of sequential data units.

`GCRY_CIPHER_MODE_EAX`
: EAX is an Authenticated Encryption with Associated Data (AEAD) block cipher mode by Bellare, Rogaway, and Wagner (see `http://web.cs.ucdavis.edu/~rogaway/papers/eax.html`).

5.3 Working with cipher handles

To use a cipher algorithm, you must first allocate an according handle. This is to be done using the open function:

`gcry_error_t gcry_cipher_open` (*gcry_cipher_hd_t* `*hd`, *int* `algo`, *int* `mode`, *unsigned int* `flags`) [Function]

This function creates the context handle required for most of the other cipher functions and returns a handle to it in 'hd'. In case of an error, an according error code is returned.

The ID of algorithm to use must be specified via *algo*. See Section 5.1 [Available ciphers], page 25, for a list of supported ciphers and the according constants.

Besides using the constants directly, the function `gcry_cipher_map_name` may be used to convert the textual name of an algorithm into the according numeric ID.

The cipher mode to use must be specified via *mode*. See Section 5.2 [Available cipher modes], page 27, for a list of supported cipher modes and the according constants.

Note that some modes are incompatible with some algorithms - in particular, stream mode (`GCRY_CIPHER_MODE_STREAM`) only works with stream ciphers. Poly1305 AEAD mode (`GCRY_CIPHER_MODE_POLY1305`) only works with ChaCha20 stream cipher. The block cipher modes (`GCRY_CIPHER_MODE_ECB`, `GCRY_CIPHER_MODE_CBC`, `GCRY_CIPHER_MODE_CFB`, `GCRY_CIPHER_MODE_OFB`, `GCRY_CIPHER_MODE_CTR` and `GCRY_CIPHER_MODE_EAX`) will work with any block cipher algorithm. GCM mode (`GCRY_CIPHER_MODE_GCM`), CCM mode (`GCRY_CIPHER_MODE_CCM`), OCB mode (`GCRY_CIPHER_MODE_OCB`), and XTS mode (`GCRY_CIPHER_MODE_XTS`) will only work with block cipher algorithms which have the block size of 16 bytes.

The third argument *flags* can either be passed as `0` or as the bit-wise OR of the following constants.

`GCRY_CIPHER_SECURE`
: Make sure that all operations are allocated in secure memory. This is useful when the key material is highly confidential.

`GCRY_CIPHER_ENABLE_SYNC`
: This flag enables the CFB sync mode, which is a special feature of Libgcrypt's CFB mode implementation to allow for OpenPGP's CFB variant. See `gcry_cipher_sync`.

`GCRY_CIPHER_CBC_CTS`
: Enable cipher text stealing (CTS) for the CBC mode. Cannot be used simultaneous as GCRY_CIPHER_CBC_MAC. CTS mode makes it possible to transform data of almost arbitrary size (only limitation is that it must be greater than the algorithm's block size).

`GCRY_CIPHER_CBC_MAC`
: Compute CBC-MAC keyed checksums. This is the same as CBC mode, but only output the last block. Cannot be used simultaneous as GCRY_CIPHER_CBC_CTS.

Use the following function to release an existing handle:

`void gcry_cipher_close` (*gcry_cipher_hd_t* `h`) [Function]

This function releases the context created by `gcry_cipher_open`. It also zeroises all sensitive information associated with this cipher handle.

In order to use a handle for performing cryptographic operations, a 'key' has to be set first:

`gcry_error_t gcry_cipher_setkey` (*gcry_cipher_hd_t* `h`, *const void* *`k`, *size_t* `l`) [Function]

Set the key *k* used for encryption or decryption in the context denoted by the handle *h*. The length *l* (in bytes) of the key *k* must match the required length of the algorithm set for this context or be in the allowed range for algorithms with variable key size. The function checks this and returns an error if there is a problem. A caller should always check for an error.

Most crypto modes requires an initialization vector (IV), which usually is a non-secret random string acting as a kind of salt value. The CTR mode requires a counter, which is also similar to a salt value. To set the IV or CTR, use these functions:

`gcry_error_t gcry_cipher_setiv` (*gcry_cipher_hd_t* `h`, *const void* *`k`, *size_t* `l`) [Function]

Set the initialization vector used for encryption or decryption. The vector is passed as the buffer *K* of length *l* bytes and copied to internal data structures. The function checks that the IV matches the requirement of the selected algorithm and mode.

This function is also used by AEAD modes and with Salsa20 and ChaCha20 stream ciphers to set or update the required nonce. In these cases it needs to be called after setting the key.

`gcry_error_t gcry_cipher_setctr` (*gcry_cipher_hd_t* `h`, *const void* *`c`, *size_t* `l`) [Function]

Set the counter vector used for encryption or decryption. The counter is passed as the buffer *c* of length *l* bytes and copied to internal data structures. The function checks that the counter matches the requirement of the selected algorithm (i.e., it must be the same size as the block size).

`gcry_error_t gcry_cipher_reset` (*gcry_cipher_hd_t* `h`) [Function]

Set the given handle's context back to the state it had after the last call to gcry_cipher_setkey and clear the initialization vector.

Note that gcry_cipher_reset is implemented as a macro.

Authenticated Encryption with Associated Data (AEAD) block cipher modes require the handling of the authentication tag and the additional authenticated data, which can be done by using the following functions:

`gcry_error_t gcry_cipher_authenticate` (*gcry_cipher_hd_t* `h`, *const void* *`abuf`, *size_t* `abuflen`) [Function]

Process the buffer *abuf* of length *abuflen* as the additional authenticated data (AAD) for AEAD cipher modes.

`gcry_error_t gcry_cipher_gettag` (*gcry_cipher_hd_t* `h`, *void* *`tag`, *size_t* `taglen`) [Function]

This function is used to read the authentication tag after encryption. The function finalizes and outputs the authentication tag to the buffer *tag* of length *taglen* bytes.

Depending on the used mode certain restrictions for *taglen* are enforced: For GCM *taglen* must be at least 16 or one of the allowed truncated lengths (4, 8, 12, 13, 14, or 15).

`gcry_error_t gcry_cipher_checktag` (*gcry_cipher_hd_t* `h`, *const void* *`tag`, *size_t* `taglen`) [Function]

Check the authentication tag after decryption. The authentication tag is passed as the buffer *tag* of length *taglen* bytes and compared to internal authentication tag computed during decryption. Error code `GPG_ERR_CHECKSUM` is returned if the authentication tag in the buffer *tag* does not match the authentication tag calculated during decryption.

Depending on the used mode certain restrictions for *taglen* are enforced: For GCM *taglen* must either be 16 or one of the allowed truncated lengths (4, 8, 12, 13, 14, or 15).

The actual encryption and decryption is done by using one of the following functions. They may be used as often as required to process all the data.

`gcry_error_t gcry_cipher_encrypt` (*gcry_cipher_hd_t* **h**, *unsigned char *out*, *size_t* **outsize**, *const unsigned char **in*, *size_t* **inlen**) [Function]

`gcry_cipher_encrypt` is used to encrypt the data. This function can either work in place or with two buffers. It uses the cipher context already setup and described by the handle *h*. There are 2 ways to use the function: If *in* is passed as `NULL` and *inlen* is `0`, in-place encryption of the data in *out* of length *outsize* takes place. With *in* being not `NULL`, *inlen* bytes are encrypted to the buffer *out* which must have at least a size of *inlen*. *outsize* must be set to the allocated size of *out*, so that the function can check that there is sufficient space. Note that overlapping buffers are not allowed.

Depending on the selected algorithms and encryption mode, the length of the buffers must be a multiple of the block size.

Some encryption modes require that `gcry_cipher_final` is used before the final data chunk is passed to this function.

The function returns `0` on success or an error code.

`gcry_error_t gcry_cipher_decrypt` (*gcry_cipher_hd_t* **h**, *unsigned char *out*, *size_t* **outsize**, *const unsigned char **in*, *size_t* **inlen**) [Function]

`gcry_cipher_decrypt` is used to decrypt the data. This function can either work in place or with two buffers. It uses the cipher context already setup and described by the handle *h*. There are 2 ways to use the function: If *in* is passed as `NULL` and *inlen* is `0`, in-place decryption of the data in *out* or length *outsize* takes place. With *in* being not `NULL`, *inlen* bytes are decrypted to the buffer *out* which must have at least a size of *inlen*. *outsize* must be set to the allocated size of *out*, so that the function can check that there is sufficient space. Note that overlapping buffers are not allowed.

Depending on the selected algorithms and encryption mode, the length of the buffers must be a multiple of the block size.

Some encryption modes require that `gcry_cipher_final` is used before the final data chunk is passed to this function.

The function returns `0` on success or an error code.

The OCB mode features integrated padding and must thus be told about the end of the input data. This is done with:

`gcry_error_t gcry_cipher_final` (*gcry_cipher_hd_t* **h**) [Function]

Set a flag in the context to tell the encrypt and decrypt functions that their next call will provide the last chunk of data. Only the first call to this function has an effect and only for modes which support it. Checking the error is in general not necessary. This is implemented as a macro.

OpenPGP (as defined in RFC-4880) requires a special sync operation in some places. The following function is used for this:

`gcry_error_t gcry_cipher_sync` (*gcry_cipher_hd_t* **h**) [Function]

Perform the OpenPGP sync operation on context *h*. Note that this is a no-op unless the context was created with the flag `GCRY_CIPHER_ENABLE_SYNC`

Some of the described functions are implemented as macros utilizing a catch-all control function. This control function is rarely used directly but there is nothing which would inhibit it:

`gcry_error_t gcry_cipher_ctl` (*gcry_cipher_hd_t* `h`, *int* `cmd`, *void* *`*buffer`*, *size_t* `buflen`) [Function]

: `gcry_cipher_ctl` controls various aspects of the cipher module and specific cipher contexts. Usually some more specialized functions or macros are used for this purpose. The semantics of the function and its parameters depends on the the command *cmd* and the passed context handle *h*. Please see the comments in the source code (`src/global.c`) for details.

`gcry_error_t gcry_cipher_info` (*gcry_cipher_hd_t* `h`, *int* `what`, *void* `*buffer`, *size_t* `*nbytes`) [Function]

: `gcry_cipher_info` is used to retrieve various information about a cipher context or the cipher module in general.

`GCRYCTL_GET_TAGLEN:`
: Return the length of the tag for an AE algorithm mode. An error is returned for modes which do not support a tag. *buffer* must be given as NULL. On success the result is stored *nbytes*. The taglen is returned in bytes.

5.4 General cipher functions

To work with the algorithms, several functions are available to map algorithm names to the internal identifiers, as well as ways to retrieve information about an algorithm or the current cipher context.

`gcry_error_t gcry_cipher_algo_info` (*int* `algo`, *int* `what`, *void* `*buffer`, *size_t* `*nbytes`) [Function]

: This function is used to retrieve information on a specific algorithm. You pass the cipher algorithm ID as *algo* and the type of information requested as *what*. The result is either returned as the return code of the function or copied to the provided *buffer* whose allocated length must be available in an integer variable with the address passed in *nbytes*. This variable will also receive the actual used length of the buffer.

Here is a list of supported codes for *what*:

`GCRYCTL_GET_KEYLEN:`
: Return the length of the key. If the algorithm supports multiple key lengths, the maximum supported value is returned. The length is returned as number of octets (bytes) and not as number of bits in *nbytes*; *buffer* must be zero. Note that it is usually better to use the convenience function `gcry_cipher_get_algo_keylen`.

`GCRYCTL_GET_BLKLEN:`
: Return the block length of the algorithm. The length is returned as a number of octets in *nbytes*; *buffer* must be zero. Note that it is usually better to use the convenience function `gcry_cipher_get_algo_blklen`.

`GCRYCTL_TEST_ALGO:`
: Returns `0` when the specified algorithm is available for use. *buffer* and *nbytes* must be zero.

`size_t gcry_cipher_get_algo_keylen` (*algo*) [Function]

This function returns length of the key for algorithm *algo*. If the algorithm supports multiple key lengths, the maximum supported key length is returned. On error `0` is returned. The key length is returned as number of octets.

This is a convenience functions which should be preferred over `gcry_cipher_algo_info` because it allows for proper type checking.

`size_t gcry_cipher_get_algo_blklen` (*int algo*) [Function]

This functions returns the block-length of the algorithm *algo* counted in octets. On error `0` is returned.

This is a convenience functions which should be preferred over `gcry_cipher_algo_info` because it allows for proper type checking.

`const char * gcry_cipher_algo_name` (*int algo*) [Function]

`gcry_cipher_algo_name` returns a string with the name of the cipher algorithm *algo*. If the algorithm is not known or another error occurred, the string `"?"` is returned. This function should not be used to test for the availability of an algorithm.

`int gcry_cipher_map_name` (*const char *name*) [Function]

`gcry_cipher_map_name` returns the algorithm identifier for the cipher algorithm described by the string *name*. If this algorithm is not available `0` is returned.

`int gcry_cipher_mode_from_oid` (*const char *string*) [Function]

Return the cipher mode associated with an ASN.1 object identifier. The object identifier is expected to be in the IETF-style dotted decimal notation. The function returns `0` for an unknown object identifier or when no mode is associated with it.

6 Public Key cryptography

Public key cryptography, also known as asymmetric cryptography, is an easy way for key management and to provide digital signatures. Libgcrypt provides two completely different interfaces to public key cryptography, this chapter explains the one based on S-expressions.

6.1 Available algorithms

Libgcrypt supports the RSA (Rivest-Shamir-Adleman) algorithms as well as DSA (Digital Signature Algorithm), Elgamal, ECDSA, ECDH, and EdDSA.

6.2 Used S-expressions

Libgcrypt's API for asymmetric cryptography is based on data structures called S-expressions (see `http://people.csail.mit.edu/rivest/sexp.html`) and does not work with contexts/handles as most of the other building blocks of Libgcrypt do.

The following information are stored in S-expressions:

- keys
- plain text data
- encrypted data
- signatures

To describe how Libgcrypt expect keys, we use examples. Note that words in italics indicate parameters whereas lowercase words are literals.

Note that all MPI (multi-precision-integers) values are expected to be in `GCRYMPI_FMT_USG` format. An easy way to create S-expressions is by using `gcry_sexp_build` which allows to pass a string with printf-like escapes to insert MPI values.

6.2.1 RSA key parameters

An RSA private key is described by this S-expression:

```
(private-key
  (rsa
    (n n-mpi)
    (e e-mpi)
    (d d-mpi)
    (p p-mpi)
    (q q-mpi)
    (u u-mpi)))
```

An RSA public key is described by this S-expression:

```
(public-key
  (rsa
    (n n-mpi)
    (e e-mpi)))
```

n-mpi RSA public modulus n.

e-mpi RSA public exponent e.

d-mpi RSA secret exponent $d = e^{-1} \bmod (p-1)(q-1)$.

p-mpi RSA secret prime p.

q-mpi RSA secret prime q with $p < q$.

u-mpi Multiplicative inverse $u = p^{-1} \bmod q$.

For signing and decryption the parameters (p, q, u) are optional but greatly improve the performance. Either all of these optional parameters must be given or none of them. They are mandatory for gcry_pk_testkey.

Note that OpenSSL uses slighly different parameters: $q < p$ and $u = q^{-1} \bmod p$. To use these parameters you will need to swap the values and recompute u. Here is example code to do this:

```
if (gcry_mpi_cmp (p, q) > 0)
  {
    gcry_mpi_swap (p, q);
    gcry_mpi_invm (u, p, q);
  }
```

6.2.2 DSA key parameters

A DSA private key is described by this S-expression:

```
(private-key
  (dsa
    (p p-mpi)
    (q q-mpi)
    (g g-mpi)
    (y y-mpi)
    (x x-mpi)))
```

p-mpi DSA prime p.

q-mpi DSA group order q (which is a prime divisor of $p-1$).

g-mpi DSA group generator g.

y-mpi DSA public key value $y = g^x \bmod p$.

x-mpi DSA secret exponent x.

The public key is similar with "private-key" replaced by "public-key" and no *x-mpi*.

6.2.3 ECC key parameters

An ECC private key is described by this S-expression:

```
(private-key
  (ecc
    (p p-mpi)
    (a a-mpi)
    (b b-mpi)
    (g g-point)
    (n n-mpi)
```

```
        (q q-point)
        (d d-mpi)))
```

p-mpi Prime specifying the field $GF(p)$.

a-mpi
b-mpi The two coefficients of the Weierstrass equation $y^2 = x^3 + ax + b$

g-point Base point g.

n-mpi Order of g

q-point The point representing the public key $Q = dG$.

d-mpi The private key d

All point values are encoded in standard format; Libgcrypt does in general only support uncompressed points, thus the first byte needs to be `0x04`. However "EdDSA" describes its own compression scheme which is used by default; the non-standard first byte `0x40` may optionally be used to explicit flag the use of the algorithm's native compression method.

The public key is similar with "private-key" replaced by "public-key" and no *d-mpi*.

If the domain parameters are well-known, the name of this curve may be used. For example

```
(private-key
  (ecc
    (curve "NIST P-192")
    (q q-point)
    (d d-mpi)))
```

Note that *q-point* is optional for a private key. The `curve` parameter may be given in any case and is used to replace missing parameters.

Currently implemented curves are:

`Curve25519`
`X25519`
`1.3.6.1.4.1.3029.1.5.1`
`1.3.101.110`
The RFC-8410 255 bit curve, its RFC name, OpenPGP and RFC OIDs.

`X448`
`1.3.101.111`
The RFC-8410 448 bit curve and its RFC OID.

`Ed25519`
`1.3.6.1.4.1.11591.15.1`
`1.3.101.112`
The signing variant of the RFC-8410 255 bit curve, its OpenPGP and RFC OIDs.

`Ed448`
`1.3.101.113`
The signing variant of the RFC-8410 448 bit curve and its RFC OID.

`NIST P-192`
`1.2.840.10045.3.1.1`
`nistp192`
`prime192v1`
`secp192r1`
: The NIST 192 bit curve, its OID and aliases.

`NIST P-224`
`1.3.132.0.33`
`nistp224`
`secp224r1`
: The NIST 224 bit curve, its OID and aliases.

`NIST P-256`
`1.2.840.10045.3.1.7`
`nistp256`
`prime256v1`
`secp256r1`
: The NIST 256 bit curve, its OID and aliases.

`NIST P-384`
`1.3.132.0.34`
`nistp384`
`secp384r1`
: The NIST 384 bit curve, its OID and aliases.

`NIST P-521`
`1.3.132.0.35`
`nistp521`
`secp521r1`
: The NIST 521 bit curve, its OID and aliases.

`brainpoolP160r1`
`1.3.36.3.3.2.8.1.1.1`
: The Brainpool 160 bit curve and its OID.

`brainpoolP192r1`
`1.3.36.3.3.2.8.1.1.3`
: The Brainpool 192 bit curve and its OID.

`brainpoolP224r1`
`1.3.36.3.3.2.8.1.1.5`
: The Brainpool 224 bit curve and its OID.

`brainpoolP256r1`
`1.3.36.3.3.2.8.1.1.7`
: The Brainpool 256 bit curve and its OID.

`brainpoolP320r1`
`1.3.36.3.3.2.8.1.1.9`
: The Brainpool 320 bit curve and its OID.

`brainpoolP384r1`
`1.3.36.3.3.2.8.1.1.11`
: The Brainpool 384 bit curve and its OID.

`brainpoolP512r1`
`1.3.36.3.3.2.8.1.1.13`
: The Brainpool 512 bit curve and its OID.

As usual the OIDs may optionally be prefixed with the string `OID.` or `oid.`.

