GnuTLS API Reference Manual

COLLABORATORS			
	TITLE : GnuTLS API Reference Manual		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		June 24, 2010	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

GnuTLS API Reference Manual

GnuTLS is a implementation of the TLS (Transport Layer Security) and SSL (Secure Sockets Layer) protocols for the GNU project.

More up to date information can be found at http://www.gnu.org/software/gnutls/.

1.1 gnutls

gnutls —

Synopsis

```
#define             HAVE_SSIZE_T
#define             GNUTLS_VERSION
#define             GNUTLS_VERSION_MAJOR
#define             GNUTLS_VERSION_MINOR
#define             GNUTLS_VERSION_PATCH
#define             GNUTLS_VERSION_NUMBER
#define             GNUTLS_CIPHER_RIJNDAEL_128_CBC
#define             GNUTLS_CIPHER_RIJNDAEL_256_CBC
#define             GNUTLS_CIPHER_RIJNDAEL_CBC
#define             GNUTLS_CIPHER_ARCFOUR
enum                gnutls_cipher_algorithm_t;
enum                gnutls_kx_algorithm_t;
enum                gnutls_params_type_t;
enum                gnutls_credentials_type_t;
#define             GNUTLS_MAC_SHA
#define             GNUTLS_DIG_SHA
enum                gnutls_mac_algorithm_t;
enum                gnutls_digest_algorithm_t;
#define             GNUTLS_MAX_ALGORITHM_NUM
enum                gnutls_compression_method_t;
enum                gnutls_connection_end_t;
enum                gnutls_alert_level_t;
enum                gnutls_alert_description_t;
enum                gnutls_handshake_description_t;
enum                gnutls_certificate_status_t;
enum                gnutls_certificate_request_t;
enum                gnutls_openpgp_crt_status_t;
```

```
enum                gnutls_close_request_t;
enum                gnutls_protocol_t;
enum                gnutls_certificate_type_t;
enum                gnutls_x509_crt_fmt_t;
enum                gnutls_certificate_print_formats_t;
enum                gnutls_pk_algorithm_t;
const char *        gnutls_pk_algorithm_get_name        (gnutls_pk_algorithm_t algorithm);
enum                gnutls_sign_algorithm_t;
const char *        gnutls_sign_algorithm_get_name      (gnutls_sign_algorithm_t sign);
typedef             gnutls_transport_ptr_t;
struct              gnutls_session_int;
typedef             gnutls_session_t;
struct              gnutls_dh_params_int;
typedef             gnutls_dh_params_t;
struct              gnutls_x509_privkey_int;
typedef             gnutls_rsa_params_t;
struct              gnutls_priority_st;
typedef             gnutls_priority_t;
int                 gnutls_init                         (gnutls_session_t *session,
                                                         gnutls_connection_end_t con_end);
void                gnutls_deinit                       (gnutls_session_t session);
int                 gnutls_bye                          (gnutls_session_t session,
                                                         gnutls_close_request_t how);
int                 gnutls_handshake                    (gnutls_session_t session);
int                 gnutls_rehandshake                  (gnutls_session_t session);
gnutls_alert_description_t  gnutls_alert_get            (gnutls_session_t session);
int                 gnutls_alert_send                   (gnutls_session_t session,
                                                         gnutls_alert_level_t level,
                                                         gnutls_alert_description_t desc);
int                 gnutls_alert_send_appropriate       (gnutls_session_t session,
                                                         int err);
const char *        gnutls_alert_get_name               (gnutls_alert_description_t alert)
gnutls_cipher_algorithm_t  gnutls_cipher_get            (gnutls_session_t session);
gnutls_kx_algorithm_t  gnutls_kx_get                    (gnutls_session_t session);
gnutls_mac_algorithm_t  gnutls_mac_get                  (gnutls_session_t session);
gnutls_compression_method_t  gnutls_compression_get     (gnutls_session_t session);
gnutls_certificate_type_t  gnutls_certificate_type_get  (gnutls_session_t session);
int                 gnutls_sign_algorithm_get_requested (gnutls_session_t session,
                                                         size_t indx,
                                                         gnutls_sign_algorithm_t *algo);
size_t              gnutls_cipher_get_key_size          (gnutls_cipher_algorithm_t algorit
size_t              gnutls_mac_get_key_size             (gnutls_mac_algorithm_t algorithm)
const char *        gnutls_cipher_get_name              (gnutls_cipher_algorithm_t algorit
const char *        gnutls_mac_get_name                 (gnutls_mac_algorithm_t algorithm)
const char *        gnutls_compression_get_name         (gnutls_compression_method_t algor
const char *        gnutls_kx_get_name                  (gnutls_kx_algorithm_t algorithm);
const char *        gnutls_certificate_type_get_name    (gnutls_certificate_type_t type);
const char *        gnutls_pk_get_name                  (gnutls_pk_algorithm_t algorithm);
const char *        gnutls_sign_get_name                (gnutls_sign_algorithm_t algorithm
gnutls_mac_algorithm_t  gnutls_mac_get_id               (const char *name);
gnutls_compression_method_t  gnutls_compression_get_id  (const char *name);
gnutls_cipher_algorithm_t  gnutls_cipher_get_id         (const char *name);
gnutls_kx_algorithm_t  gnutls_kx_get_id                 (const char *name);
gnutls_protocol_t   gnutls_protocol_get_id              (const char *name);
gnutls_certificate_type_t  gnutls_certificate_type_get_id
                                                        (const char *name);
gnutls_pk_algorithm_t  gnutls_pk_get_id                 (const char *name);
```

```
gnutls_sign_algorithm_t  gnutls_sign_get_id             (const char *name);
const gnutls_cipher_algorithm_t * gnutls_cipher_list    (void);
const gnutls_mac_algorithm_t * gnutls_mac_list          (void);
const gnutls_compression_method_t * gnutls_compression_list
                                                        (void);
const gnutls_protocol_t * gnutls_protocol_list          (void);
const gnutls_certificate_type_t * gnutls_certificate_type_list
                                                        (void);
const gnutls_kx_algorithm_t * gnutls_kx_list            (void);
const gnutls_pk_algorithm_t * gnutls_pk_list            (void);
const gnutls_sign_algorithm_t * gnutls_sign_list        (void);
const char *        gnutls_cipher_suite_info            (size_t idx,
                                                         char *cs_id,
                                                         gnutls_kx_algorithm_t *kx,
                                                         gnutls_cipher_algorithm_t *cipher
                                                         gnutls_mac_algorithm_t *mac,
                                                         gnutls_protocol_t *version);
int                 gnutls_error_is_fatal               (int error);
int                 gnutls_error_to_alert               (int err,
                                                         int *level);
void                gnutls_perror                       (int error);
const char *        gnutls_strerror                     (int error);
const char *        gnutls_strerror_name                (int error);
void                gnutls_handshake_set_private_extensions
                                                        (gnutls_session_t session,
                                                         int allow);
gnutls_handshake_description_t  gnutls_handshake_get_last_out
                                                        (gnutls_session_t session);
gnutls_handshake_description_t  gnutls_handshake_get_last_in
                                                        (gnutls_session_t session);
ssize_t             gnutls_record_send                  (gnutls_session_t session,
                                                         const void *data,
                                                         size_t sizeofdata);
ssize_t             gnutls_record_recv                  (gnutls_session_t session,
                                                         void *data,
                                                         size_t sizeofdata);
#define             gnutls_read
#define             gnutls_write
void                gnutls_session_enable_compatibility_mode
                                                        (gnutls_session_t session);
void                gnutls_record_disable_padding       (gnutls_session_t session);
int                 gnutls_record_get_direction         (gnutls_session_t session);
size_t              gnutls_record_get_max_size          (gnutls_session_t session);
ssize_t             gnutls_record_set_max_size          (gnutls_session_t session,
                                                         size_t size);
size_t              gnutls_record_check_pending         (gnutls_session_t session);
int                 gnutls_prf                          (gnutls_session_t session,
                                                         size_t label_size,
                                                         const char *label,
                                                         int server_random_first,
                                                         size_t extra_size,
                                                         const char *extra,
                                                         size_t outsize,
                                                         char *out);
int                 gnutls_prf_raw                      (gnutls_session_t session,
                                                         size_t label_size,
                                                         const char *label,
```

```
                                                size_t seed_size,
                                                const char *seed,
                                                size_t outsize,
                                                char *out);
int                 (*gnutls_ext_recv_func)     (gnutls_session_t session,
                                                unsigned char *data,
                                                size_t len);
int                 (*gnutls_ext_send_func)     (gnutls_session_t session,
                                                unsigned char *data,
                                                size_t len);
enum                gnutls_ext_parse_type_t;
int                 gnutls_ext_register         (int type,
                                                const char *name,
                                                gnutls_ext_parse_type_t parse_typ
                                                gnutls_ext_recv_func recv_func,
                                                gnutls_ext_send_func send_func);
enum                gnutls_server_name_type_t;
int                 gnutls_server_name_set      (gnutls_session_t session,
                                                gnutls_server_name_type_t type,
                                                const void *name,
                                                size_t name_length);
int                 gnutls_server_name_get      (gnutls_session_t session,
                                                void *data,
                                                size_t *data_length,
                                                unsigned int *type,
                                                unsigned int indx);
int                 gnutls_safe_renegotiation_status (gnutls_session_t session);
void                gnutls_oprfi_enable_client  (gnutls_session_t session,
                                                size_t len,
                                                unsigned char *data);
int                 (*gnutls_oprfi_callback_func) (gnutls_session_t session,
                                                void *userdata,
                                                size_t oprfi_len,
                                                unsigned char *in_oprfi,
                                                unsigned char *out_oprfi);
void                gnutls_oprfi_enable_server  (gnutls_session_t session,
                                                gnutls_oprfi_callback_func cb,
                                                void *userdata);
enum                gnutls_supplemental_data_format_type_t;
int                 gnutls_session_ticket_key_generate (gnutls_datum_t *key);
int                 gnutls_session_ticket_enable_client (gnutls_session_t session);
int                 gnutls_session_ticket_enable_server (gnutls_session_t session,
                                                const gnutls_datum_t *key);
int                 gnutls_cipher_set_priority  (gnutls_session_t session,
                                                const int *list);
int                 gnutls_mac_set_priority     (gnutls_session_t session,
                                                const int *list);
int                 gnutls_compression_set_priority (gnutls_session_t session,
                                                const int *list);
int                 gnutls_kx_set_priority      (gnutls_session_t session,
                                                const int *list);
int                 gnutls_protocol_set_priority (gnutls_session_t session,
                                                const int *list);
int                 gnutls_certificate_type_set_priority
                                                (gnutls_session_t session,
                                                const int *list);
int                 gnutls_priority_init        (gnutls_priority_t *priority_cache
```

```
                                                 const char *priorities,
                                                 const char **err_pos);
void                gnutls_priority_deinit      (gnutls_priority_t priority_cache)
int                 gnutls_priority_set         (gnutls_session_t session,
                                                 gnutls_priority_t priority);
int                 gnutls_priority_set_direct  (gnutls_session_t session,
                                                 const char *priorities,
                                                 const char **err_pos);
int                 gnutls_set_default_priority (gnutls_session_t session);
int                 gnutls_set_default_export_priority
                                                (gnutls_session_t session);
const char *        gnutls_cipher_suite_get_name
                                                (gnutls_kx_algorithm_t kx_algorith
                                                 gnutls_cipher_algorithm_t cipher_
                                                 gnutls_mac_algorithm_t mac_algori
gnutls_protocol_t   gnutls_protocol_get_version (gnutls_session_t session);
const char *        gnutls_protocol_get_name    (gnutls_protocol_t version);
int                 gnutls_session_set_data     (gnutls_session_t session,
                                                 const void *session_data,
                                                 size_t session_data_size);
int                 gnutls_session_get_data     (gnutls_session_t session,
                                                 void *session_data,
                                                 size_t *session_data_size);
int                 gnutls_session_get_data2    (gnutls_session_t session,
                                                 gnutls_datum_t *data);
#define             GNUTLS_MAX_SESSION_ID
int                 gnutls_session_get_id       (gnutls_session_t session,
                                                 void *session_id,
                                                 size_t *session_id_size);
#define             GNUTLS_MASTER_SIZE
#define             GNUTLS_RANDOM_SIZE
const void *        gnutls_session_get_server_random
                                                (gnutls_session_t session);
const void *        gnutls_session_get_client_random
                                                (gnutls_session_t session);
const void *        gnutls_session_get_master_secret
                                                (gnutls_session_t session);
void                (*gnutls_finished_callback_func)
                                                (gnutls_session_t session,
                                                 const void *finished,
                                                 size_t len);
void                gnutls_session_set_finished_function
                                                (gnutls_session_t session,
                                                 gnutls_finished_callback_func fun
int                 gnutls_session_is_resumed   (gnutls_session_t session);
int                 (*gnutls_db_store_func)     (void *Param1,
                                                 gnutls_datum_t key,
                                                 gnutls_datum_t data);
int                 (*gnutls_db_remove_func)    (void *Param1,
                                                 gnutls_datum_t key);
gnutls_datum_t      (*gnutls_db_retr_func)      (void *Param1,
                                                 gnutls_datum_t key);
void                gnutls_db_set_cache_expiration
                                                (gnutls_session_t session,
                                                 int seconds);
void                gnutls_db_remove_session    (gnutls_session_t session);
void                gnutls_db_set_retrieve_function
                                                (gnutls_session_t session,
                                                 gnutls_db_retr_func retr_func);
void                gnutls_db_set_remove_function
                                                (gnutls_session_t session,
                                                 gnutls_db_remove_func rem_func);
void                gnutls_db_set_store_function
                                                (gnutls_session_t session,
                                                 gnutls_db_store_func store_func);
void                gnutls_db_set_ptr           (gnutls_session_t session,
                                                 void *ptr);
```

```
void *              gnutls_db_get_ptr                   (gnutls_session_t session);
int                 gnutls_db_check_entry               (gnutls_session_t session,
                                                         gnutls_datum_t session_entry);
int                 (*gnutls_handshake_post_client_hello_func)
                                                        (gnutls_session_t Param1);
void                gnutls_handshake_set_post_client_hello_function
                                                        (gnutls_session_t session,
                                                         gnutls_handshake_post_client_hell
void                gnutls_handshake_set_max_packet_length
                                                        (gnutls_session_t session,
                                                         size_t max);
const char *        gnutls_check_version                (const char *req_version);
void                gnutls_credentials_clear            (gnutls_session_t session);
int                 gnutls_credentials_set              (gnutls_session_t session,
                                                         gnutls_credentials_type_t type,
                                                         void *cred);
#define             gnutls_cred_set
struct              gnutls_certificate_credentials_st;
void                gnutls_anon_free_server_credentials (gnutls_anon_server_credentials_t
int                 gnutls_anon_allocate_server_credentials
                                                        (gnutls_anon_server_credentials_t
void                gnutls_anon_set_server_dh_params    (gnutls_anon_server_credentials_t
                                                         gnutls_dh_params_t dh_params);
void                gnutls_anon_set_server_params_function
                                                        (gnutls_anon_server_credentials_t
                                                         gnutls_params_function *func);
void                gnutls_anon_free_client_credentials (gnutls_anon_client_credentials_t
int                 gnutls_anon_allocate_client_credentials
                                                        (gnutls_anon_client_credentials_t
void                gnutls_certificate_free_credentials (gnutls_certificate_credentials_t
int                 gnutls_certificate_allocate_credentials
                                                        (gnutls_certificate_credentials_t
void                gnutls_certificate_free_keys        (gnutls_certificate_credentials_t
void                gnutls_certificate_free_cas         (gnutls_certificate_credentials_t
void                gnutls_certificate_free_ca_names    (gnutls_certificate_credentials_t
void                gnutls_certificate_free_crls        (gnutls_certificate_credentials_t
void                gnutls_certificate_set_dh_params    (gnutls_certificate_credentials_t
                                                         gnutls_dh_params_t dh_params);
void                gnutls_certificate_set_rsa_export_params
                                                        (gnutls_certificate_credentials_t
                                                         gnutls_rsa_params_t rsa_params);
void                gnutls_certificate_set_verify_flags (gnutls_certificate_credentials_t
                                                         unsigned int flags);
void                gnutls_certificate_set_verify_limits
                                                        (gnutls_certificate_credentials_t
                                                         unsigned int max_bits,
                                                         unsigned int max_depth);
int                 gnutls_certificate_set_x509_trust_file
                                                        (gnutls_certificate_credentials_t
                                                         const char *cafile,
                                                         gnutls_x509_crt_fmt_t type);
int                 gnutls_certificate_set_x509_trust_mem
                                                        (gnutls_certificate_credentials_t
                                                         const gnutls_datum_t *ca,
                                                         gnutls_x509_crt_fmt_t type);
int                 gnutls_certificate_set_x509_crl_file
                                                        (gnutls_certificate_credentials_t
```

```
                                                    const char *crlfile,
                                                    gnutls_x509_crt_fmt_t type);
int                 gnutls_certificate_set_x509_crl_mem (gnutls_certificate_credentials_t
                                                    const gnutls_datum_t *CRL,
                                                    gnutls_x509_crt_fmt_t type);
int                 gnutls_certificate_set_x509_key_file
                                                   (gnutls_certificate_credentials_t
                                                    const char *certfile,
                                                    const char *keyfile,
                                                    gnutls_x509_crt_fmt_t type);
int                 gnutls_certificate_set_x509_key_mem (gnutls_certificate_credentials_t
                                                    const gnutls_datum_t *cert,
                                                    const gnutls_datum_t *key,
                                                    gnutls_x509_crt_fmt_t type);
void                gnutls_certificate_send_x509_rdn_sequence
                                                   (gnutls_session_t session,
                                                    int status);
int                 gnutls_certificate_set_x509_simple_pkcs12_file
                                                   (gnutls_certificate_credentials_t
                                                    const char *pkcs12file,
                                                    gnutls_x509_crt_fmt_t type,
                                                    const char *password);
int                 gnutls_certificate_set_x509_simple_pkcs12_mem
                                                   (gnutls_certificate_credentials_t
                                                    const gnutls_datum_t *p12blob,
                                                    gnutls_x509_crt_fmt_t type,
                                                    const char *password);
typedef             gnutls_x509_privkey_t;
struct              gnutls_x509_crl_int;
typedef             gnutls_x509_crl_t;
struct              gnutls_x509_crt_int;
typedef             gnutls_x509_crt_t;
struct              gnutls_openpgp_keyring_int;
typedef             gnutls_openpgp_keyring_t;
int                 gnutls_certificate_set_x509_key     (gnutls_certificate_credentials_t
                                                    gnutls_x509_crt_t *cert_list,
                                                    int cert_list_size,
                                                    gnutls_x509_privkey_t key);
int                 gnutls_certificate_set_x509_trust   (gnutls_certificate_credentials_t
                                                    gnutls_x509_crt_t *ca_list,
                                                    int ca_list_size);
int                 gnutls_certificate_set_x509_crl     (gnutls_certificate_credentials_t
                                                    gnutls_x509_crl_t *crl_list,
                                                    int crl_list_size);
void                gnutls_certificate_get_x509_cas     (gnutls_certificate_credentials_t
                                                    gnutls_x509_crt_t **x509_ca_list,
                                                    unsigned int *ncas);
void                gnutls_certificate_get_x509_crls    (gnutls_certificate_credentials_t
                                                    gnutls_x509_crl_t **x509_crl_list
                                                    unsigned int *ncrls);
void                gnutls_certificate_get_openpgp_keyring
                                                   (gnutls_certificate_credentials_t
                                                    gnutls_openpgp_keyring_t *keyring
int                 gnutls_global_init                  (void);
void                gnutls_global_deinit                (void);
void *              (*gnutls_alloc_function)            (size_t Param1);