6.3 Cryptographic Functions

Some functions operating on S-expressions support 'flags' to influence the operation. These flags have to be listed in a sub-S-expression named 'flags'. Flag names are case-sensitive. The following flags are known:

`comp`
`nocomp`
: If supported by the algorithm and curve the `comp` flag requests that points are returned in compact (compressed) representation. The `nocomp` flag requests that points are returned with full coordinates. The default depends on the the algorithm and curve. The compact representation requires a small overhead before a point can be used but halves the size of a to be conveyed public key. If `comp` is used with the "EdDSA" algorithm the key generation prefix the public key with a `0x40` byte.

`pkcs1`
: Use PKCS#1 block type 2 padding for encryption, block type 1 padding for signing.

`oaep`
: Use RSA-OAEP padding for encryption.

`pss`
: Use RSA-PSS padding for signing.

`eddsa`
: Use the EdDSA scheme signing instead of the default ECDSA algorithm. Note that the EdDSA uses a special form of the public key.

`rfc6979`
: For DSA and ECDSA use a deterministic scheme for the k parameter.

`no-blinding`
: Do not use a technique called 'blinding', which is used by default in order to prevent leaking of secret information. Blinding is only implemented by RSA, but it might be implemented by other algorithms in the future as well, when necessary.

`param`
: For ECC key generation also return the domain parameters. For ECC signing and verification override default parameters by provided domain parameters of the public or private key.

`transient-key`
: This flag is only meaningful for RSA, DSA, and ECC key generation. If given the key is created using a faster and a somewhat less secure random number generator. This flag may be used for keys which are only used for a short time or per-message and do not require full cryptographic strength.

`no-keytest`
: This flag skips internal failsafe tests to assert that a generated key is properly working. It currently has an effect only for standard ECC key generation. It is mostly useful along with transient-key to achieve fastest ECC key generation.

`use-x931`
: Force the use of the ANSI X9.31 key generation algorithm instead of the default algorithm. This flag is only meaningful for RSA key generation and usually not required. Note that this algorithm is implicitly used if either `derive-parms` is given or Libgcrypt is in FIPS mode.

`use-fips186`
: Force the use of the FIPS 186 key generation algorithm instead of the default algorithm. This flag is only meaningful for DSA and usually not required. Note that this algorithm is implicitly used if either `derive-parms` is given or Libgcrypt is in FIPS mode. As of now FIPS 186-2 is implemented; after the approval of FIPS 186-3 the code will be changed to implement 186-3.

`use-fips186-2`
: Force the use of the FIPS 186-2 key generation algorithm instead of the default algorithm. This algorithm is slightly different from FIPS 186-3 and allows only 1024 bit keys. This flag is only meaningful for DSA and only required for FIPS testing backward compatibility.

Now that we know the key basics, we can carry on and explain how to encrypt and decrypt data. In almost all cases the data is a random session key which is in turn used for the actual encryption of the real data. There are 2 functions to do this:

`gcry_error_t gcry_pk_encrypt` (*gcry_sexp_t* ***r_ciph***, *gcry_sexp_t* ***data***, *gcry_sexp_t* ***pkey***) [Function]

Obviously a public key must be provided for encryption. It is expected as an appropriate S-expression (see above) in *pkey*. The data to be encrypted can either be in the simple old format, which is a very simple S-expression consisting only of one MPI, or it may be a more complex S-expression which also allows to specify flags for operation, like e.g. padding rules.

If you don't want to let Libgcrypt handle the padding, you must pass an appropriate MPI using this expression for *data*:

```
(data
  (flags raw)
  (value mpi))
```

This has the same semantics as the old style MPI only way. *MPI* is the actual data, already padded appropriate for your protocol. Most RSA based systems however use PKCS#1 padding and so you can use this S-expression for *data*:

```
(data
  (flags pkcs1)
  (value block))
```

Here, the "flags" list has the "pkcs1" flag which let the function know that it should provide PKCS#1 block type 2 padding. The actual data to be encrypted is passed as a string of octets in *block*. The function checks that this data actually can be used with the given key, does the padding and encrypts it.

If the function could successfully perform the encryption, the return value will be 0 and a new S-expression with the encrypted result is allocated and assigned to the variable at the address of *r_ciph*. The caller is responsible to release this value using `gcry_sexp_release`. In case of an error, an error code is returned and *r_ciph* will be set to `NULL`.

The returned S-expression has this format when used with RSA:

```
(enc-val
  (rsa
    (a a-mpi)))
```

Where *a-mpi* is an MPI with the result of the RSA operation. When using the Elgamal algorithm, the return value will have this format:

```
(enc-val
  (elg
    (a a-mpi)
    (b b-mpi)))
```

Where *a-mpi* and *b-mpi* are MPIs with the result of the Elgamal encryption operation.

`gcry_error_t gcry_pk_decrypt` (*gcry_sexp_t* *`*r_plain`*, *gcry_sexp_t* `data`, *gcry_sexp_t* `skey`) [Function]

Obviously a private key must be provided for decryption. It is expected as an appropriate S-expression (see above) in *skey*. The data to be decrypted must match the format of the result as returned by `gcry_pk_encrypt`, but should be enlarged with a `flags` element:

```
(enc-val
  (flags)
  (elg
    (a a-mpi)
    (b b-mpi)))
```

This function does not remove padding from the data by default. To let Libgcrypt remove padding, give a hint in 'flags' telling which padding method was used when encrypting:

```
(flags padding-method)
```

Currently *padding-method* is either `pkcs1` for PKCS#1 block type 2 padding, or `oaep` for RSA-OAEP padding.

The function returns 0 on success or an error code. The variable at the address of *r_plain* will be set to NULL on error or receive the decrypted value on success. The format of *r_plain* is a simple S-expression part (i.e. not a valid one) with just one MPI if there was no `flags` element in *data*; if at least an empty `flags` is passed in *data*, the format is:

```
(value plaintext)
```

Another operation commonly performed using public key cryptography is signing data. In some sense this is even more important than encryption because digital signatures are an important instrument for key management. Libgcrypt supports digital signatures using 2 functions, similar to the encryption functions:

gcry_error_t gcry_pk_sign (*gcry_sexp_t *r_sig, gcry_sexp_t data, gcry_sexp_t skey*) [Function]

This function creates a digital signature for *data* using the private key *skey* and place it into the variable at the address of *r_sig*. *data* may either be the simple old style S-expression with just one MPI or a modern and more versatile S-expression which allows to let Libgcrypt handle padding:

```
 (data
  (flags pkcs1)
  (hash hash-algo block))
```

This example requests to sign the data in *block* after applying PKCS#1 block type 1 style padding. *hash-algo* is a string with the hash algorithm to be encoded into the signature, this may be any hash algorithm name as supported by Libgcrypt. Most likely, this will be "sha256" or "sha1". It is obvious that the length of *block* must match the size of that message digests; the function checks that this and other constraints are valid.

If PKCS#1 padding is not required (because the caller does already provide a padded value), either the old format or better the following format should be used:

```
(data
  (flags raw)
  (value mpi))
```

Here, the data to be signed is directly given as an *MPI*.

For DSA the input data is expected in this format:

```
(data
  (flags raw)
  (value mpi))
```

Here, the data to be signed is directly given as an *MPI*. It is expect that this MPI is the the hash value. For the standard DSA using a MPI is not a problem in regard to leading zeroes because the hash value is directly used as an MPI. For better standard conformance it would be better to explicit use a memory string (like with pkcs1) but that is currently not supported. However, for deterministic DSA as specified in RFC6979 this can't be used. Instead the following input is expected.

```
(data
  (flags rfc6979)
  (hash hash-algo block))
```

Note that the provided hash-algo is used for the internal HMAC; it should match the hash-algo used to create *block*.

The signature is returned as a newly allocated S-expression in *r_sig* using this format for RSA:

```
(sig-val
  (rsa
    (s s-mpi)))
```

Where *s-mpi* is the result of the RSA sign operation. For DSA the S-expression returned is:

```
(sig-val
```

```
(dsa
  (r r-mpi)
  (s s-mpi)))
```

Where *r-mpi* and *s-mpi* are the result of the DSA sign operation.

For Elgamal signing (which is slow, yields large numbers and probably is not as secure as the other algorithms), the same format is used with "elg" replacing "dsa"; for ECDSA signing, the same format is used with "ecdsa" replacing "dsa".

For the EdDSA algorithm (cf. Ed25515) the required input parameters are:

```
(data
  (flags eddsa)
  (hash-algo sha512)
  (value message))
```

Note that the *message* may be of any length; hashing is part of the algorithm. Using a large data block for *message* is in general not suggested; in that case the used protocol should better require that a hash of the message is used as input to the EdDSA algorithm. Note that for X.509 certificates *message* is the `tbsCertificate` part and in CMS *message* is the `signedAttrs` part; see RFC-8410 and RFC-8419.

The operation most commonly used is definitely the verification of a signature. Libgcrypt provides this function:

`gcry_error_t gcry_pk_verify` (*gcry_sexp_t* `sig`, *gcry_sexp_t* `data`, *gcry_sexp_t* `pkey`) [Function]

This is used to check whether the signature *sig* matches the *data*. The public key *pkey* must be provided to perform this verification. This function is similar in its parameters to `gcry_pk_sign` with the exceptions that the public key is used instead of the private key and that no signature is created but a signature, in a format as created by `gcry_pk_sign`, is passed to the function in *sig*.

The result is 0 for success (i.e. the data matches the signature), or an error code where the most relevant code is `GCRY_ERR_BAD_SIGNATURE` to indicate that the signature does not match the provided data.

6.4 Dedicated functions for elliptic curves.

The S-expression based interface is for certain operations on elliptic curves not optimal. Thus a few special functions are implemented to support common operations on curves with one of these assigned curve ids:

`GCRY_ECC_CURVE25519`
`GCRY_ECC_CURVE448`

`unsigned int gcry_ecc_get_algo_keylen` (*int* `curveid`); [Function]

Returns the length in bytes of a point on the curve with the id *curveid*. 0 is returned for curves which have no assigned id.

`gpg_error_t gcry_ecc_mul_point` (*int* `curveid`, *unsigned char* *`result`, *const unsigned char* *`scalar`, *const unsigned char* *`point`) [Function]

This function computes the scalar multiplication on the Montgomery form of the curve with id *curveid*. If *point* is NULL the base point of the curve is used. The caller needs to provide a large enough buffer for *result* and a valid *scalar* and *point*.

6.5 General public-key related Functions

A couple of utility functions are available to retrieve the length of the key, map algorithm identifiers and perform sanity checks:

`const char * gcry_pk_algo_name` (*int* `algo`) [Function]

Map the public key algorithm id *algo* to a string representation of the algorithm name. For unknown algorithms this functions returns the string `"?"`. This function should not be used to test for the availability of an algorithm.

`int gcry_pk_map_name` (*const char* *`name`) [Function]

Map the algorithm *name* to a public key algorithm Id. Returns 0 if the algorithm name is not known.

`int gcry_pk_test_algo` (*int* `algo`) [Function]

Return 0 if the public key algorithm *algo* is available for use. Note that this is implemented as a macro.

`unsigned int gcry_pk_get_nbits` (*gcry_sexp_t* `key`) [Function]

Return what is commonly referred as the key length for the given public or private in *key*.

`unsigned char * gcry_pk_get_keygrip` (*gcry_sexp_t* `key`, *unsigned char* *`array`) [Function]

Return the so called "keygrip" which is the SHA-1 hash of the public key parameters expressed in a way depended on the algorithm. *array* must either provide space for 20 bytes or be `NULL`. In the latter case a newly allocated array of that size is returned. On success a pointer to the newly allocated space or to *array* is returned. `NULL` is returned to indicate an error which is most likely an unknown algorithm or one where a "keygrip" has not yet been defined. The function accepts public or secret keys in *key*.

`gcry_error_t gcry_pk_testkey` (*gcry_sexp_t* `key`) [Function]

Return zero if the private key *key* is 'sane', an error code otherwise. Note that it is not possible to check the 'saneness' of a public key.

`gcry_error_t gcry_pk_algo_info` (*int* `algo`, *int* `what`, *void* *`buffer`, *size_t* *`nbytes`) [Function]

Depending on the value of *what* return various information about the public key algorithm with the id *algo*. Note that the function returns `-1` on error and the actual error code must be retrieved using the function `gcry_errno`. The currently defined values for *what* are:

`GCRYCTL_TEST_ALGO:`
: Return 0 if the specified algorithm is available for use. *buffer* must be `NULL`, *nbytes* may be passed as `NULL` or point to a variable with the required usage of the algorithm. This may be 0 for "don't care" or the bit-wise OR of these flags:

  `GCRY_PK_USAGE_SIGN`
  : Algorithm is usable for signing.

  `GCRY_PK_USAGE_ENCR`
  : Algorithm is usable for encryption.

  Unless you need to test for the allowed usage, it is in general better to use the macro gcry_pk_test_algo instead.

`GCRYCTL_GET_ALGO_USAGE:`
: Return the usage flags for the given algorithm. An invalid algorithm return 0. Disabled algorithms are ignored here because we want to know whether the algorithm is at all capable of a certain usage.

`GCRYCTL_GET_ALGO_NPKEY`
: Return the number of elements the public key for algorithm *algo* consist of. Return 0 for an unknown algorithm.

`GCRYCTL_GET_ALGO_NSKEY`
: Return the number of elements the private key for algorithm *algo* consist of. Note that this value is always larger than that of the public key. Return 0 for an unknown algorithm.

`GCRYCTL_GET_ALGO_NSIGN`
: Return the number of elements a signature created with the algorithm *algo* consists of. Return 0 for an unknown algorithm or for an algorithm not capable of creating signatures.

`GCRYCTL_GET_ALGO_NENCR`
: Return the number of elements a encrypted message created with the algorithm *algo* consists of. Return 0 for an unknown algorithm or for an algorithm not capable of encryption.

Please note that parameters not required should be passed as `NULL`.

`gcry_error_t gcry_pk_ctl` (*int* **`cmd`**, *void ****`buffer`**, *size_t* **`buflen`**) [Function]

This is a general purpose function to perform certain control operations. *cmd* controls what is to be done. The return value is 0 for success or an error code. Currently supported values for *cmd* are:

`GCRYCTL_DISABLE_ALGO`
: Disable the algorithm given as an algorithm id in *buffer*. *buffer* must point to an `int` variable with the algorithm id and *buflen* must have the value `sizeof (int)`. This function is not thread safe and should thus be used before any other threads are started.

Libgcrypt also provides a function to generate public key pairs:

`gcry_error_t gcry_pk_genkey` (*gcry_sexp_t* `*r_key`, *gcry_sexp_t* `parms`) [Function]

This function create a new public key pair using information given in the S-expression *parms* and stores the private and the public key in one new S-expression at the address given by *r_key*. In case of an error, *r_key* is set to `NULL`. The return code is 0 for success or an error code otherwise.

Here is an example for *parms* to create an 2048 bit RSA key:

```
(genkey
  (rsa
    (nbits 4:2048)))
```

To create an Elgamal key, substitute "elg" for "rsa" and to create a DSA key use "dsa". Valid ranges for the key length depend on the algorithms; all commonly used key lengths are supported. Currently supported parameters are:

`nbits` This is always required to specify the length of the key. The argument is a string with a number in C-notation. The value should be a multiple of 8. Note that the S-expression syntax requires that a number is prefixed with its string length; thus the `4:` in the above example.

`curve` *name*
For ECC a named curve may be used instead of giving the number of requested bits. This allows to request a specific curve to override a default selection Libgcrypt would have taken if `nbits` has been given. The available names are listed with the description of the ECC public key parameters.

`rsa-use-e` *value*
This is only used with RSA to give a hint for the public exponent. The *value* will be used as a base to test for a usable exponent. Some values are special:

‘0’ Use a secure and fast value. This is currently the number 41.

‘1’ Use a value as required by some crypto policies. This is currently the number 65537.

‘2’ Reserved

‘> 2’ Use the given value.

If this parameter is not used, Libgcrypt uses for historic reasons 65537. Note that the value must fit into a 32 bit unsigned variable and that the usual C prefixes are considered (e.g. 017 gives 15).

`qbits` *n* This is only meanigful for DSA keys. If it is given the DSA key is generated with a Q parameyer of size *n* bits. If it is not given or zero Q is deduced from NBITS in this way:

‘512 <= N <= 1024’
Q = 160

‘N = 2048’ Q = 224

'N = 3072' Q = 256

'N = 7680' Q = 384

'N = 15360'
Q = 512

Note that in this case only the values for N, as given in the table, are allowed. When specifying Q all values of N in the range 512 to 15680 are valid as long as they are multiples of 8.

`domain` *list*
This is only meaningful for DLP algorithms. If specified keys are generated with domain parameters taken from this list. The exact format of this parameter depends on the actual algorithm. It is currently only implemented for DSA using this format:

```
(genkey
  (dsa
    (domain
      (p p-mpi)
      (q q-mpi)
      (g q-mpi))))
```

`nbits` and `qbits` may not be specified because they are derived from the domain parameters.

`derive-parms` *list*
This is currently only implemented for RSA and DSA keys. It is not allowed to use this together with a `domain` specification. If given, it is used to derive the keys using the given parameters.

If given for an RSA key the X9.31 key generation algorithm is used even if libgcrypt is not in FIPS mode. If given for a DSA key, the FIPS 186 algorithm is used even if libgcrypt is not in FIPS mode.

```
(genkey
  (rsa
    (nbits 4:1024)
    (rsa-use-e 1:3)
    (derive-parms
      (Xp1 #1A1916DDB29B4EB7EB6732E128#)
      (Xp2 #192E8AAC41C576C822D93EA433#)
      (Xp  #D8CD81F035EC57EFE822955149D3BFF70C53520D
            769D6D76646C7A792E16EBD89FE6FC5B605A6493
            39DFC925A86A4C6D150B71B9EEA02D68885F5009
            B98BD984#)
      (Xq1 #1A5CF72EE770DE50CB09ACCEA9#)
      (Xq2 #134E4CAA16D2350A21D775C404#)
      (Xq  #CC1092495D867E64065DEE3E7955F2EBC7D47A2D
            7C9953388F97DDDC3E1CA19C35CA659EDC2FC325
            6D29C2627479C086A699A49C4C9CEE7EF7BD1B34
```

```
                    321DE34A#))))
          (genkey
            (dsa
              (nbits 4:1024)
              (derive-parms
                (seed seed-mpi))))
```

`flags` *flaglist*

This is preferred way to define flags. *flaglist* may contain any number of flags. See above for a specification of these flags.

Here is an example on how to create a key using curve Ed25519 with the ECDSA signature algorithm. Note that the use of ECDSA with that curve is in general not recommended.

```
(genkey
  (ecc
    (flags transient-key)))
```

`transient-key`
`use-x931`
`use-fips186`
`use-fips186-2`

These are deprecated ways to set a flag with that name; see above for a description of each flag.

The key pair is returned in a format depending on the algorithm. Both private and public keys are returned in one container and may be accompanied by some miscellaneous information.

Here are two examples; the first for Elgamal and the second for elliptic curve key generation:

```
(key-data
  (public-key
    (elg
      (p p-mpi)
      (g g-mpi)
      (y y-mpi)))
  (private-key
    (elg
      (p p-mpi)
      (g g-mpi)
      (y y-mpi)
      (x x-mpi)))
  (misc-key-info
    (pm1-factors n1 n2 ... nn))
(key-data
  (public-key
    (ecc
      (curve Ed25519)
```

```
          (flags eddsa)
          (q q-value)))
      (private-key
        (ecc
          (curve Ed25519)
          (flags eddsa)
          (q q-value)
          (d d-value))))
```

As you can see, some of the information is duplicated, but this provides an easy way to extract either the public or the private key. Note that the order of the elements is not defined, e.g. the private key may be stored before the public key. *n1 n2 ... nn* is a list of prime numbers used to composite *p-mpi*; this is in general not a very useful information and only available if the key generation algorithm provides them.

Future versions of Libgcrypt will have extended versions of the public key interface which will take an additional context to allow for pre-computations, special operations, and other optimization. As a first step a new function is introduced to help using the ECC algorithms in new ways:

`gcry_error_t gcry_pubkey_get_sexp` (*gcry_sexp_t* `*r_sexp`, *int* `mode`, *gcry_ctx_t* `ctx`) [Function]

Return an S-expression representing the context *ctx*. Depending on the state of that context, the S-expression may either be a public key, a private key or any other object used with public key operations. On success 0 is returned and a new S-expression is stored at *r_sexp*; on error an error code is returned and NULL is stored at *r_sexp*. *mode* must be one of:

`0`
: Decide what to return depending on the context. For example if the private key parameter is available a private key is returned, if not a public key is returned.

`GCRY_PK_GET_PUBKEY`
: Return the public key even if the context has the private key parameter.

`GCRY_PK_GET_SECKEY`
: Return the private key or the error `GPG_ERR_NO_SECKEY` if it is not possible.

As of now this function supports only certain ECC operations because a context object is right now only defined for ECC. Over time this function will be extended to cover more algorithms.

7 Hashing

Libgcrypt provides an easy and consistent to use interface for hashing. Hashing is buffered and several hash algorithms can be updated at once. It is possible to compute a HMAC using the same routines. The programming model follows an open/process/close paradigm and is in that similar to other building blocks provided by Libgcrypt.

For convenience reasons, a few cyclic redundancy check value operations are also supported.

7.1 Available hash algorithms

`GCRY_MD_NONE`
: This is not a real algorithm but used by some functions as an error return value. This constant is guaranteed to have the value `0`.

`GCRY_MD_SHA1`
: This is the SHA-1 algorithm which yields a message digest of 20 bytes. Note that SHA-1 begins to show some weaknesses and it is suggested to fade out its use if strong cryptographic properties are required.

`GCRY_MD_RMD160`
: This is the 160 bit version of the RIPE message digest (RIPE-MD-160). Like SHA-1 it also yields a digest of 20 bytes. This algorithm share a lot of design properties with SHA-1 and thus it is advisable not to use it for new protocols.

`GCRY_MD_MD5`
: This is the well known MD5 algorithm, which yields a message digest of 16 bytes. Note that the MD5 algorithm has severe weaknesses, for example it is easy to compute two messages yielding the same hash (collision attack). The use of this algorithm is only justified for non-cryptographic application.

`GCRY_MD_MD4`
: This is the MD4 algorithm, which yields a message digest of 16 bytes. This algorithm has severe weaknesses and should not be used.

`GCRY_MD_MD2`
: This is an reserved identifier for MD-2; there is no implementation yet. This algorithm has severe weaknesses and should not be used.

`GCRY_MD_TIGER`
: This is the TIGER/192 algorithm which yields a message digest of 24 bytes. Actually this is a variant of TIGER with a different output print order as used by GnuPG up to version 1.3.2.

`GCRY_MD_TIGER1`
: This is the TIGER variant as used by the NESSIE project. It uses the most commonly used output print order.

`GCRY_MD_TIGER2`
: This is another variant of TIGER with a different padding scheme.

GCRY_MD_HAVAL
: This is an reserved value for the HAVAL algorithm with 5 passes and 160 bit. It yields a message digest of 20 bytes. Note that there is no implementation yet available.

GCRY_MD_SHA224
: This is the SHA-224 algorithm which yields a message digest of 28 bytes. See Change Notice 1 for FIPS 180-2 for the specification.

GCRY_MD_SHA256
: This is the SHA-256 algorithm which yields a message digest of 32 bytes. See FIPS 180-2 for the specification.

GCRY_MD_SHA384
: This is the SHA-384 algorithm which yields a message digest of 48 bytes. See FIPS 180-2 for the specification.

GCRY_MD_SHA512
: This is the SHA-512 algorithm which yields a message digest of 64 bytes. See FIPS 180-2 for the specification.

GCRY_MD_SHA512_224
: This is the SHA-512/224 algorithm which yields a message digest of 28 bytes. See FIPS 180-4 for the specification.

GCRY_MD_SHA512_256
: This is the SHA-512/256 algorithm which yields a message digest of 32 bytes. See FIPS 180-4 for the specification.

GCRY_MD_SHA3_224
: This is the SHA3-224 algorithm which yields a message digest of 28 bytes. See FIPS 202 for the specification.

GCRY_MD_SHA3_256
: This is the SHA3-256 algorithm which yields a message digest of 32 bytes. See FIPS 202 for the specification.

GCRY_MD_SHA3_384
: This is the SHA3-384 algorithm which yields a message digest of 48 bytes. See FIPS 202 for the specification.

GCRY_MD_SHA3_512
: This is the SHA3-512 algorithm which yields a message digest of 64 bytes. See FIPS 202 for the specification.

GCRY_MD_SHAKE128
: This is the SHAKE128 extendable-output function (XOF) algorithm with 128 bit security strength. See FIPS 202 for the specification.

GCRY_MD_SHAKE256
: This is the SHAKE256 extendable-output function (XOF) algorithm with 256 bit security strength. See FIPS 202 for the specification.

`GCRY_MD_CRC32`
: This is the ISO 3309 and ITU-T V.42 cyclic redundancy check. It yields an output of 4 bytes. Note that this is not a hash algorithm in the cryptographic sense.

`GCRY_MD_CRC32_RFC1510`
: This is the above cyclic redundancy check function, as modified by RFC 1510. It yields an output of 4 bytes. Note that this is not a hash algorithm in the cryptographic sense.

`GCRY_MD_CRC24_RFC2440`
: This is the OpenPGP cyclic redundancy check function. It yields an output of 3 bytes. Note that this is not a hash algorithm in the cryptographic sense.

`GCRY_MD_WHIRLPOOL`
: This is the Whirlpool algorithm which yields a message digest of 64 bytes.

`GCRY_MD_GOSTR3411_94`
: This is the hash algorithm described in GOST R 34.11-94 which yields a message digest of 32 bytes.

`GCRY_MD_STRIBOG256`
: This is the 256-bit version of hash algorithm described in GOST R 34.11-2012 which yields a message digest of 32 bytes.

`GCRY_MD_STRIBOG512`
: This is the 512-bit version of hash algorithm described in GOST R 34.11-2012 which yields a message digest of 64 bytes.

`GCRY_MD_BLAKE2B_512`
: This is the BLAKE2b-512 algorithm which yields a message digest of 64 bytes. See RFC 7693 for the specification.

`GCRY_MD_BLAKE2B_384`
: This is the BLAKE2b-384 algorithm which yields a message digest of 48 bytes. See RFC 7693 for the specification.

`GCRY_MD_BLAKE2B_256`
: This is the BLAKE2b-256 algorithm which yields a message digest of 32 bytes. See RFC 7693 for the specification.

`GCRY_MD_BLAKE2B_160`
: This is the BLAKE2b-160 algorithm which yields a message digest of 20 bytes. See RFC 7693 for the specification.

`GCRY_MD_BLAKE2S_256`
: This is the BLAKE2s-256 algorithm which yields a message digest of 32 bytes. See RFC 7693 for the specification.

`GCRY_MD_BLAKE2S_224`
: This is the BLAKE2s-224 algorithm which yields a message digest of 28 bytes. See RFC 7693 for the specification.

`GCRY_MD_BLAKE2S_160`
: This is the BLAKE2s-160 algorithm which yields a message digest of 20 bytes. See RFC 7693 for the specification.

`GCRY_MD_BLAKE2S_128`
: This is the BLAKE2s-128 algorithm which yields a message digest of 16 bytes. See RFC 7693 for the specification.

`GCRY_MD_SM3`
: This is the SM3 algorithm which yields a message digest of 32 bytes.

7.2 Working with hash algorithms

To use most of these function it is necessary to create a context; this is done using:

gcry_error_t gcry_md_open (*gcry_md_hd_t *`hd`, int `algo`, unsigned int `flags`*) [Function]

Create a message digest object for algorithm *algo*. *flags* may be given as an bitwise OR of constants described below. *algo* may be given as `0` if the algorithms to use are later set using `gcry_md_enable`. *hd* is guaranteed to either receive a valid handle or NULL.

For a list of supported algorithms, see Section 7.1 [Available hash algorithms], page 51.

The flags allowed for *mode* are:

`GCRY_MD_FLAG_SECURE`
: Allocate all buffers and the resulting digest in "secure memory". Use this is the hashed data is highly confidential.

`GCRY_MD_FLAG_HMAC`
: Turn the algorithm into a HMAC message authentication algorithm. This only works if just one algorithm is enabled for the handle and that algorithm is not an extendable-output function. Note that the function `gcry_md_setkey` must be used to set the MAC key. The size of the MAC is equal to the message digest of the underlying hash algorithm. If you want CBC message authentication codes based on a cipher, see Section 5.3 [Working with cipher handles], page 28.

`GCRY_MD_FLAG_BUGEMU1`
: Versions of Libgcrypt before 1.6.0 had a bug in the Whirlpool code which led to a wrong result for certain input sizes and write patterns. Using this flag emulates that bug. This may for example be useful for applications which use Whirlpool as part of their key generation. It is strongly suggested to use this flag only if really needed and if possible to the data should be re-processed using the regular Whirlpool algorithm.

  Note that this flag works for the entire hash context. If needed arises it may be used to enable bug emulation for other hash algorithms. Thus you should not use this flag for a multi-algorithm hash context.

You may use the function `gcry_md_is_enabled` to later check whether an algorithm has been enabled.

If you want to calculate several hash algorithms at the same time, you have to use the following function right after the `gcry_md_open`:

`gcry_error_t gcry_md_enable` (*gcry_md_hd_t* `h`, *int* `algo`) [Function]

Add the message digest algorithm *algo* to the digest object described by handle *h*. Duplicated enabling of algorithms is detected and ignored.

If the flag `GCRY_MD_FLAG_HMAC` was used, the key for the MAC must be set using the function:

`gcry_error_t gcry_md_setkey` (*gcry_md_hd_t* `h`, *const void* *`key`, *size_t* `keylen`) [Function]

For use with the HMAC feature or BLAKE2 keyed hash, set the MAC key to the value of *key* of length *keylen* bytes. For HMAC, there is no restriction on the length of the key. For keyed BLAKE2b hash, length of the key must be in the range 1 to 64 bytes. For keyed BLAKE2s hash, length of the key must be in the range 1 to 32 bytes.

After you are done with the hash calculation, you should release the resources by using:

`void gcry_md_close` (*gcry_md_hd_t* `h`) [Function]

Release all resources of hash context *h*. *h* should not be used after a call to this function. A `NULL` passed as *h* is ignored. The function also zeroises all sensitive information associated with this handle.

Often you have to do several hash operations using the same algorithm. To avoid the overhead of creating and releasing context, a reset function is provided:

`void gcry_md_reset` (*gcry_md_hd_t* `h`) [Function]

Reset the current context to its initial state. This is effectively identical to a close followed by an open and enabling all currently active algorithms.

Often it is necessary to start hashing some data and then continue to hash different data. To avoid hashing the same data several times (which might not even be possible if the data is received from a pipe), a snapshot of the current hash context can be taken and turned into a new context:

`gcry_error_t gcry_md_copy` (*gcry_md_hd_t* *`handle_dst`, *gcry_md_hd_t* `handle_src`) [Function]

Create a new digest object as an exact copy of the object described by handle *handle_src* and store it in *handle_dst*. The context is not reset and you can continue to hash data using this context and independently using the original context.

Now that we have prepared everything to calculate hashes, it is time to see how it is actually done. There are two ways for this, one to update the hash with a block of memory and one macro to update the hash by just one character. Both methods can be used on the same hash context.

`void gcry_md_write (gcry_md_hd_t h, const void *buffer, size_t length)` [Function]

Pass *length* bytes of the data in *buffer* to the digest object with handle *h* to update the digest values. This function should be used for large blocks of data. If this function is used after the context has been finalized, it will keep on pushing the data through the algorithm specific transform function and change the context; however the results are not meaningful and this feature is only available to mitigate timing attacks.

`void gcry_md_putc (gcry_md_hd_t h, int c)` [Function]

Pass the byte in *c* to the digest object with handle *h* to update the digest value. This is an efficient function, implemented as a macro to buffer the data before an actual update.

The semantics of the hash functions do not provide for reading out intermediate message digests because the calculation must be finalized first. This finalization may for example include the number of bytes hashed in the message digest or some padding.

`void gcry_md_final (gcry_md_hd_t h)` [Function]

Finalize the message digest calculation. This is not really needed because `gcry_md_read` and `gcry_md_extract` do this implicitly. After this has been done no further updates (by means of `gcry_md_write` or `gcry_md_putc` should be done; However, to mitigate timing attacks it is sometimes useful to keep on updating the context after having stored away the actual digest. Only the first call to this function has an effect. It is implemented as a macro.

The way to read out the calculated message digest is by using the function:

`unsigned char * gcry_md_read (gcry_md_hd_t h, int algo)` [Function]

`gcry_md_read` returns the message digest after finalizing the calculation. This function may be used as often as required but it will always return the same value for one handle. The returned message digest is allocated within the message context and therefore valid until the handle is released or reset-ed (using `gcry_md_close` or `gcry_md_reset` or it has been updated as a mitigation measure against timing attacks. *algo* may be given as 0 to return the only enabled message digest or it may specify one of the enabled algorithms. The function does return `NULL` if the requested algorithm has not been enabled.

The way to read output of extendable-output function is by using the function:

`gpg_err_code_t gcry_md_extract (gcry_md_hd_t h, int algo, void *buffer, size_t length)` [Function]

`gcry_mac_read` returns output from extendable-output function. This function may be used as often as required to generate more output byte stream from the algorithm. Function extracts the new output bytes to *buffer* of the length *length*. Buffer will be fully populated with new output. *algo* may be given as 0 to return the only enabled message digest or it may specify one of the enabled algorithms. The function does return non-zero value if the requested algorithm has not been enabled.

Because it is often necessary to get the message digest of blocks of memory, two fast convenience function are available for this task:

`gpg_err_code_t gcry_md_hash_buffers ( int algo, unsigned int flags, void *digest, const gcry_buffer_t *iov, int iovcnt )` [Function]

`gcry_md_hash_buffers` is a shortcut function to calculate a message digest from several buffers. This function does not require a context and immediately returns the message digest of the data described by *iov* and *iovcnt*. *digest* must be allocated by the caller, large enough to hold the message digest yielded by the the specified algorithm *algo*. This required size may be obtained by using the function `gcry_md_get_algo_dlen`.

iov is an array of buffer descriptions with *iovcnt* items. The caller should zero out the structures in this array and for each array item set the fields `.data` to the address of the data to be hashed, `.len` to number of bytes to be hashed. If *.off* is also set, the data is taken starting at *.off* bytes from the begin of the buffer. The field `.size` is not used.

The only supported flag value for *flags* is *GCRY_MD_FLAG_HMAC* which turns this function into a HMAC function; the first item in *iov* is then used as the key.

On success the function returns 0 and stores the resulting hash or MAC at *digest*.

`void gcry_md_hash_buffer (int algo, void *digest, const void *buffer, size_t length);` [Function]

`gcry_md_hash_buffer` is a shortcut function to calculate a message digest of a buffer. This function does not require a context and immediately returns the message digest of the *length* bytes at *buffer*. *digest* must be allocated by the caller, large enough to hold the message digest yielded by the the specified algorithm *algo*. This required size may be obtained by using the function `gcry_md_get_algo_dlen`.

Note that in contrast to `gcry_md_hash_buffers` this function will abort the process if an unavailable algorithm is used.

Hash algorithms are identified by internal algorithm numbers (see `gcry_md_open` for a list). However, in most applications they are used by names, so two functions are available to map between string representations and hash algorithm identifiers.

`const char * gcry_md_algo_name (int algo)` [Function]

Map the digest algorithm id *algo* to a string representation of the algorithm name. For unknown algorithms this function returns the string `"?"`. This function should not be used to test for the availability of an algorithm.

`int gcry_md_map_name (const char *name)` [Function]

Map the algorithm with *name* to a digest algorithm identifier. Returns 0 if the algorithm name is not known. Names representing ASN.1 object identifiers are recognized if the IETF dotted format is used and the OID is prefixed with either `"oid."` or `"OID."`. For a list of supported OIDs, see the source code at `cipher/md.c`. This function should not be used to test for the availability of an algorithm.

`gcry_error_t gcry_md_get_asnoid (int algo, void *buffer, size_t *length)` [Function]

Return an DER encoded ASN.1 OID for the algorithm *algo* in the user allocated *buffer*. *length* must point to variable with the available size of *buffer* and receives after return the actual size of the returned OID. The returned error code may be `GPG_ERR_TOO_SHORT` if the provided buffer is to short to receive the OID; it is possible to call the function with `NULL` for *buffer* to have it only return the required size. The function returns 0 on success.

To test whether an algorithm is actually available for use, the following macro should be used:

`gcry_error_t gcry_md_test_algo` (*int* `algo`) [Function]

The macro returns 0 if the algorithm *algo* is available for use.

If the length of a message digest is not known, it can be retrieved using the following function:

`unsigned int gcry_md_get_algo_dlen` (*int* `algo`) [Function]

Retrieve the length in bytes of the digest yielded by algorithm *algo*. This is often used prior to `gcry_md_read` to allocate sufficient memory for the digest.

In some situations it might be hard to remember the algorithm used for the ongoing hashing. The following function might be used to get that information:

`int gcry_md_get_algo` (*gcry_md_hd_t* `h`) [Function]

Retrieve the algorithm used with the handle *h*. Note that this does not work reliable if more than one algorithm is enabled in *h*.

The following macro might also be useful:

`int gcry_md_is_secure` (*gcry_md_hd_t* `h`) [Function]

This function returns true when the digest object *h* is allocated in "secure memory"; i.e. *h* was created with the `GCRY_MD_FLAG_SECURE`.

`int gcry_md_is_enabled` (*gcry_md_hd_t* `h`, *int* `algo`) [Function]

This function returns true when the algorithm *algo* has been enabled for the digest object *h*.

Tracking bugs related to hashing is often a cumbersome task which requires to add a lot of printf statements into the code. Libgcrypt provides an easy way to avoid this. The actual data hashed can be written to files on request.

`void gcry_md_debug` (*gcry_md_hd_t* `h`, *const char* *`suffix`) [Function]

Enable debugging for the digest object with handle *h*. This creates files named `dbgmd-<n>.<string>` while doing the actual hashing. *suffix* is the string part in the filename. The number is a counter incremented for each new hashing. The data in the file is the raw data as passed to `gcry_md_write` or `gcry_md_putc`. If `NULL` is used for *suffix*, the debugging is stopped and the file closed. This is only rarely required because `gcry_md_close` implicitly stops debugging.

8 Message Authentication Codes

Libgcrypt provides an easy and consistent to use interface for generating Message Authentication Codes (MAC). MAC generation is buffered and interface similar to the one used with hash algorithms. The programming model follows an open/process/close paradigm and is in that similar to other building blocks provided by Libgcrypt.

8.1 Available MAC algorithms

`GCRY_MAC_NONE`
: This is not a real algorithm but used by some functions as an error return value. This constant is guaranteed to have the value `0`.

`GCRY_MAC_HMAC_SHA256`
: This is keyed-hash message authentication code (HMAC) message authentication algorithm based on the SHA-256 hash algorithm.