void *              (*gnutls_calloc_function)           (size_t Param1,
```

```
                                                 size_t Param2);
int                 (*gnutls_is_secure_function)        (const void *Param1);
void                (*gnutls_free_function)             (void *Param1);
void *              (*gnutls_realloc_function)          (void *Param1,
                                                         size_t Param2);
void                gnutls_global_set_mem_functions     (gnutls_alloc_function alloc_func,
                                                         gnutls_alloc_function secure_allo
                                                         gnutls_is_secure_function is_secu
                                                         gnutls_realloc_function realloc_f
                                                         gnutls_free_function free_func);
extern              gnutls_alloc_function gnutls_malloc;
extern              gnutls_alloc_function gnutls_secure_malloc;
extern              gnutls_realloc_function gnutls_realloc;
extern              gnutls_calloc_function gnutls_calloc;
extern              gnutls_free_function gnutls_free;
char *              (*gnutls_strdup)                    (const char *Param1);
void                (*gnutls_log_func)                  (...,
                                                         const char *Param2);
void                gnutls_global_set_log_function      (gnutls_log_func log_func);
void                gnutls_global_set_log_level         (int level);
int                 gnutls_dh_params_init               (gnutls_dh_params_t *dh_params);
void                gnutls_dh_params_deinit             (gnutls_dh_params_t dh_params);
int                 gnutls_dh_params_import_raw         (gnutls_dh_params_t dh_params,
                                                         const gnutls_datum_t *prime,
                                                         const gnutls_datum_t *generator);
int                 gnutls_dh_params_import_pkcs3       (gnutls_dh_params_t params,
                                                         const gnutls_datum_t *pkcs3_param
                                                         gnutls_x509_crt_fmt_t format);
int                 gnutls_dh_params_generate2          (gnutls_dh_params_t params,
                                                         unsigned int bits);
int                 gnutls_dh_params_export_pkcs3       (gnutls_dh_params_t params,
                                                         gnutls_x509_crt_fmt_t format,
                                                         unsigned char *params_data,
                                                         size_t *params_data_size);
int                 gnutls_dh_params_export_raw         (gnutls_dh_params_t params,
                                                         gnutls_datum_t *prime,
                                                         gnutls_datum_t *generator,
                                                         unsigned int *bits);
int                 gnutls_dh_params_cpy                (gnutls_dh_params_t dst,
                                                         gnutls_dh_params_t src);
int                 gnutls_rsa_params_init              (gnutls_rsa_params_t *rsa_params);
void                gnutls_rsa_params_deinit            (gnutls_rsa_params_t rsa_params);
int                 gnutls_rsa_params_cpy               (gnutls_rsa_params_t dst,
                                                         gnutls_rsa_params_t src);
int                 gnutls_rsa_params_import_raw        (gnutls_rsa_params_t rsa_params,
                                                         const gnutls_datum_t *m,
                                                         const gnutls_datum_t *e,
                                                         const gnutls_datum_t *d,
                                                         const gnutls_datum_t *p,
                                                         const gnutls_datum_t *q,
                                                         const gnutls_datum_t *u);
int                 gnutls_rsa_params_generate2         (gnutls_rsa_params_t params,
                                                         unsigned int bits);
int                 gnutls_rsa_params_export_raw        (gnutls_rsa_params_t params,
                                                         gnutls_datum_t *m,
                                                         gnutls_datum_t *e,
                                                         gnutls_datum_t *d,
```

```
                                                    gnutls_datum_t *p,
                                                    gnutls_datum_t *q,
                                                    gnutls_datum_t *u,
                                                    unsigned int *bits);
int                 gnutls_rsa_params_export_pkcs1  (gnutls_rsa_params_t params,
                                                    gnutls_x509_crt_fmt_t format,
                                                    unsigned char *params_data,
                                                    size_t *params_data_size);
int                 gnutls_rsa_params_import_pkcs1  (gnutls_rsa_params_t params,
                                                    const gnutls_datum_t *pkcs1_param
                                                    gnutls_x509_crt_fmt_t format);
ssize_t             (*gnutls_pull_func)             (gnutls_transport_ptr_t Param1,
                                                    void *Param2,
                                                    size_t Param3);
ssize_t             (*gnutls_push_func)             (gnutls_transport_ptr_t Param1,
                                                    const void *Param2,
                                                    size_t Param3);
void                gnutls_transport_set_ptr        (gnutls_session_t session,
                                                    gnutls_transport_ptr_t ptr);
void                gnutls_transport_set_ptr2       (gnutls_session_t session,
                                                    gnutls_transport_ptr_t recv_ptr,
                                                    gnutls_transport_ptr_t send_ptr);
gnutls_transport_ptr_t  gnutls_transport_get_ptr    (gnutls_session_t session);
void                gnutls_transport_get_ptr2       (gnutls_session_t session,
                                                    gnutls_transport_ptr_t *recv_ptr,
                                                    gnutls_transport_ptr_t *send_ptr)
void                gnutls_transport_set_lowat      (gnutls_session_t session,
                                                    int num);
void                gnutls_transport_set_push_function  (gnutls_session_t session,
                                                    gnutls_push_func push_func);
void                gnutls_transport_set_pull_function  (gnutls_session_t session,
                                                    gnutls_pull_func pull_func);
void                gnutls_transport_set_errno      (gnutls_session_t session,
                                                    int err);
void                gnutls_transport_set_global_errno   (int err);
void                gnutls_session_set_ptr          (gnutls_session_t session,
                                                    void *ptr);
void *              gnutls_session_get_ptr          (gnutls_session_t session);
void                gnutls_openpgp_send_cert        (gnutls_session_t session,
                                                    gnutls_openpgp_crt_status_t statu
int                 gnutls_fingerprint              (gnutls_digest_algorithm_t algo,
                                                    const gnutls_datum_t *data,
                                                    void *result,
                                                    size_t *result_size);
void                gnutls_srp_free_client_credentials  (gnutls_srp_client_credentials_t s
int                 gnutls_srp_allocate_client_credentials
                                                    (gnutls_srp_client_credentials_t *
int                 gnutls_srp_set_client_credentials   (gnutls_srp_client_credentials_t r
                                                    const char *username,
                                                    const char *password);
void                gnutls_srp_free_server_credentials  (gnutls_srp_server_credentials_t s
int                 gnutls_srp_allocate_server_credentials
                                                    (gnutls_srp_server_credentials_t *
int                 gnutls_srp_set_server_credentials_file
                                                    (gnutls_srp_server_credentials_t r
                                                    const char *password_file,
                                                    const char *password_conf_file);
```

```
const char *        gnutls_srp_server_get_username      (gnutls_session_t session);
void                gnutls_srp_set_prime_bits           (gnutls_session_t session,
                                                         unsigned int bits);
int                 gnutls_srp_verifier                 (const char *username,
                                                         const char *password,
                                                         const gnutls_datum_t *salt,
                                                         const gnutls_datum_t *generator,
                                                         const gnutls_datum_t *prime,
                                                         gnutls_datum_t *res);
extern              const gnutls_datum_t gnutls_srp_2048_group_prime;
extern              const gnutls_datum_t gnutls_srp_2048_group_generator;
extern              const gnutls_datum_t gnutls_srp_1536_group_prime;
extern              const gnutls_datum_t gnutls_srp_1536_group_generator;
extern              const gnutls_datum_t gnutls_srp_1024_group_prime;
extern              const gnutls_datum_t gnutls_srp_1024_group_generator;
void                gnutls_srp_set_server_credentials_function
                                                        (gnutls_srp_server_credentials_t c
                                                         gnutls_srp_server_credentials_fun
void                gnutls_srp_set_client_credentials_function
                                                        (gnutls_srp_client_credentials_t c
                                                         gnutls_srp_client_credentials_fun
int                 gnutls_srp_base64_encode            (const gnutls_datum_t *data,
                                                         char *result,
                                                         size_t *result_size);
int                 gnutls_srp_base64_encode_alloc      (const gnutls_datum_t *data,
                                                         gnutls_datum_t *result);
int                 gnutls_srp_base64_decode            (const gnutls_datum_t *b64_data,
                                                         char *result,
                                                         size_t *result_size);
int                 gnutls_srp_base64_decode_alloc      (const gnutls_datum_t *b64_data,
                                                         gnutls_datum_t *result);
enum                gnutls_psk_key_flags;
void                gnutls_psk_free_client_credentials  (gnutls_psk_client_credentials_t s
int                 gnutls_psk_allocate_client_credentials
                                                        (gnutls_psk_client_credentials_t *
int                 gnutls_psk_set_client_credentials   (gnutls_psk_client_credentials_t r
                                                         const char *username,
                                                         const gnutls_datum_t *key,
                                                         gnutls_psk_key_flags format);
void                gnutls_psk_free_server_credentials  (gnutls_psk_server_credentials_t s
int                 gnutls_psk_allocate_server_credentials
                                                        (gnutls_psk_server_credentials_t *
int                 gnutls_psk_set_server_credentials_file
                                                        (gnutls_psk_server_credentials_t r
                                                         const char *password_file);
int                 gnutls_psk_set_server_credentials_hint
                                                        (gnutls_psk_server_credentials_t r
                                                         const char *hint);
const char *        gnutls_psk_server_get_username      (gnutls_session_t session);
const char *        gnutls_psk_client_get_hint          (gnutls_session_t session);
void                gnutls_psk_set_server_credentials_function
                                                        (gnutls_psk_server_credentials_t c
                                                         gnutls_psk_server_credentials_fun
void                gnutls_psk_set_client_credentials_function
                                                        (gnutls_psk_client_credentials_t c
                                                         gnutls_psk_client_credentials_fun
int                 gnutls_hex_encode                   (const gnutls_datum_t *data,