`GCRY_MAC_HMAC_SHA224`
: This is HMAC message authentication algorithm based on the SHA-224 hash algorithm.

`GCRY_MAC_HMAC_SHA512`
: This is HMAC message authentication algorithm based on the SHA-512 hash algorithm.

`GCRY_MAC_HMAC_SHA384`
: This is HMAC message authentication algorithm based on the SHA-384 hash algorithm.

`GCRY_MAC_HMAC_SHA3_256`
: This is HMAC message authentication algorithm based on the SHA3-256 hash algorithm.

`GCRY_MAC_HMAC_SHA3_224`
: This is HMAC message authentication algorithm based on the SHA3-224 hash algorithm.

`GCRY_MAC_HMAC_SHA3_512`
: This is HMAC message authentication algorithm based on the SHA3-512 hash algorithm.

`GCRY_MAC_HMAC_SHA3_384`
: This is HMAC message authentication algorithm based on the SHA3-384 hash algorithm.

`GCRY_MAC_HMAC_SHA512_224`
: This is HMAC message authentication algorithm based on the SHA-512/224 hash algorithm.

`GCRY_MAC_HMAC_SHA512_256`
: This is HMAC message authentication algorithm based on the SHA-512/256 hash algorithm.

`GCRY_MAC_HMAC_SHA1`
: This is HMAC message authentication algorithm based on the SHA-1 hash algorithm.

`GCRY_MAC_HMAC_MD5`
: This is HMAC message authentication algorithm based on the MD5 hash algorithm.

`GCRY_MAC_HMAC_MD4`
: This is HMAC message authentication algorithm based on the MD4 hash algorithm.

`GCRY_MAC_HMAC_RMD160`
: This is HMAC message authentication algorithm based on the RIPE-MD-160 hash algorithm.

`GCRY_MAC_HMAC_WHIRLPOOL`
: This is HMAC message authentication algorithm based on the WHIRLPOOL hash algorithm.

`GCRY_MAC_HMAC_GOSTR3411_94`
: This is HMAC message authentication algorithm based on the GOST R 34.11-94 hash algorithm.

`GCRY_MAC_HMAC_STRIBOG256`
: This is HMAC message authentication algorithm based on the 256-bit hash algorithm described in GOST R 34.11-2012.

`GCRY_MAC_HMAC_STRIBOG512`
: This is HMAC message authentication algorithm based on the 512-bit hash algorithm described in GOST R 34.11-2012.

`GCRY_MAC_HMAC_BLAKE2B_512`
: This is HMAC message authentication algorithm based on the BLAKE2b-512 hash algorithm.

`GCRY_MAC_HMAC_BLAKE2B_384`
: This is HMAC message authentication algorithm based on the BLAKE2b-384 hash algorithm.

`GCRY_MAC_HMAC_BLAKE2B_256`
: This is HMAC message authentication algorithm based on the BLAKE2b-256 hash algorithm.

`GCRY_MAC_HMAC_BLAKE2B_160`
: This is HMAC message authentication algorithm based on the BLAKE2b-160 hash algorithm.

`GCRY_MAC_HMAC_BLAKE2S_256`
: This is HMAC message authentication algorithm based on the BLAKE2s-256 hash algorithm.

`GCRY_MAC_HMAC_BLAKE2S_224`
: This is HMAC message authentication algorithm based on the BLAKE2s-224 hash algorithm.

`GCRY_MAC_HMAC_BLAKE2S_160`
: This is HMAC message authentication algorithm based on the BLAKE2s-160 hash algorithm.

`GCRY_MAC_HMAC_BLAKE2S_128`
: This is HMAC message authentication algorithm based on the BLAKE2s-128 hash algorithm.

`GCRY_MAC_HMAC_SM3`
: This is HMAC message authentication algorithm based on the SM3 hash algorithm.

`GCRY_MAC_CMAC_AES`
: This is CMAC (Cipher-based MAC) message authentication algorithm based on the AES block cipher algorithm.

`GCRY_MAC_CMAC_3DES`
: This is CMAC message authentication algorithm based on the three-key EDE Triple-DES block cipher algorithm.

`GCRY_MAC_CMAC_CAMELLIA`
: This is CMAC message authentication algorithm based on the Camellia block cipher algorithm.

`GCRY_MAC_CMAC_CAST5`
: This is CMAC message authentication algorithm based on the CAST128-5 block cipher algorithm.

`GCRY_MAC_CMAC_BLOWFISH`
: This is CMAC message authentication algorithm based on the Blowfish block cipher algorithm.

`GCRY_MAC_CMAC_TWOFISH`
: This is CMAC message authentication algorithm based on the Twofish block cipher algorithm.

`GCRY_MAC_CMAC_SERPENT`
: This is CMAC message authentication algorithm based on the Serpent block cipher algorithm.

`GCRY_MAC_CMAC_SEED`
: This is CMAC message authentication algorithm based on the SEED block cipher algorithm.

`GCRY_MAC_CMAC_RFC2268`
: This is CMAC message authentication algorithm based on the Ron's Cipher 2 block cipher algorithm.

`GCRY_MAC_CMAC_IDEA`
: This is CMAC message authentication algorithm based on the IDEA block cipher algorithm.

`GCRY_MAC_CMAC_GOST28147`
: This is CMAC message authentication algorithm based on the GOST 28147-89 block cipher algorithm.

`GCRY_MAC_CMAC_SM4`
: This is CMAC message authentication algorithm based on the SM4 block cipher algorithm.

`GCRY_MAC_GMAC_AES`
: This is GMAC (GCM mode based MAC) message authentication algorithm based on the AES block cipher algorithm.

`GCRY_MAC_GMAC_CAMELLIA`
: This is GMAC message authentication algorithm based on the Camellia block cipher algorithm.

`GCRY_MAC_GMAC_TWOFISH`
: This is GMAC message authentication algorithm based on the Twofish block cipher algorithm.

`GCRY_MAC_GMAC_SERPENT`
: This is GMAC message authentication algorithm based on the Serpent block cipher algorithm.

`GCRY_MAC_GMAC_SEED`
: This is GMAC message authentication algorithm based on the SEED block cipher algorithm.

`GCRY_MAC_POLY1305`
: This is plain Poly1305 message authentication algorithm, used with one-time key.

`GCRY_MAC_POLY1305_AES`
: This is Poly1305-AES message authentication algorithm, used with key and one-time nonce.

`GCRY_MAC_POLY1305_CAMELLIA`
: This is Poly1305-Camellia message authentication algorithm, used with key and one-time nonce.

`GCRY_MAC_POLY1305_TWOFISH`
: This is Poly1305-Twofish message authentication algorithm, used with key and one-time nonce.

`GCRY_MAC_POLY1305_SERPENT`
: This is Poly1305-Serpent message authentication algorithm, used with key and one-time nonce.

`GCRY_MAC_POLY1305_SEED`
: This is Poly1305-SEED message authentication algorithm, used with key and one-time nonce.

`GCRY_MAC_GOST28147_IMIT`
: This is MAC construction defined in GOST 28147-89 (see RFC 5830 Section 8).

8.2 Working with MAC algorithms

To use most of these function it is necessary to create a context; this is done using:

`gcry_error_t gcry_mac_open` (*gcry_mac_hd_t* *`*hd`*, *int* `algo`, *unsigned int* `flags`, *gcry_ctx_t* `ctx`) [Function]

Create a MAC object for algorithm *algo*. *flags* may be given as an bitwise OR of constants described below. *hd* is guaranteed to either receive a valid handle or NULL. *ctx* is context object to associate MAC object with. *ctx* maybe set to NULL.

For a list of supported algorithms, see Section 8.1 [Available MAC algorithms], page 59.

The flags allowed for *mode* are:

`GCRY_MAC_FLAG_SECURE`
: Allocate all buffers and the resulting MAC in "secure memory". Use this if the MAC data is highly confidential.

In order to use a handle for performing MAC algorithm operations, a 'key' has to be set first:

`gcry_error_t gcry_mac_setkey` (*gcry_mac_hd_t* `h`, *const void* *`*key`*, *size_t* `keylen`) [Function]

Set the MAC key to the value of *key* of length *keylen* bytes. With HMAC algorithms, there is no restriction on the length of the key. With CMAC algorithms, the length of the key is restricted to those supported by the underlying block cipher.

GMAC algorithms and Poly1305-with-cipher algorithms need initialization vector to be set, which can be performed with function:

`gcry_error_t gcry_mac_setiv` (*gcry_mac_hd_t* `h`, *const void* *`*iv`*, *size_t* `ivlen`) [Function]

Set the IV to the value of *iv* of length *ivlen* bytes.

After you are done with the MAC calculation, you should release the resources by using:

`void gcry_mac_close` (*gcry_mac_hd_t* `h`) [Function]

Release all resources of MAC context *h*. *h* should not be used after a call to this function. A `NULL` passed as *h* is ignored. The function also clears all sensitive information associated with this handle.

Often you have to do several MAC operations using the same algorithm. To avoid the overhead of creating and releasing context, a reset function is provided:

`gcry_error_t gcry_mac_reset` (*gcry_mac_hd_t* `h`) [Function]

Reset the current context to its initial state. This is effectively identical to a close followed by an open and setting same key.

Note that gcry_mac_reset is implemented as a macro.

Now that we have prepared everything to calculate MAC, it is time to see how it is actually done.

`gcry_error_t gcry_mac_write` (*gcry_mac_hd_t* `h`, *const void* *`*buffer`*, *size_t* `length`) [Function]

Pass *length* bytes of the data in *buffer* to the MAC object with handle *h* to update the MAC values. If this function is used after the context has been finalized, it will keep on pushing the data through the algorithm specific transform function and thereby change the context; however the results are not meaningful and this feature is only available to mitigate timing attacks.

The way to read out the calculated MAC is by using the function:

`gcry_error_t gcry_mac_read` (*gcry_mac_hd_t* `h`, *void* `*buffer`, *size_t* `*length`) [Function]

`gcry_mac_read` returns the MAC after finalizing the calculation. Function copies the resulting MAC value to *buffer* of the length *length*. If *length* is larger than length of resulting MAC value, then length of MAC is returned through *length*.

To compare existing MAC value with recalculated MAC, one is to use the function:

`gcry_error_t gcry_mac_verify` (*gcry_mac_hd_t* `h`, *void* `*buffer`, *size_t* `length`) [Function]

`gcry_mac_verify` finalizes MAC calculation and compares result with *length* bytes of data in *buffer*. Error code `GPG_ERR_CHECKSUM` is returned if the MAC value in the buffer *buffer* does not match the MAC calculated in object *h*.

In some situations it might be hard to remember the algorithm used for the MAC calculation. The following function might be used to get that information:

`int gcry_mac_get_algo` (*gcry_mac_hd_t* `h`) [Function]

Retrieve the algorithm used with the handle *h*.

MAC algorithms are identified by internal algorithm numbers (see `gcry_mac_open` for a list). However, in most applications they are used by names, so two functions are available to map between string representations and MAC algorithm identifiers.

`const char * gcry_mac_algo_name` (*int* `algo`) [Function]

Map the MAC algorithm id *algo* to a string representation of the algorithm name. For unknown algorithms this function returns the string `"?"`. This function should not be used to test for the availability of an algorithm.

`int gcry_mac_map_name` (*const char* `*name`) [Function]

Map the algorithm with *name* to a MAC algorithm identifier. Returns 0 if the algorithm name is not known. This function should not be used to test for the availability of an algorithm.

To test whether an algorithm is actually available for use, the following macro should be used:

`gcry_error_t gcry_mac_test_algo` (*int* `algo`) [Function]

The macro returns 0 if the MAC algorithm *algo* is available for use.

If the length of a message digest is not known, it can be retrieved using the following function:

`unsigned int gcry_mac_get_algo_maclen (int algo)` [Function]

Retrieve the length in bytes of the MAC yielded by algorithm *algo*. This is often used prior to `gcry_mac_read` to allocate sufficient memory for the MAC value. On error `0` is returned.

`unsigned int gcry_mac_get_algo_keylen (algo)` [Function]

This function returns length of the key for MAC algorithm *algo*. If the algorithm supports multiple key lengths, the default supported key length is returned. On error `0` is returned. The key length is returned as number of octets.

9 Key Derivation

Libgcypt provides a general purpose function to derive keys from strings.

`gpg_error_t gcry_kdf_derive` (*const void* *`*passphrase`*, *size_t* `passphraselen`, *int* `algo`, *int* `subalgo`, *const void* *`*salt`*, *size_t* `saltlen`, *unsigned long* `iterations`, *size_t* `keysize`, *void* *`*keybuffer`*) [Function]

Derive a key from a passphrase. *keysize* gives the requested size of the keys in octets. *keybuffer* is a caller provided buffer filled on success with the derived key. The input passphrase is taken from *passphrase* which is an arbitrary memory buffer of *passphraselen* octets. *algo* specifies the KDF algorithm to use; see below. *subalgo* specifies an algorithm used internally by the KDF algorithms; this is usually a hash algorithm but certain KDF algorithms may use it differently. *salt* is a salt of length *saltlen* octets, as needed by most KDF algorithms. *iterations* is a positive integer parameter to most KDFs.

On success 0 is returned; on failure an error code.

Currently supported KDFs (parameter *algo*):

`GCRY_KDF_SIMPLE_S2K`
: The OpenPGP simple S2K algorithm (cf. RFC4880). Its use is strongly deprecated. *salt* and *iterations* are not needed and may be passed as `NULL/0`.

`GCRY_KDF_SALTED_S2K`
: The OpenPGP salted S2K algorithm (cf. RFC4880). Usually not used. *iterations* is not needed and may be passed as `0`. *saltlen* must be given as 8.

`GCRY_KDF_ITERSALTED_S2K`
: The OpenPGP iterated+salted S2K algorithm (cf. RFC4880). This is the default for most OpenPGP applications. *saltlen* must be given as 8. Note that OpenPGP defines a special encoding of the *iterations*; however this function takes the plain decoded iteration count.

`GCRY_KDF_PBKDF2`
: The PKCS#5 Passphrase Based Key Derivation Function number 2.

`GCRY_KDF_SCRYPT`
: The SCRYPT Key Derivation Function. The subalgorithm is used to specify the CPU/memory cost parameter N, and the number of iterations is used for the parallelization parameter p. The block size is fixed at 8 in the current implementation.

10 Random Numbers

10.1 Quality of random numbers

Libgcypt offers random numbers of different quality levels:

`gcry_random_level_t` [Data type]

The constants for the random quality levels are of this enum type.

`GCRY_WEAK_RANDOM`
: For all functions, except for `gcry_mpi_randomize`, this level maps to GCRY_STRONG_RANDOM. If you do not want this, consider using `gcry_create_nonce`.

`GCRY_STRONG_RANDOM`
: Use this level for session keys and similar purposes.

`GCRY_VERY_STRONG_RANDOM`
: Use this level for long term key material.

10.2 Retrieving random numbers

`void gcry_randomize` (*unsigned char **`buffer`, *size_t* `length`, *enum gcry_random_level* `level`) [Function]

Fill *buffer* with *length* random bytes using a random quality as defined by *level*.

`void * gcry_random_bytes` (*size_t* `nbytes`, *enum gcry_random_level* `level`) [Function]

Convenience function to allocate a memory block consisting of *nbytes* fresh random bytes using a random quality as defined by *level*.

`void * gcry_random_bytes_secure` (*size_t* `nbytes`, *enum gcry_random_level* `level`) [Function]

Convenience function to allocate a memory block consisting of *nbytes* fresh random bytes using a random quality as defined by *level*. This function differs from `gcry_random_bytes` in that the returned buffer is allocated in a "secure" area of the memory.

`void gcry_create_nonce` (*unsigned char **`buffer`, *size_t* `length`) [Function]

Fill *buffer* with *length* unpredictable bytes. This is commonly called a nonce and may also be used for initialization vectors and padding. This is an extra function nearly independent of the other random function for 3 reasons: It better protects the regular random generator's internal state, provides better performance and does not drain the precious entropy pool.

11 S-expressions

S-expressions are used by the public key functions to pass complex data structures around. These LISP like objects are used by some cryptographic protocols (cf. RFC-2692) and Libgcrypt provides functions to parse and construct them. For detailed information, see *Ron Rivest, code and description of S-expressions*, `http://theory.lcs.mit.edu/~rivest/sexp.html`.

11.1 Data types for S-expressions

`gcry_sexp_t` [Data type]

The `gcry_sexp_t` type describes an object with the Libgcrypt internal representation of an S-expression.

11.2 Working with S-expressions

There are several functions to create an Libgcrypt S-expression object from its external representation or from a string template. There is also a function to convert the internal representation back into one of the external formats:

`gcry_error_t gcry_sexp_new` (*gcry_sexp_t* *`r_sexp`, *const void* *`buffer`, *size_t* `length`, *int* `autodetect`) [Function]

This is the generic function to create an new S-expression object from its external representation in *buffer* of *length* bytes. On success the result is stored at the address given by *r_sexp*. With *autodetect* set to 0, the data in *buffer* is expected to be in canonized format, with *autodetect* set to 1 the parses any of the defined external formats. If *buffer* does not hold a valid S-expression an error code is returned and *r_sexp* set to `NULL`. Note that the caller is responsible for releasing the newly allocated S-expression using `gcry_sexp_release`.

`gcry_error_t gcry_sexp_create` (*gcry_sexp_t* *`r_sexp`, *void* *`buffer`, *size_t* `length`, *int* `autodetect`, *void* (*`freefnc`)(*void**)) [Function]

This function is identical to `gcry_sexp_new` but has an extra argument *freefnc*, which, when not set to `NULL`, is expected to be a function to release the *buffer*; most likely the standard `free` function is used for this argument. This has the effect of transferring the ownership of *buffer* to the created object in *r_sexp*. The advantage of using this function is that Libgcrypt might decide to directly use the provided buffer and thus avoid extra copying.

`gcry_error_t gcry_sexp_sscan` (*gcry_sexp_t* *`r_sexp`, *size_t* *`erroff`, *const char* *`buffer`, *size_t* `length`) [Function]

This is another variant of the above functions. It behaves nearly identical but provides an *erroff* argument which will receive the offset into the buffer where the parsing stopped on error.

`gcry_error_t gcry_sexp_build` (*gcry_sexp_t* *`r_sexp`, *size_t* *`erroff`, *const char* *`format`, *...*) [Function]

This function creates an internal S-expression from the string template *format* and stores it at the address of *r_sexp*. If there is a parsing error, the function returns an appropriate error code and stores the offset into *format* where the parsing stopped in *erroff*. The function supports a couple of printf-like formatting characters and expects arguments for some of these escape sequences right after *format*. The following format characters are defined:

'%m' The next argument is expected to be of type `gcry_mpi_t` and a copy of its value is inserted into the resulting S-expression. The MPI is stored as a signed integer.

'%M' The next argument is expected to be of type `gcry_mpi_t` and a copy of its value is inserted into the resulting S-expression. The MPI is stored as an unsigned integer.

'%s' The next argument is expected to be of type `char *` and that string is inserted into the resulting S-expression.

'%d' The next argument is expected to be of type `int` and its value is inserted into the resulting S-expression.

'%u' The next argument is expected to be of type `unsigned int` and its value is inserted into the resulting S-expression.

'%b' The next argument is expected to be of type `int` directly followed by an argument of type `char *`. This represents a buffer of given length to be inserted into the resulting S-expression.

'%S' The next argument is expected to be of type `gcry_sexp_t` and a copy of that S-expression is embedded in the resulting S-expression. The argument needs to be a regular S-expression, starting with a parenthesis.

No other format characters are defined and would return an error. Note that the format character '%%' does not exists, because a percent sign is not a valid character in an S-expression.