```

```
                                                    char *result,
                                                    size_t *result_size);
int                 gnutls_hex_decode               (const gnutls_datum_t *hex_data,
                                                    char *result,
                                                    size_t *result_size);
void                gnutls_psk_set_server_dh_params (gnutls_psk_server_credentials_t r
                                                    gnutls_dh_params_t dh_params);
void                gnutls_psk_set_server_params_function
                                                    (gnutls_psk_server_credentials_t r
                                                    gnutls_params_function *func);
int                 gnutls_psk_netconf_derive_key   (const char *password,
                                                    const char *psk_identity,
                                                    const char *psk_identity_hint,
                                                    gnutls_datum_t *output_key);
enum                gnutls_x509_subject_alt_name_t;
struct              gnutls_openpgp_crt_int;
typedef             gnutls_openpgp_crt_t;
struct              gnutls_openpgp_privkey_int;
typedef             gnutls_openpgp_privkey_t;
gnutls_credentials_type_t  gnutls_auth_get_type     (gnutls_session_t session);
gnutls_credentials_type_t  gnutls_auth_server_get_type
                                                    (gnutls_session_t session);
gnutls_credentials_type_t  gnutls_auth_client_get_type
                                                    (gnutls_session_t session);
void                gnutls_dh_set_prime_bits        (gnutls_session_t session,
                                                    unsigned int bits);
int                 gnutls_dh_get_secret_bits       (gnutls_session_t session);
int                 gnutls_dh_get_peers_public_bits (gnutls_session_t session);
int                 gnutls_dh_get_prime_bits        (gnutls_session_t session);
int                 gnutls_dh_get_group             (gnutls_session_t session,
                                                    gnutls_datum_t *raw_gen,
                                                    gnutls_datum_t *raw_prime);
int                 gnutls_dh_get_pubkey            (gnutls_session_t session,
                                                    gnutls_datum_t *raw_key);
int                 gnutls_rsa_export_get_pubkey    (gnutls_session_t session,
                                                    gnutls_datum_t *exponent,
                                                    gnutls_datum_t *modulus);
int                 gnutls_rsa_export_get_modulus_bits
                                                    (gnutls_session_t session);
int                 (*gnutls_sign_func)             (gnutls_session_t session,
                                                    void *userdata,
                                                    gnutls_certificate_type_t cert_ty
                                                    const gnutls_datum_t *cert,
                                                    const gnutls_datum_t *hash,
                                                    gnutls_datum_t *signature);
void                gnutls_sign_callback_set        (gnutls_session_t session,
                                                    gnutls_sign_func sign_func,
                                                    void *userdata);
gnutls_sign_func    gnutls_sign_callback_get        (gnutls_session_t session,
                                                    void **userdata);
void                gnutls_certificate_client_set_retrieve_function
                                                    (gnutls_certificate_credentials_t
                                                    gnutls_certificate_client_retriev
void                gnutls_certificate_server_set_retrieve_function
                                                    (gnutls_certificate_credentials_t
                                                    gnutls_certificate_server_retriev
void                gnutls_certificate_set_verify_function
                                                    (gnutls_certificate_credentials_t
                                                    gnutls_certificate_verify_functio
void                gnutls_certificate_server_set_request
```

```
                                                    (gnutls_session_t session,
                                                     gnutls_certificate_request_t req)
const gnutls_datum_t * gnutls_certificate_get_peers     (gnutls_session_t session,
                                                     unsigned int *list_size);
const gnutls_datum_t * gnutls_certificate_get_ours      (gnutls_session_t session);
time_t              gnutls_certificate_activation_time_peers
                                                    (gnutls_session_t session);
time_t              gnutls_certificate_expiration_time_peers
                                                    (gnutls_session_t session);
int                 gnutls_certificate_client_get_request_status
                                                    (gnutls_session_t session);
int                 gnutls_certificate_verify_peers2    (gnutls_session_t session,
                                                     unsigned int *status);
int                 gnutls_certificate_verify_peers     (gnutls_session_t session);
int                 gnutls_pem_base64_encode            (const char *msg,
                                                     const gnutls_datum_t *data,
                                                     char *result,
                                                     size_t *result_size);
int                 gnutls_pem_base64_decode            (const char *header,
                                                     const gnutls_datum_t *b64_data,
                                                     unsigned char *result,
                                                     size_t *result_size);
int                 gnutls_pem_base64_encode_alloc      (const char *msg,
                                                     const gnutls_datum_t *data,
                                                     gnutls_datum_t *result);
int                 gnutls_pem_base64_decode_alloc      (const char *header,
                                                     const gnutls_datum_t *b64_data,
                                                     gnutls_datum_t *result);
#define             GNUTLS_KEY_DIGITAL_SIGNATURE
#define             GNUTLS_KEY_NON_REPUDIATION
#define             GNUTLS_KEY_KEY_ENCIPHERMENT
#define             GNUTLS_KEY_DATA_ENCIPHERMENT
#define             GNUTLS_KEY_KEY_AGREEMENT
#define             GNUTLS_KEY_KEY_CERT_SIGN
#define             GNUTLS_KEY_CRL_SIGN
#define             GNUTLS_KEY_ENCIPHER_ONLY
#define             GNUTLS_KEY_DECIPHER_ONLY
void                gnutls_certificate_set_params_function
                                                    (gnutls_certificate_credentials_t
                                                     gnutls_params_function *func);
void                gnutls_anon_set_params_function     (gnutls_anon_server_credentials_t
                                                     gnutls_params_function *func);
void                gnutls_psk_set_params_function      (gnutls_psk_server_credentials_t r
                                                     gnutls_params_function *func);
int                 gnutls_hex2bin                      (const char *hex_data,
                                                     size_t hex_size,
                                                     char *bin_data,
                                                     size_t *bin_size);
#define             GNUTLS_E_SUCCESS
#define             GNUTLS_E_UNKNOWN_COMPRESSION_ALGORITHM
#define             GNUTLS_E_UNKNOWN_CIPHER_TYPE
#define             GNUTLS_E_LARGE_PACKET
#define             GNUTLS_E_UNSUPPORTED_VERSION_PACKET
#define             GNUTLS_E_UNEXPECTED_PACKET_LENGTH
#define             GNUTLS_E_INVALID_SESSION
#define             GNUTLS_E_FATAL_ALERT_RECEIVED
#define             GNUTLS_E_UNEXPECTED_PACKET
```

```
#define             GNUTLS_E_WARNING_ALERT_RECEIVED
#define             GNUTLS_E_ERROR_IN_FINISHED_PACKET
#define             GNUTLS_E_UNEXPECTED_HANDSHAKE_PACKET
#define             GNUTLS_E_UNKNOWN_CIPHER_SUITE
#define             GNUTLS_E_UNWANTED_ALGORITHM
#define             GNUTLS_E_MPI_SCAN_FAILED
#define             GNUTLS_E_DECRYPTION_FAILED
#define             GNUTLS_E_MEMORY_ERROR
#define             GNUTLS_E_DECOMPRESSION_FAILED
#define             GNUTLS_E_COMPRESSION_FAILED
#define             GNUTLS_E_AGAIN
#define             GNUTLS_E_EXPIRED
#define             GNUTLS_E_DB_ERROR
#define             GNUTLS_E_SRP_PWD_ERROR
#define             GNUTLS_E_INSUFFICIENT_CREDENTIALS
#define             GNUTLS_E_INSUFICIENT_CREDENTIALS
#define             GNUTLS_E_INSUFFICIENT_CRED
#define             GNUTLS_E_INSUFICIENT_CRED
#define             GNUTLS_E_HASH_FAILED
#define             GNUTLS_E_BASE64_DECODING_ERROR
#define             GNUTLS_E_MPI_PRINT_FAILED
#define             GNUTLS_E_REHANDSHAKE
#define             GNUTLS_E_GOT_APPLICATION_DATA
#define             GNUTLS_E_RECORD_LIMIT_REACHED
#define             GNUTLS_E_ENCRYPTION_FAILED
#define             GNUTLS_E_PK_ENCRYPTION_FAILED
#define             GNUTLS_E_PK_DECRYPTION_FAILED
#define             GNUTLS_E_PK_SIGN_FAILED
#define             GNUTLS_E_X509_UNSUPPORTED_CRITICAL_EXTENSION
#define             GNUTLS_E_KEY_USAGE_VIOLATION
#define             GNUTLS_E_NO_CERTIFICATE_FOUND
#define             GNUTLS_E_INVALID_REQUEST
#define             GNUTLS_E_SHORT_MEMORY_BUFFER
#define             GNUTLS_E_INTERRUPTED
#define             GNUTLS_E_PUSH_ERROR
#define             GNUTLS_E_PULL_ERROR
#define             GNUTLS_E_RECEIVED_ILLEGAL_PARAMETER
#define             GNUTLS_E_REQUESTED_DATA_NOT_AVAILABLE
#define             GNUTLS_E_PKCS1_WRONG_PAD
#define             GNUTLS_E_RECEIVED_ILLEGAL_EXTENSION
#define             GNUTLS_E_INTERNAL_ERROR
#define             GNUTLS_E_DH_PRIME_UNACCEPTABLE
#define             GNUTLS_E_FILE_ERROR
#define             GNUTLS_E_TOO_MANY_EMPTY_PACKETS
#define             GNUTLS_E_UNKNOWN_PK_ALGORITHM
#define             GNUTLS_E_INIT_LIBEXTRA
#define             GNUTLS_E_LIBRARY_VERSION_MISMATCH
#define             GNUTLS_E_NO_TEMPORARY_RSA_PARAMS
#define             GNUTLS_E_LZO_INIT_FAILED
#define             GNUTLS_E_NO_COMPRESSION_ALGORITHMS
#define             GNUTLS_E_NO_CIPHER_SUITES
#define             GNUTLS_E_OPENPGP_GETKEY_FAILED
#define             GNUTLS_E_PK_SIG_VERIFY_FAILED
#define             GNUTLS_E_ILLEGAL_SRP_USERNAME
#define             GNUTLS_E_SRP_PWD_PARSING_ERROR
#define             GNUTLS_E_NO_TEMPORARY_DH_PARAMS
#define             GNUTLS_E_ASN1_ELEMENT_NOT_FOUND
```

```
#define             GNUTLS_E_ASN1_IDENTIFIER_NOT_FOUND
#define             GNUTLS_E_ASN1_DER_ERROR
#define             GNUTLS_E_ASN1_VALUE_NOT_FOUND
#define             GNUTLS_E_ASN1_GENERIC_ERROR
#define             GNUTLS_E_ASN1_VALUE_NOT_VALID
#define             GNUTLS_E_ASN1_TAG_ERROR
#define             GNUTLS_E_ASN1_TAG_IMPLICIT
#define             GNUTLS_E_ASN1_TYPE_ANY_ERROR
#define             GNUTLS_E_ASN1_SYNTAX_ERROR
#define             GNUTLS_E_ASN1_DER_OVERFLOW
#define             GNUTLS_E_OPENPGP_UID_REVOKED
#define             GNUTLS_E_CERTIFICATE_ERROR
#define             GNUTLS_E_X509_CERTIFICATE_ERROR
#define             GNUTLS_E_CERTIFICATE_KEY_MISMATCH
#define             GNUTLS_E_UNSUPPORTED_CERTIFICATE_TYPE
#define             GNUTLS_E_X509_UNKNOWN_SAN
#define             GNUTLS_E_OPENPGP_FINGERPRINT_UNSUPPORTED
#define             GNUTLS_E_X509_UNSUPPORTED_ATTRIBUTE
#define             GNUTLS_E_UNKNOWN_HASH_ALGORITHM
#define             GNUTLS_E_UNKNOWN_PKCS_CONTENT_TYPE
#define             GNUTLS_E_UNKNOWN_PKCS_BAG_TYPE
#define             GNUTLS_E_INVALID_PASSWORD
#define             GNUTLS_E_MAC_VERIFY_FAILED
#define             GNUTLS_E_CONSTRAINT_ERROR
#define             GNUTLS_E_WARNING_IA_IPHF_RECEIVED
#define             GNUTLS_E_WARNING_IA_FPHF_RECEIVED
#define             GNUTLS_E_IA_VERIFY_FAILED
#define             GNUTLS_E_UNKNOWN_ALGORITHM
#define             GNUTLS_E_UNSUPPORTED_SIGNATURE_ALGORITHM
#define             GNUTLS_E_SAFE_RENEGOTIATION_FAILED
#define             GNUTLS_E_UNSAFE_RENEGOTIATION_DENIED
#define             GNUTLS_E_UNKNOWN_SRP_USERNAME
#define             GNUTLS_E_BASE64_ENCODING_ERROR
#define             GNUTLS_E_INCOMPATIBLE_GCRYPT_LIBRARY
#define             GNUTLS_E_INCOMPATIBLE_CRYPTO_LIBRARY
#define             GNUTLS_E_INCOMPATIBLE_LIBTASN1_LIBRARY
#define             GNUTLS_E_OPENPGP_KEYRING_ERROR
#define             GNUTLS_E_X509_UNSUPPORTED_OID
#define             GNUTLS_E_RANDOM_FAILED
#define             GNUTLS_E_BASE64_UNEXPECTED_HEADER_ERROR
#define             GNUTLS_E_OPENPGP_SUBKEY_ERROR
#define             GNUTLS_E_CRYPTO_ALREADY_REGISTERED
#define             GNUTLS_E_HANDSHAKE_TOO_LARGE
#define             GNUTLS_E_CRYPTODEV_IOCTL_ERROR
#define             GNUTLS_E_CRYPTODEV_DEVICE_ERROR
#define             GNUTLS_E_UNIMPLEMENTED_FEATURE
#define             GNUTLS_E_APPLICATION_ERROR_MAX
#define             GNUTLS_E_APPLICATION_ERROR_MIN
```