`void gcry_sexp_release` (*gcry_sexp_t* `sexp`) [Function]

Release the S-expression object *sexp*. If the S-expression is stored in secure memory it explicitly zeroises that memory; note that this is done in addition to the zeroisation always done when freeing secure memory.

The next 2 functions are used to convert the internal representation back into a regular external S-expression format and to show the structure for debugging.

`size_t gcry_sexp_sprint` (*gcry_sexp_t* `sexp`, *int* `mode`, *char* *`buffer`, *size_t* `maxlength`) [Function]

Copies the S-expression object *sexp* into *buffer* using the format specified in *mode*. *maxlength* must be set to the allocated length of *buffer*. The function returns the actual length of valid bytes put into *buffer* or 0 if the provided buffer is too short. Passing `NULL` for *buffer* returns the required length for *buffer*. For convenience reasons an extra byte with value 0 is appended to the buffer.

The following formats are supported:

`GCRYSEXP_FMT_DEFAULT`

Returns a convenient external S-expression representation.

`GCRYSEXP_FMT_CANON`
: Return the S-expression in canonical format.

`GCRYSEXP_FMT_BASE64`
: Not currently supported.

`GCRYSEXP_FMT_ADVANCED`
: Returns the S-expression in advanced format.

`void gcry_sexp_dump` (*gcry_sexp_t* `sexp`) [Function]

Dumps *sexp* in a format suitable for debugging to Libgcrypt's logging stream.

Often canonical encoding is used in the external representation. The following function can be used to check for valid encoding and to learn the length of the S-expression.

`size_t gcry_sexp_canon_len` (*const unsigned char* *`buffer`, *size_t* `length`, *size_t* *`erroff`, *int* *`errcode`) [Function]

Scan the canonical encoded *buffer* with implicit length values and return the actual length this S-expression uses. For a valid S-expression it should never return 0. If *length* is not 0, the maximum length to scan is given; this can be used for syntax checks of data passed from outside. *errcode* and *erroff* may both be passed as `NULL`.

There are functions to parse S-expressions and retrieve elements:

`gcry_sexp_t gcry_sexp_find_token` (*const gcry_sexp_t* `list`, *const char* *`token`, *size_t* `toklen`) [Function]

Scan the S-expression for a sublist with a type (the car of the list) matching the string *token*. If *toklen* is not 0, the token is assumed to be raw memory of this length. The function returns a newly allocated S-expression consisting of the found sublist or `NULL` when not found.

`int gcry_sexp_length` (*const gcry_sexp_t* `list`) [Function]

Return the length of the *list*. For a valid S-expression this should be at least 1.

`gcry_sexp_t gcry_sexp_nth` (*const gcry_sexp_t* `list`, *int* `number`) [Function]

Create and return a new S-expression from the element with index *number* in *list*. Note that the first element has the index 0. If there is no such element, `NULL` is returned.

`gcry_sexp_t gcry_sexp_car` (*const gcry_sexp_t* `list`) [Function]

Create and return a new S-expression from the first element in *list*; this is called the "type" and should always exist per S-expression specification and in general be a string. `NULL` is returned in case of a problem.

`gcry_sexp_t gcry_sexp_cdr` (*const gcry_sexp_t* `list`) [Function]

Create and return a new list form all elements except for the first one. Note that this function may return an invalid S-expression because it is not guaranteed, that the type exists and is a string. However, for parsing a complex S-expression it might be useful for intermediate lists. Returns `NULL` on error.

const char * **gcry_sexp_nth_data** (*const gcry_sexp_t list*, *int number*, *size_t *datalen*) [Function]

This function is used to get data from a *list*. A pointer to the actual data with index *number* is returned and the length of this data will be stored to *datalen*. If there is no data at the given index or the index represents another list, `NULL` is returned. **Caution:** The returned pointer is valid as long as *list* is not modified or released.

Here is an example on how to extract and print the surname (Meier) from the S-expression '`(Name Otto Meier (address Burgplatz 3))`':

```
size_t len;
const char *name;

name = gcry_sexp_nth_data (list, 2, &len);
printf ("my name is %.*s\n", (int)len, name);
```

void * **gcry_sexp_nth_buffer** (*const gcry_sexp_t list*, *int number*, *size_t *rlength*) [Function]

This function is used to get data from a *list*. A malloced buffer with the actual data at list index *number* is returned and the length of this buffer will be stored to *rlength*. If there is no data at the given index or the index represents another list, `NULL` is returned. The caller must release the result using `gcry_free`.

Here is an example on how to extract and print the CRC value from the S-expression '`(hash crc32 #23ed00d7)`':

```
size_t len;
char *value;

value = gcry_sexp_nth_buffer (list, 2, &len);
if (value)
  fwrite (value, len, 1, stdout);
gcry_free (value);
```

char * **gcry_sexp_nth_string** (*gcry_sexp_t list*, *int number*) [Function]

This function is used to get and convert data from a *list*. The data is assumed to be a Nul terminated string. The caller must release this returned value using `gcry_free`. If there is no data at the given index, the index represents a list or the value can't be converted to a string, `NULL` is returned.

gcry_mpi_t **gcry_sexp_nth_mpi** (*gcry_sexp_t list*, *int number*, *int mpifmt*) [Function]

This function is used to get and convert data from a *list*. This data is assumed to be an MPI stored in the format described by *mpifmt* and returned as a standard Libgcrypt MPI. The caller must release this returned value using `gcry_mpi_release`. If there is no data at the given index, the index represents a list or the value can't be converted to an MPI, `NULL` is returned. If you use this function to parse results of a public key function, you most likely want to use `GCRYMPI_FMT_USG`.

gpg_error_t gcry_sexp_extract_param (*gcry_sexp_t* sexp, *const char* *path, *const char* *list, ...) [Function]

Extract parameters from an S-expression using a list of parameter names. The names of these parameters are specified in LIST. White space between the parameter names are ignored. Some special characters and character sequences may be given to control the conversion:

'+' Switch to unsigned integer format (GCRYMPI_FMT_USG). This is the default mode.

'-' Switch to standard signed format (GCRYMPI_FMT_STD).

'/' Switch to opaque MPI format. The resulting MPIs may not be used for computations; see `gcry_mpi_get_opaque` for details.

'&' Switch to buffer descriptor mode. See below for details.

'%s' Switch to string mode. The expected argument is the address of a `char *` variable; the caller must release that value. If the parameter was marked optional and is not found, NULL is stored.

'%#s' Switch to multi string mode. The expected argument is the address of a `char *` variable; the caller must release that value. If the parameter was marked optional and is not found, NULL is stored. A multi string takes all values, assumes they are strings and concatenates them using a space as delimiter. In case a value is actually another list this is not further parsed but a `()` is inserted in place of that sublist.

'%u' Switch to unsigned integer mode. The expected argument is address of a `unsigned int` variable.

'%lu' Switch to unsigned long integer mode. The expected argument is address of a `unsigned long` variable.

'%d' Switch to signed integer mode. The expected argument is address of a `int` variable.

'%ld' Switch to signed long integer mode. The expected argument is address of a `long` variable.

'%zu' Switch to size_t mode. The expected argument is address of a `size_t` variable.

'?' If immediately following a parameter letter (no white space allowed), that parameter is considered optional.

In general parameter names are single letters. To use a string for a parameter name, enclose the name in single quotes.

Unless in buffer descriptor mode for each parameter name a pointer to an `gcry_mpi_t` variable is expected that must be set to `NULL` prior to invoking this function, and finally a `NULL` is expected. For example

```
gcry_sexp_extract_param (key, NULL, "n/x+e d-'foo'",
                         &mpi_n, &mpi_x, &mpi_e, &mpi_d, &mpi_foo, NULL)
```

stores the parameter 'n' from *key* as an unsigned MPI into *mpi_n*, the parameter 'x' as an opaque MPI into *mpi_x*, the parameters 'e' and 'd' again as an unsigned MPI into *mpi_e* and *mpi_d* and finally the parameter 'foo' as a signed MPI into *mpi_foo*.

path is an optional string used to locate a token. The exclamation mark separated tokens are used via `gcry_sexp_find_token` to find a start point inside the S-expression.

In buffer descriptor mode a pointer to a `gcry_buffer_t` descriptor is expected instead of a pointer to an MPI. The caller may use two different operation modes here: If the *data* field of the provided descriptor is `NULL`, the function allocates a new buffer and stores it at *data*; the other fields are set accordingly with *off* set to 0. If *data* is not `NULL`, the function assumes that the *data*, *size*, and *off* fields specify a buffer where to but the value of the respective parameter; on return the *len* field receives the number of bytes copied to that buffer; in case the buffer is too small, the function immediately returns with an error code (and *len* is set to 0).

The function returns 0 on success. On error an error code is returned, all passed MPIs that might have been allocated up to this point are deallocated and set to `NULL`, and all passed buffers are either truncated if the caller supplied the buffer, or deallocated if the function allocated the buffer.

12 MPI library

Public key cryptography is based on mathematics with large numbers. To implement the public key functions, a library for handling these large numbers is required. Because of the general usefulness of such a library, its interface is exposed by Libgcrypt. In the context of Libgcrypt and in most other applications, these large numbers are called MPIs (multi-precision-integers).

12.1 Data types

`gcry_mpi_t` [Data type]

This type represents an object to hold an MPI.

`gcry_mpi_point_t` [Data type]

This type represents an object to hold a point for elliptic curve math.

12.2 Basic functions

To work with MPIs, storage must be allocated and released for the numbers. This can be done with one of these functions:

`gcry_mpi_t gcry_mpi_new` (*unsigned int* `nbits`) [Function]

Allocate a new MPI object, initialize it to 0 and initially allocate enough memory for a number of at least *nbits*. This pre-allocation is only a small performance issue and not actually necessary because Libgcrypt automatically re-allocates the required memory.

`gcry_mpi_t gcry_mpi_snew` (*unsigned int* `nbits`) [Function]

This is identical to `gcry_mpi_new` but allocates the MPI in the so called "secure memory" which in turn will take care that all derived values will also be stored in this "secure memory". Use this for highly confidential data like private key parameters.

`gcry_mpi_t gcry_mpi_copy` (*const gcry_mpi_t* `a`) [Function]

Create a new MPI as the exact copy of *a* but with the constant and immutable flags cleared.

`void gcry_mpi_release` (*gcry_mpi_t* `a`) [Function]

Release the MPI *a* and free all associated resources. Passing `NULL` is allowed and ignored. When a MPI stored in the "secure memory" is released, that memory gets wiped out immediately.

The simplest operations are used to assign a new value to an MPI:

`gcry_mpi_t gcry_mpi_set` (*gcry_mpi_t* `w`, *const gcry_mpi_t* `u`) [Function]

Assign the value of *u* to *w* and return *w*. If `NULL` is passed for *w*, a new MPI is allocated, set to the value of *u* and returned.

`gcry_mpi_t gcry_mpi_set_ui` (*gcry_mpi_t* `w`, *unsigned long* `u`) [Function]

Assign the value of *u* to *w* and return *w*. If `NULL` is passed for *w*, a new MPI is allocated, set to the value of *u* and returned. This function takes an `unsigned int` as type for *u* and thus it is only possible to set *w* to small values (usually up to the word size of the CPU).

gcry_error_t **gcry_mpi_get_ui** (*unsigned int *w, gcry_mpi_t u*) [Function]

If *u* is not negative and small enough to be stored in an `unsigned int` variable, store its value at *w*. If the value does not fit or is negative return GPG_ERR_ERANGE and do not change the value stored at *w*. Note that this function returns an `unsigned int` so that this value can immediately be used with the bit test functions. This is in contrast to the other "_ui" functions which allow for values up to an `unsigned long`.

void **gcry_mpi_swap** (*gcry_mpi_t a, gcry_mpi_t b*) [Function]

Swap the values of *a* and *b*.

void **gcry_mpi_snatch** (*gcry_mpi_t w, const gcry_mpi_t u*) [Function]

Set *u* into *w* and release *u*. If *w* is `NULL` only *u* will be released.

void **gcry_mpi_neg** (*gcry_mpi_t w, gcry_mpi_t u*) [Function]

Set the sign of *w* to the negative of *u*.

void **gcry_mpi_abs** (*gcry_mpi_t w*) [Function]

Clear the sign of *w*.

12.3 MPI formats

The following functions are used to convert between an external representation of an MPI and the internal one of Libgcrypt.

gcry_error_t **gcry_mpi_scan** (*gcry_mpi_t *r_mpi, enum gcry_mpi_format format, const unsigned char *buffer, size_t buflen, size_t *nscanned*) [Function]

Convert the external representation of an integer stored in *buffer* with a length of *buflen* into a newly created MPI returned which will be stored at the address of *r_mpi*. For certain formats the length argument is not required and should be passed as `0`. A *buflen* larger than 16 MiByte will be rejected. After a successful operation the variable *nscanned* receives the number of bytes actually scanned unless *nscanned* was given as `NULL`. *format* describes the format of the MPI as stored in *buffer*:

`GCRYMPI_FMT_STD`
: 2-complement stored without a length header. Note that `gcry_mpi_print` stores a `0` as a string of zero length.

`GCRYMPI_FMT_PGP`
: As used by OpenPGP (only defined as unsigned). This is basically `GCRYMPI_FMT_STD` with a 2 byte big endian length header. A length header indicating a length of more than 16384 is not allowed.

`GCRYMPI_FMT_SSH`
: As used in the Secure Shell protocol. This is `GCRYMPI_FMT_STD` with a 4 byte big endian header.

`GCRYMPI_FMT_HEX`
: Stored as a string with each byte of the MPI encoded as 2 hex digits. Negative numbers are prefix with a minus sign and in addition the high bit is always zero to make clear that an explicit sign ist used. When using this format, *buflen* must be zero.

`GCRYMPI_FMT_USG`
: Simple unsigned integer.

Note that all of the above formats store the integer in big-endian format (MSB first).

`gcry_error_t gcry_mpi_print` (*enum gcry_mpi_format* `format`, *unsigned char* *`buffer`, *size_t* `buflen`, *size_t* *`nwritten`, *const gcry_mpi_t* `a`) [Function]

Convert the MPI *a* into an external representation described by *format* (see above) and store it in the provided *buffer* which has a usable length of at least the *buflen* bytes. If *nwritten* is not NULL, it will receive the number of bytes actually stored in *buffer* after a successful operation.

`gcry_error_t gcry_mpi_aprint` (*enum gcry_mpi_format* `format`, *unsigned char* **`buffer`, *size_t* *`nbytes`, *const gcry_mpi_t* `a`) [Function]

Convert the MPI *a* into an external representation described by *format* (see above) and store it in a newly allocated buffer which address will be stored in the variable *buffer* points to. The number of bytes stored in this buffer will be stored in the variable *nbytes* points to, unless *nbytes* is `NULL`.

Even if *nbytes* is zero, the function allocates at least one byte and store a zero there. Thus with formats `GCRYMPI_FMT_STD` and `GCRYMPI_FMT_USG` the caller may safely set a returned length of 0 to 1 to represent a zero as a 1 byte string.

`void gcry_mpi_dump` (*const gcry_mpi_t* `a`) [Function]

Dump the value of *a* in a format suitable for debugging to Libgcrypt's logging stream. Note that one leading space but no trailing space or linefeed will be printed. It is okay to pass `NULL` for *a*.

12.4 Calculations

Basic arithmetic operations:

`void gcry_mpi_add` (*gcry_mpi_t* `w`, *gcry_mpi_t* `u`, *gcry_mpi_t* `v`) [Function]

$w = u + v$.

`void gcry_mpi_add_ui` (*gcry_mpi_t* `w`, *gcry_mpi_t* `u`, *unsigned long* `v`) [Function]

$w = u + v$. Note that v is an unsigned integer.

`void gcry_mpi_addm` (*gcry_mpi_t* `w`, *gcry_mpi_t* `u`, *gcry_mpi_t* `v`, *gcry_mpi_t* `m`) [Function]

$w = u + v \bmod m$.

`void gcry_mpi_sub` (*gcry_mpi_t* `w`, *gcry_mpi_t* `u`, *gcry_mpi_t* `v`) [Function]

$w = u - v$.

`void gcry_mpi_sub_ui` (*gcry_mpi_t* `w`, *gcry_mpi_t* `u`, *unsigned long* `v`) [Function]

$w = u - v$. v is an unsigned integer.

void gcry_mpi_subm (*gcry_mpi_t* w, *gcry_mpi_t* u, *gcry_mpi_t* v, *gcry_mpi_t* m) [Function]

$w = u - v \bmod m$.

void gcry_mpi_mul (*gcry_mpi_t* w, *gcry_mpi_t* u, *gcry_mpi_t* v) [Function]

$w = u * v$.

void gcry_mpi_mul_ui (*gcry_mpi_t* w, *gcry_mpi_t* u, *unsigned long* v) [Function]

$w = u * v$. v is an unsigned integer.

void gcry_mpi_mulm (*gcry_mpi_t* w, *gcry_mpi_t* u, *gcry_mpi_t* v, *gcry_mpi_t* m) [Function]

$w = u * v \bmod m$.

void gcry_mpi_mul_2exp (*gcry_mpi_t* w, *gcry_mpi_t* u, *unsigned long* e) [Function]

$w = u * 2^e$.

void gcry_mpi_div (*gcry_mpi_t* q, *gcry_mpi_t* r, *gcry_mpi_t* dividend, *gcry_mpi_t* divisor, *int* round) [Function]

$q = dividend/divisor$, $r = dividend \bmod divisor$. q and r may be passed as NULL. *round* is either negative for floored division (rounds towards the next lower integer) or zero for truncated division (rounds towards zero).

void gcry_mpi_mod (*gcry_mpi_t* r, *gcry_mpi_t* dividend, *gcry_mpi_t* divisor) [Function]

$r = dividend \bmod divisor$.

void gcry_mpi_powm (*gcry_mpi_t* w, *const gcry_mpi_t* b, *const gcry_mpi_t* e, *const gcry_mpi_t* m) [Function]

$w = b^e \bmod m$.

int gcry_mpi_gcd (*gcry_mpi_t* g, *gcry_mpi_t* a, *gcry_mpi_t* b) [Function]

Set g to the greatest common divisor of a and b. Return true if the g is 1.

int gcry_mpi_invm (*gcry_mpi_t* x, *gcry_mpi_t* a, *gcry_mpi_t* m) [Function]

Set x to the multiplicative inverse of $a \bmod m$. Return true if the inverse exists.

12.5 Comparisons

The next 2 functions are used to compare MPIs:

int gcry_mpi_cmp (*const gcry_mpi_t* u, *const gcry_mpi_t* v) [Function]

Compare the multi-precision-integers number u and v returning 0 for equality, a positive value for $u > v$ and a negative for $u < v$. If both numbers are opaque values (cf, gcry_mpi_set_opaque) the comparison is done by checking the bit sizes using memcmp. If only one number is an opaque value, the opaque value is less than the other number.

`int gcry_mpi_cmp_ui (const gcry_mpi_t u, unsigned long v)` [Function]
: Compare the multi-precision-integers number *u* with the unsigned integer *v* returning 0 for equality, a positive value for *u* > *v* and a negative for *u* < *v*.

`int gcry_mpi_is_neg (const gcry_mpi_t a)` [Function]
: Return 1 if *a* is less than zero; return 0 if zero or positive.

12.6 Bit manipulations

There are a couple of functions to get information on arbitrary bits in an MPI and to set or clear them:

`unsigned int gcry_mpi_get_nbits (gcry_mpi_t a)` [Function]
: Return the number of bits required to represent *a*.

`int gcry_mpi_test_bit (gcry_mpi_t a, unsigned int n)` [Function]
: Return true if bit number *n* (counting from 0) is set in *a*.