Description

Details

HAVE_SSIZE_T

```
# define HAVE_SSIZE_T
```

GNUTLS_VERSION

```
#define GNUTLS_VERSION "2.10.0"
```

GNUTLS_VERSION_MAJOR

```
#define GNUTLS_VERSION_MAJOR 2
```

GNUTLS_VERSION_MINOR

```
#define GNUTLS_VERSION_MINOR 10
```

GNUTLS_VERSION_PATCH

```
#define GNUTLS_VERSION_PATCH 0
```

GNUTLS_VERSION_NUMBER

```
#define GNUTLS_VERSION_NUMBER 0x020a00
```

GNUTLS_CIPHER_RIJNDAEL_128_CBC

```
#define GNUTLS_CIPHER_RIJNDAEL_128_CBC GNUTLS_CIPHER_AES_128_CBC
```

GNUTLS_CIPHER_RIJNDAEL_256_CBC

```
#define GNUTLS_CIPHER_RIJNDAEL_256_CBC GNUTLS_CIPHER_AES_256_CBC
```

GNUTLS_CIPHER_RIJNDAEL_CBC

```
#define GNUTLS_CIPHER_RIJNDAEL_CBC GNUTLS_CIPHER_AES_128_CBC
```

GNUTLS_CIPHER_ARCFOUR

```
#define GNUTLS_CIPHER_ARCFOUR GNUTLS_CIPHER_ARCFOUR_128
```

enum gnutls_cipher_algorithm_t

```
typedef enum gnutls_cipher_algorithm
{
  GNUTLS_CIPHER_UNKNOWN = 0,
  GNUTLS_CIPHER_NULL = 1,
  GNUTLS_CIPHER_ARCFOUR_128 = 2,
  GNUTLS_CIPHER_3DES_CBC = 3,
  GNUTLS_CIPHER_AES_128_CBC = 4,
  GNUTLS_CIPHER_AES_256_CBC = 5,
  GNUTLS_CIPHER_ARCFOUR_40 = 6,
  GNUTLS_CIPHER_CAMELLIA_128_CBC = 7,
  GNUTLS_CIPHER_CAMELLIA_256_CBC = 8,
  GNUTLS_CIPHER_RC2_40_CBC = 90,
  GNUTLS_CIPHER_DES_CBC = 91,
  GNUTLS_CIPHER_AES_192_CBC = 92,

  /* used only for PGP internals. Ignored in TLS/SSL
   */
  GNUTLS_CIPHER_IDEA_PGP_CFB = 200,
  GNUTLS_CIPHER_3DES_PGP_CFB = 201,
  GNUTLS_CIPHER_CAST5_PGP_CFB = 202,
  GNUTLS_CIPHER_BLOWFISH_PGP_CFB = 203,
  GNUTLS_CIPHER_SAFER_SK128_PGP_CFB = 204,
  GNUTLS_CIPHER_AES128_PGP_CFB = 205,
  GNUTLS_CIPHER_AES192_PGP_CFB = 206,
  GNUTLS_CIPHER_AES256_PGP_CFB = 207,
  GNUTLS_CIPHER_TWOFISH_PGP_CFB = 208
} gnutls_cipher_algorithm_t;
```

Enumeration of different symmetric encryption algorithms.

GNUTLS_CIPHER_UNKNOWN Unknown algorithm.

GNUTLS_CIPHER_NULL NULL algorithm.

GNUTLS_CIPHER_ARCFOUR_128 ARCFOUR stream cipher with 128-bit keys.

GNUTLS_CIPHER_3DES_CBC 3DES in CBC mode.

GNUTLS_CIPHER_AES_128_CBC AES in CBC mode with 128-bit keys.

GNUTLS_CIPHER_AES_256_CBC AES in CBC mode with 256-bit keys.

GNUTLS_CIPHER_ARCFOUR_40 ARCFOUR stream cipher with 40-bit keys.

GNUTLS_CIPHER_CAMELLIA_128_CBC Camellia in CBC mode with 128-bit keys.

GNUTLS_CIPHER_CAMELLIA_256_CBC Camellia in CBC mode with 256-bit keys.

GNUTLS_CIPHER_RC2_40_CBC RC2 in CBC mode with 40-bit keys.

GNUTLS_CIPHER_DES_CBC DES in CBC mode (56-bit keys).

GNUTLS_CIPHER_AES_192_CBC AES in CBC mode with 192-bit keys.

GNUTLS_CIPHER_IDEA_PGP_CFB IDEA in CFB mode.

GNUTLS_CIPHER_3DES_PGP_CFB 3DES in CFB mode.

GNUTLS_CIPHER_CAST5_PGP_CFB CAST5 in CFB mode.

GNUTLS_CIPHER_BLOWFISH_PGP_CFB Blowfish in CFB mode.

GNUTLS_CIPHER_SAFER_SK128_PGP_CFB Safer-SK in CFB mode with 128-bit keys.

GNUTLS_CIPHER_AES128_PGP_CFB AES in CFB mode with 128-bit keys.

GNUTLS_CIPHER_AES192_PGP_CFB AES in CFB mode with 192-bit keys.

GNUTLS_CIPHER_AES256_PGP_CFB AES in CFB mode with 256-bit keys.

GNUTLS_CIPHER_TWOFISH_PGP_CFB Twofish in CFB mode.

enum gnutls_kx_algorithm_t

```
typedef enum
{
  GNUTLS_KX_UNKNOWN = 0,
  GNUTLS_KX_RSA = 1,
  GNUTLS_KX_DHE_DSS = 2,
  GNUTLS_KX_DHE_RSA = 3,
  GNUTLS_KX_ANON_DH = 4,
  GNUTLS_KX_SRP = 5,
  GNUTLS_KX_RSA_EXPORT = 6,
  GNUTLS_KX_SRP_RSA = 7,
  GNUTLS_KX_SRP_DSS = 8,
  GNUTLS_KX_PSK = 9,
  GNUTLS_KX_DHE_PSK = 10
} gnutls_kx_algorithm_t;
```

Enumeration of different key exchange algorithms.