`void gcry_mpi_set_bit (gcry_mpi_t a, unsigned int n)` [Function]
: Set bit number *n* in *a*.

`void gcry_mpi_clear_bit (gcry_mpi_t a, unsigned int n)` [Function]
: Clear bit number *n* in *a*.

`void gcry_mpi_set_highbit (gcry_mpi_t a, unsigned int n)` [Function]
: Set bit number *n* in *a* and clear all bits greater than *n*.

`void gcry_mpi_clear_highbit (gcry_mpi_t a, unsigned int n)` [Function]
: Clear bit number *n* in *a* and all bits greater than *n*.

`void gcry_mpi_rshift (gcry_mpi_t x, gcry_mpi_t a, unsigned int n)` [Function]
: Shift the value of *a* by *n* bits to the right and store the result in *x*.

`void gcry_mpi_lshift (gcry_mpi_t x, gcry_mpi_t a, unsigned int n)` [Function]
: Shift the value of *a* by *n* bits to the left and store the result in *x*.

12.7 EC functions

Libgcrypt provides an API to access low level functions used by its elliptic curve implementation. These functions allow to implement elliptic curve methods for which no explicit support is available.

`gcry_mpi_point_t gcry_mpi_point_new (unsigned int nbits)` [Function]
: Allocate a new point object, initialize it to 0, and allocate enough memory for a points of at least *nbits*. This pre-allocation yields only a small performance win and is not really necessary because Libgcrypt automatically re-allocates the required memory. Using 0 for *nbits* is usually the right thing to do.

`void gcry_mpi_point_release (gcry_mpi_point_t point)` [Function]
: Release *point* and free all associated resources. Passing `NULL` is allowed and ignored.

gcry_mpi_point_t gcry_mpi_point_copy (*gcry_mpi_point_t* point) [Function]

Allocate and return a new point object and initialize it with *point*. If *point* is NULL the function is identical to gcry_mpi_point_new(0).

void gcry_mpi_point_get (*gcry_mpi_t* x, *gcry_mpi_t* y, *gcry_mpi_t* z, *gcry_mpi_point_t* point) [Function]

Store the projective coordinates from *point* into the MPIs *x*, *y*, and *z*. If a coordinate is not required, NULL may be used for *x*, *y*, or *z*.

void gcry_mpi_point_snatch_get (*gcry_mpi_t* x, *gcry_mpi_t* y, *gcry_mpi_t* z, *gcry_mpi_point_t* point) [Function]

Store the projective coordinates from *point* into the MPIs *x*, *y*, and *z*. If a coordinate is not required, NULL may be used for *x*, *y*, or *z*. The object *point* is then released. Using this function instead of gcry_mpi_point_get and gcry_mpi_point_release has the advantage of avoiding some extra memory allocations and copies.

gcry_mpi_point_t gcry_mpi_point_set (*gcry_mpi_point_t* point, *gcry_mpi_t* x, *gcry_mpi_t* y, *gcry_mpi_t* z) [Function]

Store the projective coordinates from *x*, *y*, and *z* into *point*. If a coordinate is given as NULL, the value 0 is used. If NULL is used for *point* a new point object is allocated and returned. Returns *point* or the newly allocated point object.

gcry_mpi_point_t gcry_mpi_point_snatch_set (*gcry_mpi_point_t* point, *gcry_mpi_t* x, *gcry_mpi_t* y, *gcry_mpi_t* z) [Function]

Store the projective coordinates from *x*, *y*, and *z* into *point*. If a coordinate is given as NULL, the value 0 is used. If NULL is used for *point* a new point object is allocated and returned. The MPIs *x*, *y*, and *z* are released. Using this function instead of gcry_mpi_point_set and 3 calls to gcry_mpi_release has the advantage of avoiding some extra memory allocations and copies. Returns *point* or the newly allocated point object.

gpg_error_t gcry_mpi_ec_new (*gcry_ctx_t* *r_ctx, *gcry_sexp_t* keyparam, *const char* *curvename) [Function]

Allocate a new context for elliptic curve operations. If *keyparam* is given it specifies the parameters of the curve (see [ecc_keyparam], page 36). If *curvename* is given in addition to *keyparam* and the key parameters do not include a named curve reference, the string *curvename* is used to fill in missing parameters. If only *curvename* is given, the context is initialized for this named curve.

If a parameter specifying a point (e.g. g or q) is not found, the parser looks for a non-encoded point by appending .x, .y, and .z to the parameter name and looking them all up to create a point. A parameter with the suffix .z is optional and defaults to 1.

On success the function returns 0 and stores the new context object at *r_ctx*; this object eventually needs to be released (see [gcry_ctx_release], page 89). On error the function stores NULL at *r_ctx* and returns an error code.

gcry_mpi_t gcry_mpi_ec_get_mpi (*const char* *name, *gcry_ctx_t* ctx, *int* copy) [Function]

Return the MPI with *name* from the context *ctx*. If not found NULL is returned. If the returned MPI may later be modified, it is suggested to pass 1 to *copy*, so that the function guarantees that a modifiable copy of the MPI is returned. If `0` is used for *copy*, this function may return a constant flagged MPI. In any case `gcry_mpi_release` needs to be called to release the result. For valid names [ecc_keyparam], page 36. If the public key `q` is requested but only the private key `d` is available, `q` will be recomputed on the fly. If a point parameter is requested it is returned as an uncompressed encoded point unless these special names are used:

q@eddsa Return an EdDSA style compressed point. This is only supported for Twisted Edwards curves.

`gcry_mpi_point_t gcry_mpi_ec_get_point` (*const char* `*name`, *gcry_ctx_t* `ctx`, *int* `copy`) [Function]

Return the point with *name* from the context *ctx*. If not found `NULL` is returned. If the returned MPI may later be modified, it is suggested to pass `1` to *copy*, so that the function guarantees that a modifiable copy of the MPI is returned. If `0` is used for *copy*, this function may return a constant flagged point. In any case `gcry_mpi_point_release` needs to be called to release the result. If the public key `q` is requested but only the private key `d` is available, `q` will be recomputed on the fly.

`gpg_error_t gcry_mpi_ec_set_mpi` (*const char* `*name`, *gcry_mpi_t* `newvalue`, *gcry_ctx_t* `ctx`) [Function]

Store the MPI *newvalue* at *name* into the context *ctx*. On success `0` is returned; on error an error code. Valid names are the MPI parameters of an elliptic curve (see [ecc_keyparam], page 36).

`gpg_error_t gcry_mpi_ec_set_point` (*const char* `*name`, *gcry_mpi_point_t* `newvalue`, *gcry_ctx_t* `ctx`) [Function]

Store the point *newvalue* at *name* into the context *ctx*. On success `0` is returned; on error an error code. Valid names are the point parameters of an elliptic curve (see [ecc_keyparam], page 36).

`gpg_err_code_t gcry_mpi_ec_decode_point` (*mpi_point_t* `result`, *gcry_mpi_t* `value`, *gcry_ctx_t* `ctx`) [Function]

Decode the point given as an MPI in *value* and store at *result*. To decide which encoding is used the function takes a context *ctx* which can be created with `gcry_mpi_ec_new`. If `NULL` is given for the context the function assumes a 0x04 prefixed uncompressed encoding. On error an error code is returned and *result* might be changed.

`int gcry_mpi_ec_get_affine` (*gcry_mpi_t* `x`, *gcry_mpi_t* `y`, *gcry_mpi_point_t* `point`, *gcry_ctx_t* `ctx`) [Function]

Compute the affine coordinates from the projective coordinates in *point* and store them into *x* and *y*. If one coordinate is not required, `NULL` may be passed to *x* or *y*. *ctx* is the context object which has been created using `gcry_mpi_ec_new`. Returns 0 on success or not 0 if *point* is at infinity.

Note that you can use `gcry_mpi_ec_set_point` with the value `GCRYMPI_CONST_ONE` for *z* to convert affine coordinates back into projective coordinates.

void **gcry_mpi_ec_dup** (*gcry_mpi_point_t* w, *gcry_mpi_point_t* u, *gcry_ctx_t* ctx) [Function]

Double the point *u* of the elliptic curve described by *ctx* and store the result into *w*.

void **gcry_mpi_ec_add** (*gcry_mpi_point_t* w, *gcry_mpi_point_t* u, *gcry_mpi_point_t* v, *gcry_ctx_t* ctx) [Function]

Add the points *u* and *v* of the elliptic curve described by *ctx* and store the result into *w*.

void **gcry_mpi_ec_sub** (*gcry_mpi_point_t* w, *gcry_mpi_point_t* u, *gcry_mpi_point_t* v, *gcry_ctx_t* ctx) [Function]

Subtracts the point *v* from the point *u* of the elliptic curve described by *ctx* and store the result into *w*. Only Twisted Edwards curves are supported for now.

void **gcry_mpi_ec_mul** (*gcry_mpi_point_t* w, *gcry_mpi_t* n, *gcry_mpi_point_t* u, *gcry_ctx_t* ctx) [Function]

Multiply the point *u* of the elliptic curve described by *ctx* by *n* and store the result into *w*.

int **gcry_mpi_ec_curve_point** (*gcry_mpi_point_t* point, *gcry_ctx_t* ctx) [Function]

Return true if *point* is on the elliptic curve described by *ctx*.

12.8 Miscellaneous

An MPI data type is allowed to be "misused" to store an arbitrary value. Two functions implement this kludge:

gcry_mpi_t **gcry_mpi_set_opaque** (*gcry_mpi_t* a, *void* *p, *unsigned int* nbits) [Function]

Store *nbits* of the value *p* points to in *a* and mark *a* as an opaque value (i.e. an value that can't be used for any math calculation and is only used to store an arbitrary bit pattern in *a*). Ownership of *p* is taken by this function and thus the user may not use dereference the passed value anymore. It is required that them memory referenced by *p* has been allocated in a way that `gcry_free` is able to release it.

WARNING: Never use an opaque MPI for actual math operations. The only valid functions are gcry_mpi_get_opaque and gcry_mpi_release. Use gcry_mpi_scan to convert a string of arbitrary bytes into an MPI.

gcry_mpi_t **gcry_mpi_set_opaque_copy** (*gcry_mpi_t* a, *const void* *p, *unsigned int* nbits) [Function]

Same as `gcry_mpi_set_opaque` but ownership of *p* is not taken instead a copy of *p* is used.

void * **gcry_mpi_get_opaque** (*gcry_mpi_t* a, *unsigned int* *nbits) [Function]

Return a pointer to an opaque value stored in *a* and return its size in *nbits*. Note that the returned pointer is still owned by *a* and that the function should never be used for an non-opaque MPI.

Each MPI has an associated set of flags for special purposes. The currently defined flags are:

`GCRYMPI_FLAG_SECURE`
: Setting this flag converts *a* into an MPI stored in "secure memory". Clearing this flag is not allowed.

`GCRYMPI_FLAG_OPAQUE`
: This is an internal flag, indicating the an opaque valuue and not an integer is stored. This is an read-only flag; it may not be set or cleared.

`GCRYMPI_FLAG_IMMUTABLE`
: If this flag is set, the MPI is marked as immutable. Setting or changing the value of that MPI is ignored and an error message is logged. The flag is sometimes useful for debugging.

`GCRYMPI_FLAG_CONST`
: If this flag is set, the MPI is marked as a constant and as immutable Setting or changing the value of that MPI is ignored and an error message is logged. Such an MPI will never be deallocated and may thus be used without copying. Note that using gcry_mpi_copy will return a copy of that constant with this and the immutable flag cleared. A few commonly used constants are pre-defined and accessible using the macros `GCRYMPI_CONST_ONE`, `GCRYMPI_CONST_TWO`, `GCRYMPI_CONST_THREE`, `GCRYMPI_CONST_FOUR`, and `GCRYMPI_CONST_EIGHT`.

`GCRYMPI_FLAG_USER1`
`GCRYMPI_FLAG_USER2`
`GCRYMPI_FLAG_USER3`
`GCRYMPI_FLAG_USER4`
: These flags are reserved for use by the application.

`void gcry_mpi_set_flag` (*gcry_mpi_t* `a`, *enum gcry_mpi_flag* `flag`) [Function]

Set the *flag* for the MPI *a*. The only allowed flags are `GCRYMPI_FLAG_SECURE`, `GCRYMPI_FLAG_IMMUTABLE`, and `GCRYMPI_FLAG_CONST`.

`void gcry_mpi_clear_flag` (*gcry_mpi_t* `a`, *enum gcry_mpi_flag* `flag`) [Function]

Clear *flag* for the multi-precision-integers *a*. The only allowed flag is `GCRYMPI_FLAG_IMMUTABLE` but only if `GCRYMPI_FLAG_CONST` is not set. If `GCRYMPI_FLAG_CONST` is set, clearing `GCRYMPI_FLAG_IMMUTABLE` will simply be ignored.

o

`int gcry_mpi_get_flag` (*gcry_mpi_t* `a`, *enum gcry_mpi_flag* `flag`) [Function]

Return true if *flag* is set for *a*.

To put a random value into an MPI, the following convenience function may be used:

`void gcry_mpi_randomize` (*gcry_mpi_t* `w`, *unsigned int* `nbits`, *enum gcry_random_level* `level`) [Function]

Set the multi-precision-integers *w* to a random non-negative number of *nbits*, using random data quality of level *level*. In case *nbits* is not a multiple of a byte, *nbits* is rounded up to the next byte boundary. When using a *level* of `GCRY_WEAK_RANDOM` this function makes use of `gcry_create_nonce`.

13 Prime numbers

13.1 Generation

`gcry_error_t gcry_prime_generate` (*gcry_mpi_t* ***prime***, *unsigned int* ***prime_bits***, *unsigned int* ***factor_bits***, *gcry_mpi_t* ****factors***, *gcry_prime_check_func_t* ***cb_func***, *void* ****cb_arg***, *gcry_random_level_t* ***random_level***, *unsigned int* ***flags***) [Function]

Generate a new prime number of *prime_bits* bits and store it in *prime*. If *factor_bits* is non-zero, one of the prime factors of (*prime* - 1) / 2 must be *factor_bits* bits long. If *factors* is non-zero, allocate a new, `NULL`-terminated array holding the prime factors and store it in *factors*. *flags* might be used to influence the prime number generation process.

`gcry_error_t gcry_prime_group_generator` (*gcry_mpi_t* ****r_g***, *gcry_mpi_t* ***prime***, *gcry_mpi_t* ****factors***, *gcry_mpi_t* ***start_g***) [Function]

Find a generator for *prime* where the factorization of (*prime*-1) is in the `NULL` terminated array *factors*. Return the generator as a newly allocated MPI in *r_g*. If *start_g* is not NULL, use this as the start for the search.

`void gcry_prime_release_factors` (*gcry_mpi_t* ****factors***) [Function]

Convenience function to release the *factors* array.

13.2 Checking

`gcry_error_t gcry_prime_check` (*gcry_mpi_t* ***p***, *unsigned int* ***flags***) [Function]

Check whether the number *p* is prime. Returns zero in case *p* is indeed a prime, returns `GPG_ERR_NO_PRIME` in case *p* is not a prime and a different error code in case something went horribly wrong.

14 Utilities

14.1 Memory allocation

`void * gcry_malloc (size_t n)` [Function]

This function tries to allocate *n* bytes of memory. On success it returns a pointer to the memory area, in an out-of-core condition, it returns NULL.

`void * gcry_malloc_secure (size_t n)` [Function]

Like `gcry_malloc`, but uses secure memory.

`void * gcry_calloc (size_t n, size_t m)` [Function]

This function allocates a cleared block of memory (i.e. initialized with zero bytes) long enough to contain a vector of *n* elements, each of size *m* bytes. On success it returns a pointer to the memory block; in an out-of-core condition, it returns NULL.

`void * gcry_calloc_secure (size_t n, size_t m)` [Function]

Like `gcry_calloc`, but uses secure memory.

`void * gcry_realloc (void *p, size_t n)` [Function]

This function tries to resize the memory area pointed to by *p* to *n* bytes. On success it returns a pointer to the new memory area, in an out-of-core condition, it returns NULL. Depending on whether the memory pointed to by *p* is secure memory or not, gcry_realloc tries to use secure memory as well.

`void gcry_free (void *p)` [Function]

Release the memory area pointed to by *p*.

14.2 Context management

Some function make use of a context object. As of now there are only a few math functions. However, future versions of Libgcrypt may make more use of this context object.

`gcry_ctx_t` [Data type]

This type is used to refer to the general purpose context object.

`void gcry_ctx_release (gcry_ctx_t ctx)` [Function]

Release the context object *ctx* and all associated resources. A `NULL` passed as *ctx* is ignored.

14.3 Buffer description

To help hashing non-contiguous areas of memory a general purpose data type is defined:

`gcry_buffer_t` [Data type]

This type is a structure to describe a buffer. The user should make sure that this structure is initialized to zero. The available fields of this structure are:

`.size` This is either 0 for no information available or indicates the allocated length of the buffer.

`.off` This is the offset into the buffer.

`.len` This is the valid length of the buffer starting at `.off`.

`.data` This is the address of the buffer.

14.4 How to return Libgcrypt's configuration.

Although `GCRYCTL_PRINT_CONFIG` can be used to print configuration options, it is sometimes necessary to check them in a program. This can be accomplished by using this function:

`char * gcry_get_config` (*int* `mode`, *const char* *`what`) [Function]

This function returns a malloced string with colon delimited configure options. With a value of 0 for *mode* this string resembles the output of `GCRYCTL_PRINT_CONFIG`. However, if *what* is not NULL, only the line where the first field (e.g. "cpu-arch") matches *what* is returned.

Other values than 0 for *mode* are not defined. The caller shall free the string using `gcry_free`. On error NULL is returned and ERRNO is set; if a value for WHAT is unknow ERRNO will be set to 0.

15 Tools

15.1 A HMAC-SHA-256 tool

This is a standalone HMAC-SHA-256 implementation used to compute an HMAC-SHA-256 message authentication code. The tool has originally been developed as a second implementation for Libgcrypt to allow comparing against the primary implementation and to be used for internal consistency checks. It should not be used for sensitive data because no mechanisms to clear the stack etc are used.

The code has been written in a highly portable manner and requires only a few standard definitions to be provided in a config.h file.

`hmac256` is commonly invoked as

```
hmac256 "This is my key" foo.txt
```

This compute the MAC on the file `foo.txt` using the key given on the command line.

`hmac256` understands these options:

`--binary` Print the MAC as a binary string. The default is to print the MAC encoded has lower case hex digits.

`--version`
Print version of the program and exit.

16 Configuration files and environment variables

This chapter describes which files and environment variables can be used to change the behaviour of Libgcrypt.

The environment variables considered by Libgcrypt are:

`GCRYPT_BARRETT`
: By setting this variable to any value a different algorithm for modular reduction is used for ECC.

`GCRYPT_RNDUNIX_DBG`
`GCRYPT_RNDUNIX_DBGALL`
: These two environment variables are used to enable debug output for the rndunix entropy gatherer, which is used on systems lacking a /dev/random device. The value of `GCRYPT_RNDUNIX_DBG` is a file name or `-` for stdout. Debug output is the written to this file. By setting `GCRYPT_RNDUNIX_DBGALL` to any value the debug output will be more verbose.

`GCRYPT_RNDW32_NOPERF`
: Setting this environment variable on Windows to any value disables the use of performance data (`HKEY_PERFORMANCE_DATA`) as source for entropy. On some older Windows systems this could help to speed up the creation of random numbers but also decreases the amount of data used to init the random number generator.

`GCRYPT_RNDW32_DBG`
: Setting the value of this variable to a positive integer logs information about the Windows entropy gatherer using the standard log interface.

`HOME`
: This is used to locate the socket to connect to the EGD random daemon. The EGD can be used on system without a /dev/random to speed up the random number generator. It is not needed on the majority of today's operating systems and support for EGD requires the use of a configure option at build time.

The files which Libgcrypt uses to retrieve system information and the files which can be created by the user to modify Libgcrypt's behavior are:

`/etc/gcrypt/hwf.deny`
: This file can be used to disable the use of hardware based optimizations, see [hardware features], page 7.

`/etc/gcrypt/random.conf`
: This file can be used to globally change parameters of the random generator. The file is a simple text file where empty lines and lines with the first non white-space character being '#' are ignored. Supported options are

  `disable-jent`
  : Disable the use of the jitter based entropy generator.

  `only-urandom`
  : Always use the non-blocking /dev/urandom or the respective system call instead of the blocking /dev/random. If Libgcrypt is used

early in the boot process of the system, this option should only be used if the system also supports the getrandom system call.

`/etc/gcrypt/fips_enabled`
`/proc/sys/crypto/fips_enabled`
: On Linux these files are used to enable FIPS mode, see [enabling fips mode], page 7.

`/proc/cpuinfo`
`/proc/self/auxv`
: On Linux running on the ARM architecture, these files are used to read hardware capabilities of the CPU.

17 Architecture

This chapter describes the internal architecture of Libgcrypt.

Libgcrypt is a function library written in ISO C-90. Any compliant compiler should be able to build Libgcrypt as long as the target is either a POSIX platform or compatible to the API used by Windows NT. Provisions have been take so that the library can be directly used from C++ applications; however building with a C++ compiler is not supported.

Building Libgcrypt is done by using the common `./configure && make` approach. The configure command is included in the source distribution and as a portable shell script it works on any Unix-alike system. The result of running the configure script are a C header file (`config.h`), customized Makefiles, the setup of symbolic links and a few other things. After that the make tool builds and optionally installs the library and the documentation. See the files `INSTALL` and `README` in the source distribution on how to do this.

Libgcrypt is developed using a Subversion[1] repository. Although all released versions are tagged in this repository, they should not be used to build production versions of Libgcrypt. Instead released tarballs should be used. These tarballs are available from several places with the master copy at '`ftp://ftp.gnupg.org/gcrypt/libgcrypt/`'. Announcements of new releases are posted to the '`gnupg-announce@gnupg.org`' mailing list[2].

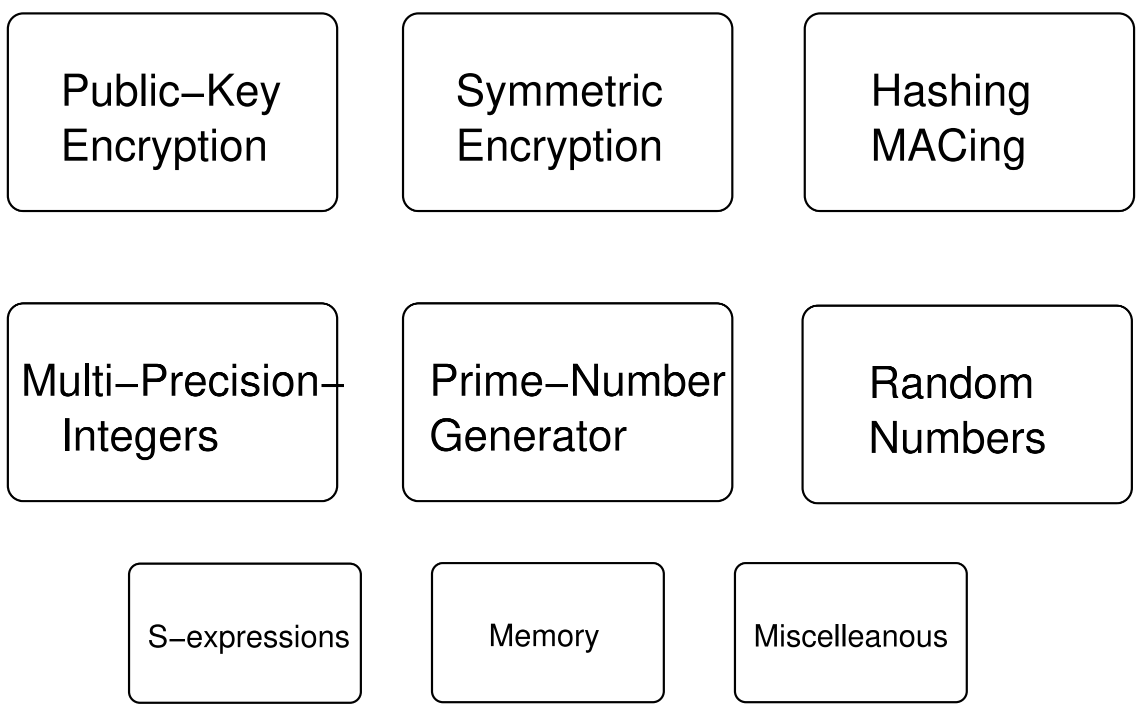

Figure 17.1: Libgcrypt subsystems

Libgcrypt consists of several subsystems (see Figure 17.1) and all these subsystems provide a public API; this includes the helper subsystems like the one for S-expressions. The

[1] A version control system available for many platforms

[2] See `http://www.gnupg.org/documentation/mailing-lists.en.html` for details.

API style depends on the subsystem; in general an open-use-close approach is implemented. The open returns a handle to a context used for all further operations on this handle, several functions may then be used on this handle and a final close function releases all resources associated with the handle.

17.1 Public-Key Architecture

Because public key cryptography is almost always used to process small amounts of data (hash values or session keys), the interface is not implemented using the open-use-close paradigm, but with single self-contained functions. Due to the wide variety of parameters required by different algorithms S-expressions, as flexible way to convey these parameters, are used. There is a set of helper functions to work with these S-expressions.

Aside of functions to register new algorithms, map algorithms names to algorithms identifiers and to lookup properties of a key, the following main functions are available:

`gcry_pk_encrypt`
: Encrypt data using a public key.

`gcry_pk_decrypt`
: Decrypt data using a private key.

`gcry_pk_sign`
: Sign data using a private key.

`gcry_pk_verify`
: Verify that a signature matches the data.

`gcry_pk_testkey`
: Perform a consistency over a public or private key.

`gcry_pk_genkey`
: Create a new public/private key pair.

All these functions lookup the module implementing the algorithm and pass the actual work to that module. The parsing of the S-expression input and the construction of S-expression for the return values is done by the high level code (`cipher/pubkey.c`). Thus the internal interface between the algorithm modules and the high level functions passes data in a custom format.

By default Libgcrypt uses a blinding technique for RSA decryption to mitigate real world timing attacks over a network: Instead of using the RSA decryption directly, a blinded value $y = xr^e \bmod n$ is decrypted and the unblinded value $x' = y'r^{-1} \bmod n$ returned. The blinding value r is a random value with the size of the modulus n and generated with `GCRY_WEAK_RANDOM` random level.

The algorithm used for RSA and DSA key generation depends on whether Libgcrypt is operated in standard or in FIPS mode. In standard mode an algorithm based on the Lim-Lee prime number generator is used. In FIPS mode RSA keys are generated as specified in ANSI X9.31 (1998) and DSA keys as specified in FIPS 186-2.

17.2 Symmetric Encryption Subsystem Architecture

The interface to work with symmetric encryption algorithms is made up of functions from the `gcry_cipher_` name space. The implementation follows the open-use-close paradigm and uses registered algorithm modules for the actual work. Unless a module implements optimized cipher mode implementations, the high level code (`cipher/cipher.c`) implements the modes and calls the core algorithm functions to process each block.

The most important functions are:

`gcry_cipher_open`
: Create a new instance to encrypt or decrypt using a specified algorithm and mode.

`gcry_cipher_close`
: Release an instance.

`gcry_cipher_setkey`
: Set a key to be used for encryption or decryption.

`gcry_cipher_setiv`
: Set an initialization vector to be used for encryption or decryption.

`gcry_cipher_encrypt`
`gcry_cipher_decrypt`
: Encrypt or decrypt data. These functions may be called with arbitrary amounts of data and as often as needed to encrypt or decrypt all data.

  There is no strict alignment requirements for data, but the best performance can be archived if data is aligned to cacheline boundary.

There are also functions to query properties of algorithms or context, like block length, key length, map names or to enable features like padding methods.

17.3 Hashing and MACing Subsystem Architecture

The interface to work with message digests and CRC algorithms is made up of functions from the `gcry_md_` name space. The implementation follows the open-use-close paradigm and uses registered algorithm modules for the actual work. Although CRC algorithms are not considered cryptographic hash algorithms, they share enough properties so that it makes sense to handle them in the same way. It is possible to use several algorithms at once with one context and thus compute them all on the same data.

The most important functions are:

`gcry_md_open`
: Create a new message digest instance and optionally enable one algorithm. A flag may be used to turn the message digest algorithm into a HMAC algorithm.

`gcry_md_enable`
: Enable an additional algorithm for the instance.

`gcry_md_setkey`
: Set the key for the MAC.

`gcry_md_write`
: Pass more data for computing the message digest to an instance.

  There is no strict alignment requirements for data, but the best performance can be archived if data is aligned to cacheline boundary.

`gcry_md_putc`
: Buffered version of `gcry_md_write` implemented as a macro.

`gcry_md_read`
: Finalize the computation of the message digest or HMAC and return the result.

`gcry_md_close`
: Release an instance

`gcry_md_hash_buffer`
: Convenience function to directly compute a message digest over a memory buffer without the need to create an instance first.

There are also functions to query properties of algorithms or the instance, like enabled algorithms, digest length, map algorithm names. it is also possible to reset an instance or to copy the current state of an instance at any time. Debug functions to write the hashed data to files are available as well.

17.4 Multi-Precision-Integer Subsystem Architecture

The implementation of Libgcrypt's big integer computation code is based on an old release of GNU Multi-Precision Library (GMP). The decision not to use the GMP library directly was due to stalled development at that time and due to security requirements which could not be provided by the code in GMP. As GMP does, Libgcrypt provides high performance assembler implementations of low level code for several CPUS to gain much better performance than with a generic C implementation.

Major features of Libgcrypt's multi-precision-integer code compared to GMP are:

- Avoidance of stack based allocations to allow protection against swapping out of sensitive data and for easy zeroing of sensitive intermediate results.
- Optional use of secure memory and tracking of its use so that results are also put into secure memory.
- MPIs are identified by a handle (implemented as a pointer) to give better control over allocations and to augment them with extra properties like opaque data.
- Removal of unnecessary code to reduce complexity.
- Functions specialized for public key cryptography.

17.5 Prime-Number-Generator Subsystem Architecture

Libgcrypt provides an interface to its prime number generator. These functions make use of the internal prime number generator which is required for the generation for public key key pairs. The plain prime checking function is exported as well.

The generation of random prime numbers is based on the Lim and Lee algorithm to create practically save primes.[3] This algorithm creates a pool of smaller primes, select a few of them to create candidate primes of the form $2 * p_0 * p_1 * ... * p_n + 1$, tests the candidate for primality and permutates the pool until a prime has been found. It is possible to clamp one of the small primes to a certain size to help DSA style algorithms. Because most of the small primes in the pool are not used for the resulting prime number, they are saved for later use (see `save_pool_prime` and `get_pool_prime` in `cipher/primegen.c`). The prime generator optionally supports the finding of an appropriate generator.

The primality test works in three steps:

1. The standard sieve algorithm using the primes up to 4999 is used as a quick first check.
2. A Fermat test filters out almost all non-primes.
3. A 5 round Rabin-Miller test is finally used. The first round uses a witness of 2, whereas the next rounds use a random witness.

To support the generation of RSA and DSA keys in FIPS mode according to X9.31 and FIPS 186-2, Libgcrypt implements two additional prime generation functions: `_gcry_derive_x931_prime` and `_gcry_generate_fips186_2_prime`. These functions are internal and not available through the public API.

17.6 Random-Number Subsystem Architecture

Libgcrypt provides 3 levels or random quality: The level `GCRY_VERY_STRONG_RANDOM` usually used for key generation, the level `GCRY_STRONG_RANDOM` for all other strong random requirements and the function `gcry_create_nonce` which is used for weaker usages like nonces. There is also a level `GCRY_WEAK_RANDOM` which in general maps to `GCRY_STRONG_RANDOM` except when used with the function `gcry_mpi_randomize`, where it randomizes an multi-precision-integer using the `gcry_create_nonce` function.

There are two distinct random generators available:

- The Continuously Seeded Pseudo Random Number Generator (CSPRNG), which is based on the classic GnuPG derived big pool implementation. Implemented in `random/random-csprng.c` and used by default.
- A FIPS approved ANSI X9.31 PRNG using AES with a 128 bit key. Implemented in `random/random-fips.c` and used if Libgcrypt is in FIPS mode.

Both generators make use of so-called entropy gathering modules:

rndlinux
: Uses the operating system provided `/dev/random` and `/dev/urandom` devices. The `/dev/gcrypt/random.conf` config option `only-urandom` can be used to inhibit the use of the blocking `/dev/random` device.

rndunix
: Runs several operating system commands to collect entropy from sources like virtual machine and process statistics. It is a kind of poor-man's `/dev/random` implementation. It is not available in FIPS mode.

[3] Chae Hoon Lim and Pil Joong Lee. A key recovery attack on discrete log-based schemes using a prime order subgroup. In Burton S. Kaliski Jr., editor, Advances in Cryptology: Crypto '97, pages 249-263, Berlin / Heidelberg / New York, 1997. Springer-Verlag. Described on page 260.

rndegd Uses the operating system provided Entropy Gathering Daemon (EGD). The EGD basically uses the same algorithms as rndunix does. However as a system daemon it keeps on running and thus can serve several processes requiring entropy input and does not waste collected entropy if the application does not need all the collected entropy. It is not available in FIPS mode.

rndw32 Targeted for the Microsoft Windows OS. It uses certain properties of that system and is the only gathering module available for that OS.

rndhw Extra module to collect additional entropy by utilizing a hardware random number generator. As of now the supported hardware RNG is the Padlock engine of VIA (Centaur) CPUs and x86 CPUs with the RDRAND instruction. It is not available in FIPS mode.

rndjent Extra module to collect additional entropy using a CPU jitter based approach. This is only used on X86 hardware where the RDTSC opcode is available. The `/dev/gcrypt/random.conf` config option `disable-jent` can be used to inhibit the use of this module.

17.6.1 Description of the CSPRNG

This random number generator is loosely modelled after the one described in Peter Gutmann's paper: "Software Generation of Practically Strong Random Numbers".[4]

A pool of 600 bytes is used and mixed using the core SHA-1 hash transform function. Several extra features are used to make the robust against a wide variety of attacks and to protect against failures of subsystems. The state of the generator may be saved to a file and initially seed form a file.

Depending on how Libgcrypt was build the generator is able to select the best working entropy gathering module. It makes use of the slow and fast collection methods and requires the pool to initially seeded form the slow gatherer or a seed file. An entropy estimation is used to mix in enough data from the gather modules before returning the actual random output. Process fork detection and protection is implemented.

The implementation of the nonce generator (for `gcry_create_nonce`) is a straightforward repeated hash design: A 28 byte buffer is initially seeded with the PID and the time in seconds in the first 20 bytes and with 8 bytes of random taken from the `GCRY_STRONG_RANDOM` generator. Random numbers are then created by hashing all the 28 bytes with SHA-1 and saving that again in the first 20 bytes. The hash is also returned as result.

17.6.2 Description of the FIPS X9.31 PRNG

The core of this deterministic random number generator is implemented according to the document "NIST-Recommended Random Number Generator Based on ANSI X9.31 Appendix A.2.4 Using the 3-Key Triple DES and AES Algorithms", dated 2005-01-31. This implementation uses the AES variant.

The generator is based on contexts to utilize the same core functions for all random levels as required by the high-level interface. All random generators return their data in 128 bit blocks. If the caller requests less bits, the extra bits are not used. The key for each

[4] Also described in chapter 6 of his book "Cryptographic Security Architecture", New York, 2004, ISBN 0-387-95387-6.

generator is only set once at the first time a generator context is used. The seed value is set along with the key and again after 1000 output blocks.

On Unix like systems the GCRY_VERY_STRONG_RANDOM and GCRY_STRONG_RANDOM generators are keyed and seeded using the rndlinux module with the /dev/random device. Thus these generators may block until the OS kernel has collected enough entropy. When used with Microsoft Windows the rndw32 module is used instead.

The generator used for gcry_create_nonce is keyed and seeded from the GCRY_STRONG_RANDOM generator. Thus is may also block if the GCRY_STRONG_RANDOM generator has not yet been used before and thus gets initialized on the first use by gcry_create_nonce. This special treatment is justified by the weaker requirements for a nonce generator and to save precious kernel entropy for use by the "real" random generators.

A self-test facility uses a separate context to check the functionality of the core X9.31 functions using a known answers test. During runtime each output block is compared to the previous one to detect a stuck generator.

The DT value for the generator is made up of the current time down to microseconds (if available) and a free running 64 bit counter. When used with the test context the DT value is taken from the context and incremented on each use.

Appendix A Description of the Self-Tests

In addition to the build time regression test suite, Libgcrypt implements self-tests to be performed at runtime. Which self-tests are actually used depends on the mode Libgcrypt is used in. In standard mode a limited set of self-tests is run at the time an algorithm is first used. Note that not all algorithms feature a self-test in standard mode. The `GCRYCTL_SELFTEST` control command may be used to run all implemented self-tests at any time; this will even run more tests than those run in FIPS mode.

If any of the self-tests fails, the library immediately returns an error code to the caller. If Libgcrypt is in FIPS mode the self-tests will be performed within the "Self-Test" state and any failure puts the library into the "Error" state.

A.1 Power-Up Tests

Power-up tests are only performed if Libgcrypt is in FIPS mode.

A.1.1 Symmetric Cipher Algorithm Power-Up Tests

The following symmetric encryption algorithm tests are run during power-up:

3DES
: To test the 3DES 3-key EDE encryption in ECB mode these tests are run:

1. A known answer test is run on a 64 bit test vector processed by 64 rounds of Single-DES block encryption and decryption using a key changed with each round.
2. A known answer test is run on a 64 bit test vector processed by 16 rounds of 2-key and 3-key Triple-DES block encryption and decryptions using a key changed with each round.
3. 10 known answer tests using 3-key Triple-DES EDE encryption, comparing the ciphertext to the known value, then running a decryption and comparing it to the initial plaintext.