GNUTLS_KX_UNKNOWN Unknown key-exchange algorithm.

GNUTLS_KX_RSA RSA key-exchange algorithm.

GNUTLS_KX_DHE_DSS DHE-DSS key-exchange algorithm.

GNUTLS_KX_DHE_RSA DHE-RSA key-exchange algorithm.

GNUTLS_KX_ANON_DH Anon-DH key-exchange algorithm.

GNUTLS_KX_SRP SRP key-exchange algorithm.

GNUTLS_KX_RSA_EXPORT RSA-EXPORT key-exchange algorithm.

GNUTLS_KX_SRP_RSA SRP-RSA key-exchange algorithm.

GNUTLS_KX_SRP_DSS SRP-DSS key-exchange algorithm.

GNUTLS_KX_PSK PSK key-exchange algorithm.

GNUTLS_KX_DHE_PSK DHE-PSK key-exchange algorithm.

enum gnutls_params_type_t

```
typedef enum
{
  GNUTLS_PARAMS_RSA_EXPORT = 1,
  GNUTLS_PARAMS_DH = 2
} gnutls_params_type_t;
```

Enumeration of different TLS session parameter types.

GNUTLS_PARAMS_RSA_EXPORT Session RSA-EXPORT parameters.

GNUTLS_PARAMS_DH Session Diffie-Hellman parameters.

enum gnutls_credentials_type_t

```
typedef enum
{
  GNUTLS_CRD_CERTIFICATE = 1,
  GNUTLS_CRD_ANON,
  GNUTLS_CRD_SRP,
  GNUTLS_CRD_PSK,
  GNUTLS_CRD_IA
} gnutls_credentials_type_t;
```

Enumeration of different credential types.

GNUTLS_CRD_CERTIFICATE Certificate credential.

GNUTLS_CRD_ANON Anonymous credential.

GNUTLS_CRD_SRP SRP credential.

GNUTLS_CRD_PSK PSK credential.

GNUTLS_CRD_IA IA credential.

GNUTLS_MAC_SHA

```
#define GNUTLS_MAC_SHA GNUTLS_MAC_SHA1
```

GNUTLS_DIG_SHA

```
#define GNUTLS_DIG_SHA GNUTLS_DIG_SHA1
```

enum gnutls_mac_algorithm_t

```
typedef enum
{
  GNUTLS_MAC_UNKNOWN = 0,
  GNUTLS_MAC_NULL = 1,
  GNUTLS_MAC_MD5 = 2,
  GNUTLS_MAC_SHA1 = 3,
  GNUTLS_MAC_RMD160 = 4,
  GNUTLS_MAC_MD2 = 5,
  GNUTLS_MAC_SHA256 = 6,
  GNUTLS_MAC_SHA384 = 7,
  GNUTLS_MAC_SHA512 = 8,
  GNUTLS_MAC_SHA224 = 9
    /* If you add anything here, make sure you align with
       gnutls_digest_algorithm_t. */
} gnutls_mac_algorithm_t;
```

Enumeration of different Message Authentication Code (MAC) algorithms.

GNUTLS_MAC_UNKNOWN Unknown MAC algorithm.

GNUTLS_MAC_NULL NULL MAC algorithm (empty output).

GNUTLS_MAC_MD5 HMAC-MD5 algorithm.

GNUTLS_MAC_SHA1 HMAC-SHA-1 algorithm.

GNUTLS_MAC_RMD160 HMAC-RMD160 algorithm.

GNUTLS_MAC_MD2 HMAC-MD2 algorithm.

GNUTLS_MAC_SHA256 HMAC-SHA-256 algorithm.

GNUTLS_MAC_SHA384 HMAC-SHA-384 algorithm.

GNUTLS_MAC_SHA512 HMAC-SHA-512 algorithm.

GNUTLS_MAC_SHA224 HMAC-SHA-224 algorithm.

enum gnutls_digest_algorithm_t

```
typedef enum
{
  GNUTLS_DIG_UNKNOWN = GNUTLS_MAC_UNKNOWN,
  GNUTLS_DIG_NULL = GNUTLS_MAC_NULL,
  GNUTLS_DIG_MD5 = GNUTLS_MAC_MD5,
  GNUTLS_DIG_SHA1 = GNUTLS_MAC_SHA1,
  GNUTLS_DIG_RMD160 = GNUTLS_MAC_RMD160,
  GNUTLS_DIG_MD2 = GNUTLS_MAC_MD2,
  GNUTLS_DIG_SHA256 = GNUTLS_MAC_SHA256,
  GNUTLS_DIG_SHA384 = GNUTLS_MAC_SHA384,
  GNUTLS_DIG_SHA512 = GNUTLS_MAC_SHA512,
  GNUTLS_DIG_SHA224 = GNUTLS_MAC_SHA224,
  /* If you add anything here, make sure you align with
     gnutls_mac_algorithm_t. */
} gnutls_digest_algorithm_t;
```

Enumeration of different digest (hash) algorithms.

GNUTLS_DIG_UNKNOWN Unknown hash algorithm.

GNUTLS_DIG_NULL NULL hash algorithm (empty output).

GNUTLS_DIG_MD5 MD5 algorithm.

GNUTLS_DIG_SHA1 SHA-1 algorithm.

GNUTLS_DIG_RMD160 RMD160 algorithm.

GNUTLS_DIG_MD2 MD2 algorithm.

GNUTLS_DIG_SHA256 SHA-256 algorithm.

GNUTLS_DIG_SHA384 SHA-384 algorithm.

GNUTLS_DIG_SHA512 SHA-512 algorithm.

GNUTLS_DIG_SHA224 SHA-224 algorithm.

GNUTLS_MAX_ALGORITHM_NUM

```
#define GNUTLS_MAX_ALGORITHM_NUM 16
```

enum gnutls_compression_method_t

```
typedef enum
{
  GNUTLS_COMP_UNKNOWN = 0,
  GNUTLS_COMP_NULL = 1,
  GNUTLS_COMP_DEFLATE = 2,
  GNUTLS_COMP_ZLIB = GNUTLS_COMP_DEFLATE,
  GNUTLS_COMP_LZO = 3   /* only available if gnutls-extra has
         been initialized
       */
} gnutls_compression_method_t;
```

Enumeration of different TLS compression methods.

GNUTLS_COMP_UNKNOWN Unknown compression method.

GNUTLS_COMP_NULL The NULL compression method (uncompressed).

GNUTLS_COMP_DEFLATE The deflate/zlib compression method.

GNUTLS_COMP_ZLIB Same as GNUTLS_COMP_DEFLATE.

GNUTLS_COMP_LZO The non-standard LZO compression method.

enum gnutls_connection_end_t

```
typedef enum
{
  GNUTLS_SERVER = 1,
  GNUTLS_CLIENT
} gnutls_connection_end_t;
```

Enumeration of different TLS connection end types.

GNUTLS_SERVER Connection end is a server.

GNUTLS_CLIENT Connection end is a client.

enum gnutls_alert_level_t

```
typedef enum
{
  GNUTLS_AL_WARNING = 1,
  GNUTLS_AL_FATAL
} gnutls_alert_level_t;
```

Enumeration of different TLS alert severities.

GNUTLS_AL_WARNING Alert of warning severity.

GNUTLS_AL_FATAL Alert of fatal severity.

enum gnutls_alert_description_t

```
typedef enum
{
  GNUTLS_A_CLOSE_NOTIFY,
  GNUTLS_A_UNEXPECTED_MESSAGE = 10,
  GNUTLS_A_BAD_RECORD_MAC = 20,
  GNUTLS_A_DECRYPTION_FAILED,
  GNUTLS_A_RECORD_OVERFLOW,
  GNUTLS_A_DECOMPRESSION_FAILURE = 30,
  GNUTLS_A_HANDSHAKE_FAILURE = 40,
  GNUTLS_A_SSL3_NO_CERTIFICATE = 41,
  GNUTLS_A_BAD_CERTIFICATE = 42,
  GNUTLS_A_UNSUPPORTED_CERTIFICATE,
  GNUTLS_A_CERTIFICATE_REVOKED,
  GNUTLS_A_CERTIFICATE_EXPIRED,
  GNUTLS_A_CERTIFICATE_UNKNOWN,
  GNUTLS_A_ILLEGAL_PARAMETER,
  GNUTLS_A_UNKNOWN_CA,
  GNUTLS_A_ACCESS_DENIED,
  GNUTLS_A_DECODE_ERROR = 50,
  GNUTLS_A_DECRYPT_ERROR,
  GNUTLS_A_EXPORT_RESTRICTION = 60,
  GNUTLS_A_PROTOCOL_VERSION = 70,
  GNUTLS_A_INSUFFICIENT_SECURITY,
  GNUTLS_A_INTERNAL_ERROR = 80,
  GNUTLS_A_USER_CANCELED = 90,
  GNUTLS_A_NO_RENEGOTIATION = 100,
  GNUTLS_A_UNSUPPORTED_EXTENSION = 110,
  GNUTLS_A_CERTIFICATE_UNOBTAINABLE = 111,
  GNUTLS_A_UNRECOGNIZED_NAME = 112,
  GNUTLS_A_UNKNOWN_PSK_IDENTITY = 115,
  GNUTLS_A_INNER_APPLICATION_FAILURE = 208,
  GNUTLS_A_INNER_APPLICATION_VERIFICATION = 209
} gnutls_alert_description_t;
```

Enumeration of different TLS alerts.

GNUTLS_A_CLOSE_NOTIFY Close notify.

GNUTLS_A_UNEXPECTED_MESSAGE Unexpected message.

GNUTLS_A_BAD_RECORD_MAC Bad record MAC.

GNUTLS_A_DECRYPTION_FAILED Decryption failed.

GNUTLS_A_RECORD_OVERFLOW Record overflow.

GNUTLS_A_DECOMPRESSION_FAILURE Decompression failed.