(`cipher/des.c:selftest`)

AES-128
: A known answer tests is run using one test vector and one test key with AES in ECB mode. (`cipher/rijndael.c:selftest_basic_128`)

AES-192
: A known answer tests is run using one test vector and one test key with AES in ECB mode. (`cipher/rijndael.c:selftest_basic_192`)

AES-256
: A known answer tests is run using one test vector and one test key with AES in ECB mode. (`cipher/rijndael.c:selftest_basic_256`)

A.1.2 Hash Algorithm Power-Up Tests

The following hash algorithm tests are run during power-up:

SHA-1
: A known answer test using the string `"abc"` is run. (`cipher/sha1.c:selftests_sha1`)

SHA-224
: A known answer test using the string `"abc"` is run. (`cipher/sha256.c:selftests_sha224`)

SHA-256 A known answer test using the string `"abc"` is run. (`cipher/sha256.c:selftests_sha256`)

SHA-384 A known answer test using the string `"abc"` is run. (`cipher/sha512.c:selftests_sha384`)

SHA-512 A known answer test using the string `"abc"` is run. (`cipher/sha512.c:selftests_sha512`)

A.1.3 MAC Algorithm Power-Up Tests

The following MAC algorithm tests are run during power-up:

HMAC SHA-1
: A known answer test using 9 byte of data and a 64 byte key is run. (`cipher/hmac-tests.c:selftests_sha1`)

HMAC SHA-224
: A known answer test using 28 byte of data and a 4 byte key is run. (`cipher/hmac-tests.c:selftests_sha224`)

HMAC SHA-256
: A known answer test using 28 byte of data and a 4 byte key is run. (`cipher/hmac-tests.c:selftests_sha256`)

HMAC SHA-384
: A known answer test using 28 byte of data and a 4 byte key is run. (`cipher/hmac-tests.c:selftests_sha384`)

HMAC SHA-512
: A known answer test using 28 byte of data and a 4 byte key is run. (`cipher/hmac-tests.c:selftests_sha512`)

A.1.4 Random Number Power-Up Test

The DRNG is tested during power-up this way:

1. Requesting one block of random using the public interface to check general working and the duplicated block detection.
2. 3 know answer tests using pre-defined keys, seed and initial DT values. For each test 3 blocks of 16 bytes are requested and compared to the expected result. The DT value is incremented for each block.

A.1.5 Public Key Algorithm Power-Up Tests

The public key algorithms are tested during power-up:

RSA A pre-defined 1024 bit RSA key is used and these tests are run in turn:

1. Conversion of S-expression to internal format. (`cipher/rsa.c:selftests_rsa`)
2. Private key consistency check. (`cipher/rsa.c:selftests_rsa`)
3. A pre-defined 20 byte value is signed with PKCS#1 padding for SHA-1. The result is verified using the public key against the original data and against modified data. (`cipher/rsa.c:selftest_sign_1024`)

4. A 1000 bit random value is encrypted and checked that it does not match the original random value. The encrypted result is then decrypted and checked that it matches the original random value. (`cipher/rsa.c:selftest_encr_1024`)

DSA
: A pre-defined 1024 bit DSA key is used and these tests are run in turn:

1. Conversion of S-expression to internal format. (`cipher/dsa.c:selftests_dsa`)
2. Private key consistency check. (`cipher/dsa.c:selftests_dsa`)
3. A pre-defined 20 byte value is signed with PKCS#1 padding for SHA-1. The result is verified using the public key against the original data and against modified data. (`cipher/dsa.c:selftest_sign_1024`)

A.1.6 Integrity Power-Up Tests

The integrity of the Libgcrypt is tested during power-up but only if checking has been enabled at build time. The check works by computing a HMAC SHA-256 checksum over the file used to load Libgcrypt into memory. That checksum is compared against a checksum stored in a file of the same name but with a single dot as a prefix and a suffix of `.hmac`.

A.1.7 Critical Functions Power-Up Tests

The 3DES weak key detection is tested during power-up by calling the detection function with keys taken from a table listening all weak keys. The table itself is protected using a SHA-1 hash. (`cipher/des.c:selftest`)

A.2 Conditional Tests

The conditional tests are performed if a certain condition is met. This may occur at any time; the library does not necessary enter the "Self-Test" state to run these tests but will transit to the "Error" state if a test failed.

A.2.1 Key-Pair Generation Tests

After an asymmetric key-pair has been generated, Libgcrypt runs a pair-wise consistency tests on the generated key. On failure the generated key is not used, an error code is returned and, if in FIPS mode, the library is put into the "Error" state.

RSA
: The test uses a random number 64 bits less the size of the modulus as plaintext and runs an encryption and decryption operation in turn. The encrypted value is checked to not match the plaintext and the result of the decryption is checked to match the plaintext.

  A new random number of the same size is generated, signed and verified to test the correctness of the signing operation. As a second signing test, the signature is modified by incrementing its value and then verified with the expected result that the verification fails. (`cipher/rsa.c:test_keys`)

DSA
: The test uses a random number of the size of the Q parameter to create a signature and then checks that the signature verifies. As a second signing test, the data is modified by incrementing its value and then verified against the

signature with the expected result that the verification fails. (`cipher/dsa.c:test_keys`)

A.2.2 Software Load Tests

No code is loaded at runtime.

A.2.3 Manual Key Entry Tests

A manual key entry feature is not implemented in Libgcrypt.

A.2.4 Continuous RNG Tests

The continuous random number test is only used in FIPS mode. The RNG generates blocks of 128 bit size; the first block generated per context is saved in the context and another block is generated to be returned to the caller. Each block is compared against the saved block and then stored in the context. If a duplicated block is detected an error is signaled and the library is put into the “Fatal-Error” state. (`random/random-fips.c:x931_aes_driver`)

A.3 Application Requested Tests

The application may requests tests at any time by means of the `GCRYCTL_SELFTEST` control command. Note that using these tests is not FIPS conform: Although Libgcrypt rejects all application requests for services while running self-tests, it does not ensure that no other operations of Libgcrypt are still being executed. Thus, in FIPS mode an application requesting self-tests needs to power-cycle Libgcrypt instead.

When self-tests are requested, Libgcrypt runs all the tests it does during power-up as well as a few extra checks as described below.

A.3.1 Symmetric Cipher Algorithm Tests

The following symmetric encryption algorithm tests are run in addition to the power-up tests:

AES-128
: A known answer tests with test vectors taken from NIST SP800-38a and using the high level functions is run for block modes CFB and OFB.

A.3.2 Hash Algorithm Tests

The following hash algorithm tests are run in addition to the power-up tests:

SHA-1
SHA-224
SHA-256

1. A known answer test using a 56 byte string is run.
2. A known answer test using a string of one million letters "a" is run.

(`cipher/sha1.c:selftests_sha1`, `cipher/sha256.c:selftests_sha224`, `cipher/sha256.c:selftests_sha256`)

SHA-384

SHA-512

1. A known answer test using a 112 byte string is run.

2. A known answer test using a string of one million letters "a" is run.

(cipher/sha512.c:selftests_sha384, cipher/sha512.c:selftests_sha512)

A.3.3 MAC Algorithm Tests

The following MAC algorithm tests are run in addition to the power-up tests:

HMAC SHA-1

1. A known answer test using 9 byte of data and a 20 byte key is run.
2. A known answer test using 9 byte of data and a 100 byte key is run.
3. A known answer test using 9 byte of data and a 49 byte key is run.

(cipher/hmac-tests.c:selftests_sha1)

HMAC SHA-224
HMAC SHA-256
HMAC SHA-384
HMAC SHA-512

1. A known answer test using 9 byte of data and a 20 byte key is run.
2. A known answer test using 50 byte of data and a 20 byte key is run.
3. A known answer test using 50 byte of data and a 26 byte key is run.
4. A known answer test using 54 byte of data and a 131 byte key is run.
5. A known answer test using 152 byte of data and a 131 byte key is run.

(cipher/hmac-tests.c:selftests_sha224, cipher/hmac-tests.c:selftests_sha256, cipher/hmac-tests.c:selftests_sha384, cipher/hmac-tests.c:selftests_sha512)

Appendix B Description of the FIPS Mode

This appendix gives detailed information pertaining to the FIPS mode. In particular, the changes to the standard mode and the finite state machine are described. The self-tests required in this mode are described in the appendix on self-tests.

B.1 Restrictions in FIPS Mode

If Libgcrypt is used in FIPS mode these restrictions are effective:

- The cryptographic algorithms are restricted to this list:

  GCRY_CIPHER_3DES
  : 3 key EDE Triple-DES symmetric encryption.

  GCRY_CIPHER_AES128
  : AES 128 bit symmetric encryption.

  GCRY_CIPHER_AES192
  : AES 192 bit symmetric encryption.

  GCRY_CIPHER_AES256
  : AES 256 bit symmetric encryption.

  GCRY_MD_SHA1
  : SHA-1 message digest.

  GCRY_MD_SHA224
  : SHA-224 message digest.

  GCRY_MD_SHA256
  : SHA-256 message digest.

  GCRY_MD_SHA384
  : SHA-384 message digest.

  GCRY_MD_SHA512
  : SHA-512 message digest.

  GCRY_MD_SHA1,GCRY_MD_FLAG_HMAC
  : HMAC using a SHA-1 message digest.

  GCRY_MD_SHA224,GCRY_MD_FLAG_HMAC
  : HMAC using a SHA-224 message digest.

  GCRY_MD_SHA256,GCRY_MD_FLAG_HMAC
  : HMAC using a SHA-256 message digest.

  GCRY_MD_SHA384,GCRY_MD_FLAG_HMAC
  : HMAC using a SHA-384 message digest.

  GCRY_MD_SHA512,GCRY_MD_FLAG_HMAC
  : HMAC using a SHA-512 message digest.

  GCRY_PK_RSA
  : RSA encryption and signing.

GCRY_PK_DSA
    DSA signing.

Note that the CRC algorithms are not considered cryptographic algorithms and thus are in addition available.

- RSA key generation refuses to create a key with a keysize of less than 1024 bits.
- DSA key generation refuses to create a key with a keysize other than 1024 bits.
- The `transient-key` flag for RSA and DSA key generation is ignored.
- Support for the VIA Padlock engine is disabled.
- FIPS mode may only be used on systems with a /dev/random device. Switching into FIPS mode on other systems will fail at runtime.
- Saving and loading a random seed file is ignored.
- An X9.31 style random number generator is used in place of the large-pool-CSPRNG generator.
- The command `GCRYCTL_ENABLE_QUICK_RANDOM` is ignored.
- Message digest debugging is disabled.
- All debug output related to cryptographic data is suppressed.
- On-the-fly self-tests are not performed, instead self-tests are run before entering operational state.
- The function `gcry_set_allocation_handler` may not be used. If it is used Libgcrypt disables FIPS mode unless Enforced FIPS mode is enabled, in which case Libgcrypt will enter the error state.
- The digest algorithm MD5 may not be used. If it is used Libgcrypt disables FIPS mode unless Enforced FIPS mode is enabled, in which case Libgcrypt will enter the error state.
- In Enforced FIPS mode the command `GCRYCTL_DISABLE_SECMEM` is ignored. In standard FIPS mode it disables FIPS mode.
- A handler set by `gcry_set_outofcore_handler` is ignored.
- A handler set by `gcry_set_fatalerror_handler` is ignored.

Note that when we speak about disabling FIPS mode, it merely means that the function `gcry_fips_mode_active` returns false; it does not mean that any non FIPS algorithms are allowed.

B.2 FIPS Finite State Machine

The FIPS mode of libgcrypt implements a finite state machine (FSM) using 8 states (see Table B.1) and checks at runtime that only valid transitions (see Table B.2) may happen.

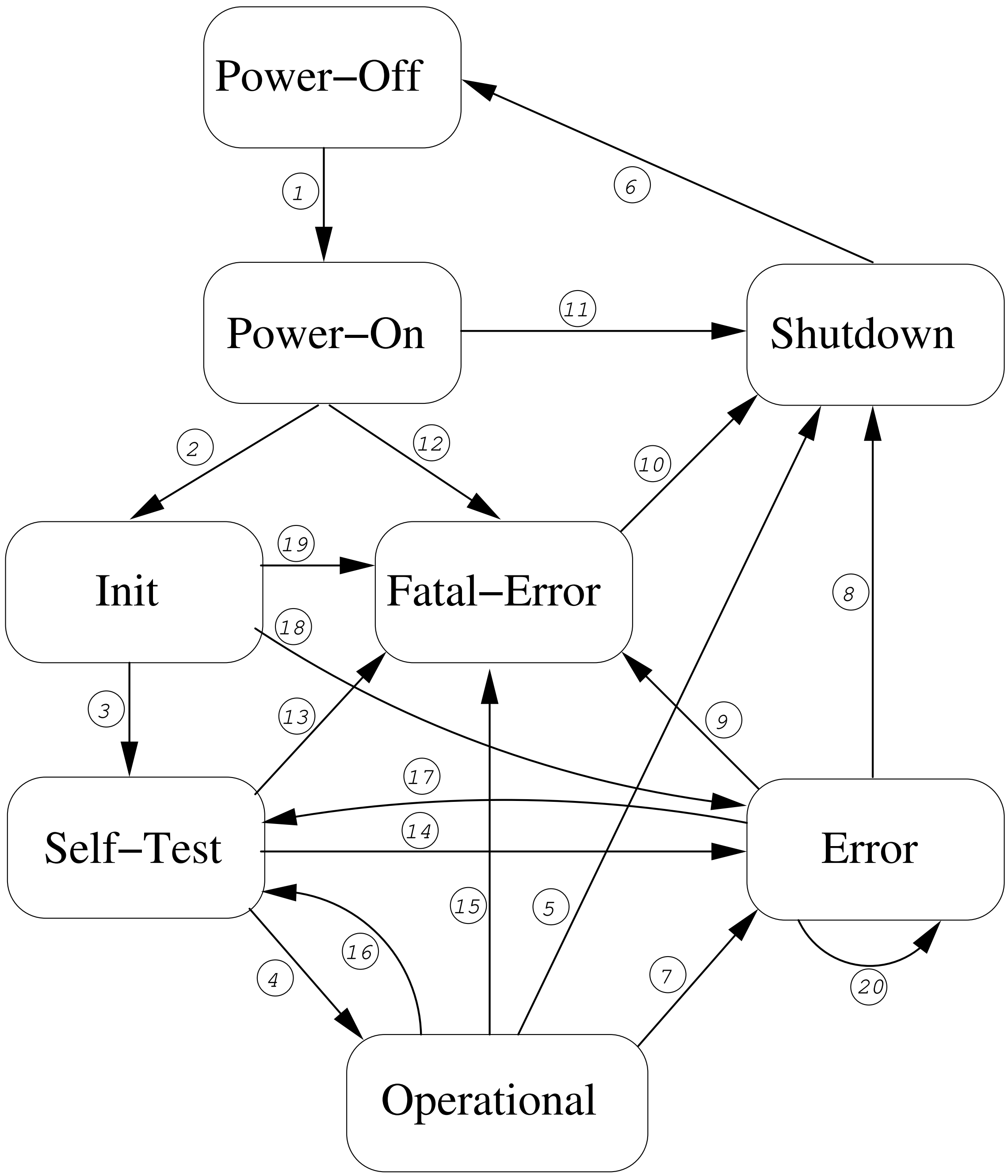

Figure B.1: FIPS mode state diagram

States used by the FIPS FSM:

Power-Off
: Libgcrypt is not runtime linked to another application. This usually means that the library is not loaded into main memory. This state is documentation only.

Power-On
: Libgcrypt is loaded into memory and API calls may be made. Compiler introduced constructor functions may be run. Note that Libgcrypt does not implement any arbitrary constructor functions to be called by the operating system

Init
: The Libgcrypt initialization functions are performed and the library has not yet run any self-test.

Self-Test
: Libgcrypt is performing self-tests.

Operational
: Libgcrypt is in the operational state and all interfaces may be used.

Error
: Libgrypt is in the error state. When calling any FIPS relevant interfaces they either return an error (`GPG_ERR_NOT_OPERATIONAL`) or put Libgcrypt into the Fatal-Error state and won't return.

Fatal-Error
: Libgcrypt is in a non-recoverable error state and will automatically transit into the Shutdown state.

Shutdown
: Libgcrypt is about to be terminated and removed from the memory. The application may at this point still running cleanup handlers.

Table B.1: FIPS mode states

The valid state transitions (see Figure B.1) are:

1	Power-Off to Power-On is implicitly done by the OS loading Libgcrypt as a shared library and having it linked to an application.
2	Power-On to Init is triggered by the application calling the Libgcrypt initialization function `gcry_check_version`.
3	Init to Self-Test is either triggered by a dedicated API call or implicit by invoking a libgrypt service controlled by the FSM.
4	Self-Test to Operational is triggered after all self-tests passed successfully.
5	Operational to Shutdown is an artificial state without any direct action in Libgcrypt. When reaching the Shutdown state the library is deinitialized and can't return to any other state again.
6	Shutdown to Power-off is the process of removing Libgcrypt from the computer's memory. For obvious reasons the Power-Off state can't be represented within Libgcrypt and thus this transition is for documentation only.
7	Operational to Error is triggered if Libgcrypt detected an application error which can't be returned to the caller but still allows Libgcrypt to properly run. In the Error state all FIPS relevant interfaces return an error code.
8	Error to Shutdown is similar to the Operational to Shutdown transition (5).
9	Error to Fatal-Error is triggered if Libgrypt detects an fatal error while already being in Error state.
10	Fatal-Error to Shutdown is automatically entered by Libgcrypt after having reported the error.
11	Power-On to Shutdown is an artificial state to document that Libgcrypt has not ye been initialized but the process is about to terminate.
12	Power-On to Fatal-Error will be triggered if certain Libgcrypt functions are used without having reached the Init state.
13	Self-Test to Fatal-Error is triggered by severe errors in Libgcrypt while running self-tests.
14	Self-Test to Error is triggered by a failed self-test.
15	Operational to Fatal-Error is triggered if Libcrypt encountered a non-recoverable error.
16	Operational to Self-Test is triggered if the application requested to run the self-tests again.
17	Error to Self-Test is triggered if the application has requested to run self-tests to get to get back into operational state after an error.
18	Init to Error is triggered by errors in the initialization code.
19	Init to Fatal-Error is triggered by non-recoverable errors in the initialization code.
20	Error to Error is triggered by errors while already in the Error state.

Table B.2: FIPS mode state transitions

B.3 FIPS Miscellaneous Information

Libgcrypt does not do any key management on itself; the application needs to care about it. Keys which are passed to Libgcrypt should be allocated in secure memory as available with the functions `gcry_malloc_secure` and `gcry_calloc_secure`. By calling `gcry_free` on this memory, the memory and thus the keys are overwritten with zero bytes before releasing the memory.

For use with the random number generator, Libgcrypt generates 3 internal keys which are stored in the encryption contexts used by the RNG. These keys are stored in secure memory for the lifetime of the process. Application are required to use `GCRYCTL_TERM_SECMEM` before process termination. This will zero out the entire secure memory and thus also the encryption contexts with these keys.

GNU Lesser General Public License

Version 2.1, February 1999

[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 contributions 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 distribution 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 software 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 APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT 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 MERCHANTABILITY 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 2, June 1991

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) 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 this service 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 make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. 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.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

1. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

   Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.
2. You may copy and distribute 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 and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

   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.
3. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, 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. You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.
   b. You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.
   c. If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

   These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, 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 Program, 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 Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

4. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

   a. 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; or,

   b. Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

   c. Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

   The source code for a work means the preferred form of the work for making modifications to it. For an executable work, 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 executable. However, as a special exception, the source code 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.

   If distribution of executable or 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 counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

5. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program 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.

6. 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 Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

7. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program 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 to this License.
8. 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 Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program 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 Program.

   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 contributions 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.
9. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program 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.
10. The Free Software Foundation may publish revised and/or new versions of the 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 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 Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.
11. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, 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 software and of promoting the sharing and reuse of software generally.

NO WARRANTY

12. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, 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.
13. 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 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.

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 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 program's name and an idea of what it does.
Copyright (C) 19yy  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 2
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, 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.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

```
Gnomovision version 69, Copyright (C) 19yy name of author
Gnomovision 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, the commands you use may be called something other than '`show w`' and '`show c`'; they could even be mouse-clicks or menu items—whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

```
Yoyodyne, Inc., hereby disclaims all copyright
interest in the program 'Gnomovision'
(which makes passes at compilers) written
by James Hacker.

signature of Ty Coon, 1 April 1989
Ty Coon, President of Vice
```

This 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 Library General Public License instead of this License.

List of Figures and Tables

Concept Index

I

L

M

N

O

P

R

S

T

W

X

Function and Data Index

A

G