GNUTLS_A_HANDSHAKE_FAILURE Handshake failed.

GNUTLS_A_SSL3_NO_CERTIFICATE No certificate.

GNUTLS_A_BAD_CERTIFICATE Certificate is bad.

GNUTLS_A_UNSUPPORTED_CERTIFICATE Certificate is not supported.

GNUTLS_A_CERTIFICATE_REVOKED Certificate was revoked.

GNUTLS_A_CERTIFICATE_EXPIRED Certificate is expired.

GNUTLS_A_CERTIFICATE_UNKNOWN Unknown certificate.

GNUTLS_A_ILLEGAL_PARAMETER Illegal parameter.

GNUTLS_A_UNKNOWN_CA CA is unknown.

GNUTLS_A_ACCESS_DENIED Access was denied.

GNUTLS_A_DECODE_ERROR Decode error.

GNUTLS_A_DECRYPT_ERROR Decrypt error.

GNUTLS_A_EXPORT_RESTRICTION Export restriction.

GNUTLS_A_PROTOCOL_VERSION Error in protocol version.

GNUTLS_A_INSUFFICIENT_SECURITY Insufficient security.

GNUTLS_A_INTERNAL_ERROR Internal error.

GNUTLS_A_USER_CANCELED User canceled.

GNUTLS_A_NO_RENEGOTIATION No renegotiation is allowed.

GNUTLS_A_UNSUPPORTED_EXTENSION An unsupported extension was sent.

GNUTLS_A_CERTIFICATE_UNOBTAINABLE Could not retrieve the specified certificate.

GNUTLS_A_UNRECOGNIZED_NAME The server name sent was not recognized.

GNUTLS_A_UNKNOWN_PSK_IDENTITY The SRP/PSK username is missing or not known.

GNUTLS_A_INNER_APPLICATION_FAILURE Inner application negotiation failed.

GNUTLS_A_INNER_APPLICATION_VERIFICATION Inner application verification failed.

enum gnutls_handshake_description_t

```
typedef enum
{
  GNUTLS_HANDSHAKE_HELLO_REQUEST = 0,
  GNUTLS_HANDSHAKE_CLIENT_HELLO = 1,
  GNUTLS_HANDSHAKE_SERVER_HELLO = 2,
  GNUTLS_HANDSHAKE_NEW_SESSION_TICKET = 4,
  GNUTLS_HANDSHAKE_CERTIFICATE_PKT = 11,
  GNUTLS_HANDSHAKE_SERVER_KEY_EXCHANGE = 12,
  GNUTLS_HANDSHAKE_CERTIFICATE_REQUEST = 13,
  GNUTLS_HANDSHAKE_SERVER_HELLO_DONE = 14,
  GNUTLS_HANDSHAKE_CERTIFICATE_VERIFY = 15,
  GNUTLS_HANDSHAKE_CLIENT_KEY_EXCHANGE = 16,
  GNUTLS_HANDSHAKE_FINISHED = 20,
  GNUTLS_HANDSHAKE_SUPPLEMENTAL = 23
} gnutls_handshake_description_t;
```

Enumeration of different TLS handshake packets.

GNUTLS_HANDSHAKE_HELLO_REQUEST Hello request.

GNUTLS_HANDSHAKE_CLIENT_HELLO Client hello.

GNUTLS_HANDSHAKE_SERVER_HELLO Server hello.

GNUTLS_HANDSHAKE_NEW_SESSION_TICKET New session ticket.

GNUTLS_HANDSHAKE_CERTIFICATE_PKT Certificate packet.

GNUTLS_HANDSHAKE_SERVER_KEY_EXCHANGE Server key exchange.

GNUTLS_HANDSHAKE_CERTIFICATE_REQUEST Certificate request.

GNUTLS_HANDSHAKE_SERVER_HELLO_DONE Server hello done.

GNUTLS_HANDSHAKE_CERTIFICATE_VERIFY Certificate verify.

GNUTLS_HANDSHAKE_CLIENT_KEY_EXCHANGE Client key exchange.

GNUTLS_HANDSHAKE_FINISHED Finished.

GNUTLS_HANDSHAKE_SUPPLEMENTAL Supplemental.

enum gnutls_certificate_status_t

```
typedef enum
{
  GNUTLS_CERT_INVALID = 2,
  GNUTLS_CERT_REVOKED = 32,
  GNUTLS_CERT_SIGNER_NOT_FOUND = 64,
  GNUTLS_CERT_SIGNER_NOT_CA = 128,
  GNUTLS_CERT_INSECURE_ALGORITHM = 256,
  GNUTLS_CERT_NOT_ACTIVATED = 512,
  GNUTLS_CERT_EXPIRED = 1024
} gnutls_certificate_status_t;
```

Enumeration of certificate status codes. Note that the status bits have different meanings in OpenPGP keys and X.509 certificate verification.

GNUTLS_CERT_INVALID Will be set if the certificate was not verified.

GNUTLS_CERT_REVOKED Certificate revoked. In X.509 this will be set only if CRLs are checked.

GNUTLS_CERT_SIGNER_NOT_FOUND Certificate not verified. Signer not found.

GNUTLS_CERT_SIGNER_NOT_CA Certificate not verified. Signer not a CA certificate.

GNUTLS_CERT_INSECURE_ALGORITHM Certificate not verified, insecure algorithm.

GNUTLS_CERT_NOT_ACTIVATED Certificate not yet activated.

GNUTLS_CERT_EXPIRED Certificate expired.

enum gnutls_certificate_request_t

```
typedef enum
{
  GNUTLS_CERT_IGNORE = 0,
  GNUTLS_CERT_REQUEST = 1,
  GNUTLS_CERT_REQUIRE = 2
} gnutls_certificate_request_t;
```

Enumeration of certificate request types.

GNUTLS_CERT_IGNORE Ignore certificate.

GNUTLS_CERT_REQUEST Request certificate.

GNUTLS_CERT_REQUIRE Require certificate.

enum gnutls_openpgp_crt_status_t

```
typedef enum
{
  GNUTLS_OPENPGP_CERT = 0,
  GNUTLS_OPENPGP_CERT_FINGERPRINT = 1
} gnutls_openpgp_crt_status_t;
```

Enumeration of ways to send OpenPGP certificate.

GNUTLS_OPENPGP_CERT Send entire certificate.

GNUTLS_OPENPGP_CERT_FINGERPRINT Send only certificate fingerprint.

enum gnutls_close_request_t

```
typedef enum
{
  GNUTLS_SHUT_RDWR = 0,
  GNUTLS_SHUT_WR = 1
} gnutls_close_request_t;
```

Enumeration of how TLS session should be terminated. See gnutls_bye().

GNUTLS_SHUT_RDWR Disallow further receives/sends.

GNUTLS_SHUT_WR Disallow further sends.

enum gnutls_protocol_t

```
typedef enum
{
  GNUTLS_SSL3 = 1,
  GNUTLS_TLS1_0 = 2,
  GNUTLS_TLS1 = GNUTLS_TLS1_0,
  GNUTLS_TLS1_1 = 3,
  GNUTLS_TLS1_2 = 4,
  GNUTLS_VERSION_MAX = GNUTLS_TLS1_2,
  GNUTLS_VERSION_UNKNOWN = 0xff
} gnutls_protocol_t;
```

Enumeration of different SSL/TLS protocol versions.

GNUTLS_SSL3 SSL version 3.0.

GNUTLS_TLS1_0 TLS version 1.0.

GNUTLS_TLS1 Same as GNUTLS_TLS1_0.

GNUTLS_TLS1_1 TLS version 1.1.

GNUTLS_TLS1_2 TLS version 1.2.

GNUTLS_VERSION_MAX Maps to the highest supported TLS version.

GNUTLS_VERSION_UNKNOWN Unknown SSL/TLS version.

enum gnutls_certificate_type_t

```
typedef enum
{
  GNUTLS_CRT_UNKNOWN = 0,
  GNUTLS_CRT_X509 = 1,
  GNUTLS_CRT_OPENPGP = 2
} gnutls_certificate_type_t;
```

Enumeration of different certificate types.

GNUTLS_CRT_UNKNOWN Unknown certificate type.

GNUTLS_CRT_X509 X.509 Certificate.

GNUTLS_CRT_OPENPGP OpenPGP certificate.

enum gnutls_x509_crt_fmt_t

```
typedef enum
{
  GNUTLS_X509_FMT_DER = 0,
  GNUTLS_X509_FMT_PEM = 1
} gnutls_x509_crt_fmt_t;
```

Enumeration of different certificate encoding formats.

GNUTLS_X509_FMT_DER X.509 certificate in DER format (binary).

GNUTLS_X509_FMT_PEM X.509 certificate in PEM format (text).

enum gnutls_certificate_print_formats_t

```
typedef enum gnutls_certificate_print_formats
{
  GNUTLS_CRT_PRINT_FULL = 0,
  GNUTLS_CRT_PRINT_ONELINE = 1,
  GNUTLS_CRT_PRINT_UNSIGNED_FULL = 2
} gnutls_certificate_print_formats_t;
```

Enumeration of different certificate printing variants.

GNUTLS_CRT_PRINT_FULL Full information about certificate.

GNUTLS_CRT_PRINT_ONELINE Information about certificate in one line.

GNUTLS_CRT_PRINT_UNSIGNED_FULL All info for an unsigned certificate.

enum gnutls_pk_algorithm_t

```
typedef enum
{
  GNUTLS_PK_UNKNOWN = 0,
  GNUTLS_PK_RSA = 1,
  GNUTLS_PK_DSA = 2
} gnutls_pk_algorithm_t;
```

Enumeration of different public-key algorithms.

GNUTLS_PK_UNKNOWN Unknown public-key algorithm.

GNUTLS_PK_RSA RSA public-key algorithm.

GNUTLS_PK_DSA DSA public-key algorithm.

gnutls_pk_algorithm_get_name ()

```
const char *        gnutls_pk_algorithm_get_name        (gnutls_pk_algorithm_t algorithm);
```

Convert a gnutls_pk_algorithm_t value to a string.

algorithm : is a pk algorithm

Returns : a string that contains the name of the specified public key algorithm, or NULL.

enum gnutls_sign_algorithm_t

```
typedef enum
{
  GNUTLS_SIGN_UNKNOWN = 0,
  GNUTLS_SIGN_RSA_SHA1 = 1,
  GNUTLS_SIGN_RSA_SHA = GNUTLS_SIGN_RSA_SHA1,
  GNUTLS_SIGN_DSA_SHA1 = 2,
  GNUTLS_SIGN_DSA_SHA = GNUTLS_SIGN_DSA_SHA1,
  GNUTLS_SIGN_RSA_MD5 = 3,
  GNUTLS_SIGN_RSA_MD2 = 4,
  GNUTLS_SIGN_RSA_RMD160 = 5,
  GNUTLS_SIGN_RSA_SHA256 = 6,
  GNUTLS_SIGN_RSA_SHA384 = 7,
  GNUTLS_SIGN_RSA_SHA512 = 8,
  GNUTLS_SIGN_RSA_SHA224 = 9
} gnutls_sign_algorithm_t;
```

Enumeration of different digital signature algorithms.

GNUTLS_SIGN_UNKNOWN Unknown signature algorithm.

GNUTLS_SIGN_RSA_SHA1 Digital signature algorithm RSA with SHA-1

GNUTLS_SIGN_RSA_SHA Same as GNUTLS_SIGN_RSA_SHA1.

GNUTLS_SIGN_DSA_SHA1 Digital signature algorithm DSA with SHA-1

GNUTLS_SIGN_DSA_SHA Same as GNUTLS_SIGN_DSA_SHA1.

GNUTLS_SIGN_RSA_MD5 Digital signature algorithm RSA with MD5.

GNUTLS_SIGN_RSA_MD2 Digital signature algorithm RSA with MD2.

GNUTLS_SIGN_RSA_RMD160 Digital signature algorithm RSA with RMD-160.

GNUTLS_SIGN_RSA_SHA256 Digital signature algorithm RSA with SHA-256.

GNUTLS_SIGN_RSA_SHA384 Digital signature algorithm RSA with SHA-384.

GNUTLS_SIGN_RSA_SHA512 Digital signature algorithm RSA with SHA-512.

GNUTLS_SIGN_RSA_SHA224 Digital signature algorithm RSA with SHA-224.

gnutls_sign_algorithm_get_name ()

```
const char *        gnutls_sign_algorithm_get_name      (gnutls_sign_algorithm_t sign);
```

Convert a gnutls_sign_algorithm_t value to a string.

sign : is a sign algorithm

Returns : a string that contains the name of the specified sign algorithm, or NULL.

gnutls_transport_ptr_t

```
  typedef void *gnutls_transport_ptr_t;
```

struct gnutls_session_int

```
struct gnutls_session_int;
```

gnutls_session_t

```
  typedef struct gnutls_session_int *gnutls_session_t;
```

struct gnutls_dh_params_int

```
struct gnutls_dh_params_int;
```

gnutls_dh_params_t

```
  typedef struct gnutls_dh_params_int *gnutls_dh_params_t;
```

struct gnutls_x509_privkey_int

```
struct gnutls_x509_privkey_int;
```

gnutls_rsa_params_t

```
  typedef struct gnutls_x509_privkey_int *gnutls_rsa_params_t;
```

struct gnutls_priority_st

```
struct gnutls_priority_st;
```

gnutls_priority_t

```
  typedef struct gnutls_priority_st *gnutls_priority_t;
```

gnutls_init ()

```
int                 gnutls_init                         (gnutls_session_t *session,
                                                         gnutls_connection_end_t con_end);
```

This function initializes the current session to null. Every session must be initialized before use, so internal structures can be allocated. This function allocates structures which can only be free'd by calling gnutls_deinit(). Returns zero on success.

con_end can be one of GNUTLS_CLIENT and GNUTLS_SERVER.

session **:** is a pointer to a gnutls_session_t structure.

con_end **:** indicate if this session is to be used for server or client.

Returns **:** GNUTLS_E_SUCCESS on success, or an error code.

gnutls_deinit ()

```
void                gnutls_deinit                       (gnutls_session_t session);
```

This function clears all buffers associated with the *session*. This function will also remove session data from the session database if the session was terminated abnormally.

session **:** is a gnutls_session_t structure.

gnutls_bye ()

```
int                 gnutls_bye                          (gnutls_session_t session,
                                                         gnutls_close_request_t how);
```

Terminates the current TLS/SSL connection. The connection should have been initiated using gnutls_handshake(). *how* should be one of GNUTLS_SHUT_RDWR, GNUTLS_SHUT_WR.

In case of GNUTLS_SHUT_RDWR then the TLS connection gets terminated and further receives and sends will be disallowed. If the return value is zero you may continue using the connection. GNUTLS_SHUT_RDWR actually sends an alert containing a close request and waits for the peer to reply with the same message.

In case of GNUTLS_SHUT_WR then the TLS connection gets terminated and further sends will be disallowed. In order to reuse the connection you should wait for an EOF from the peer. GNUTLS_SHUT_WR sends an alert containing a close request.

Note that not all implementations will properly terminate a TLS connection. Some of them, usually for performance reasons, will terminate only the underlying transport layer, thus causing a transmission error to the peer. This error cannot be distinguished from a malicious party prematurely terminating the session, thus this behavior is not recommended.

This function may also return GNUTLS_E_AGAIN or GNUTLS_E_INTERRUPTED; cf. gnutls_record_get_direction().

session **:** is a gnutls_session_t structure.

how **:** is an integer

Returns **:** GNUTLS_E_SUCCESS on success, or an error code, see function documentation for entire semantics.

gnutls_handshake ()

```
int                 gnutls_handshake                    (gnutls_session_t session);
```

This function does the handshake of the TLS/SSL protocol, and initializes the TLS connection.

This function will fail if any problem is encountered, and will return a negative error code. In case of a client, if the client has asked to resume a session, but the server couldn't, then a full handshake will be performed.

The non-fatal errors such as GNUTLS_E_AGAIN and GNUTLS_E_INTERRUPTED interrupt the handshake procedure, which should be later be resumed. Call this function again, until it returns 0; cf. gnutls_record_get_direction() and gnutls_error_is_fatal().

If this function is called by a server after a rehandshake request then GNUTLS_E_GOT_APPLICATION_DATA or GNUTLS_E_WARN may be returned. Note that these are non fatal errors, only in the specific case of a rehandshake. Their meaning is that the client rejected the rehandshake request or in the case of GNUTLS_E_GOT_APPLICATION_DATA it might also mean that some data were pending.

session : is a gnutls_session_t structure.

Returns : GNUTLS_E_SUCCESS on success, otherwise an error.

gnutls_rehandshake ()

```
int                 gnutls_rehandshake                  (gnutls_session_t session);
```

This function will renegotiate security parameters with the client. This should only be called in case of a server.

This message informs the peer that we want to renegotiate parameters (perform a handshake).

If this function succeeds (returns 0), you must call the gnutls_handshake() function in order to negotiate the new parameters.

Since TLS is full duplex some application data might have been sent during peer's processing of this message. In that case one should call gnutls_record_recv() until GNUTLS_E_REHANDSHAKE is returned to clear any pending data. Care must be taken if rehandshake is mandatory to terminate if it does not start after some threshold.

If the client does not wish to renegotiate parameters he will should with an alert message, thus the return code will be GNUTLS_E_WARN and the alert will be GNUTLS_A_NO_RENEGOTIATION. A client may also choose to ignore this message.

session : is a gnutls_session_t structure.

Returns : GNUTLS_E_SUCCESS on success, otherwise an error.

gnutls_alert_get ()

```
gnutls_alert_description_t  gnutls_alert_get            (gnutls_session_t session);
```

This function will return the last alert number received. This function should be called if GNUTLS_E_WARNING_ALERT_RECEIVED or GNUTLS_E_FATAL_ALERT_RECEIVED has been returned by a gnutls function. The peer may send alerts if he thinks some things were not right. Check gnutls.h for the available alert descriptions.

If no alert has been received the returned value is undefined.

session : is a gnutls_session_t structure.

Returns : returns the last alert received, a gnutls_alert_description_t value.

gnutls_alert_send ()

```
int                 gnutls_alert_send                   (gnutls_session_t session,
                                                         gnutls_alert_level_t level,
                                                         gnutls_alert_description_t desc);
```

This function will send an alert to the peer in order to inform him of something important (eg. his Certificate could not be verified). If the alert level is Fatal then the peer is expected to close the connection, otherwise he may ignore the alert and continue.

The error code of the underlying record send function will be returned, so you may also receive GNUTLS_E_INTERRUPTED or GNUTLS_E_AGAIN as well.

session : is a gnutls_session_t structure.

level : is the level of the alert

desc : is the alert description

Returns : On success, GNUTLS_E_SUCCESS (0) is returned, otherwise an error code is returned.

gnutls_alert_send_appropriate ()

```
int                 gnutls_alert_send_appropriate       (gnutls_session_t session,
                                                         int err);
```

Sends an alert to the peer depending on the error code returned by a gnutls function. This function will call gnutls_error_to_alert() to determine the appropriate alert to send.

This function may also return GNUTLS_E_AGAIN, or GNUTLS_E_INTERRUPTED.

If the return value is GNUTLS_E_INVALID_REQUEST, then no alert has been sent to the peer.

session : is a gnutls_session_t structure.

err : is an integer

Returns : On success, GNUTLS_E_SUCCESS (0) is returned, otherwise an error code is returned.

gnutls_alert_get_name ()

```
const char *        gnutls_alert_get_name               (gnutls_alert_description_t alert);
```

This function will return a string that describes the given alert number, or NULL. See gnutls_alert_get().

alert : is an alert number gnutls_session_t structure.

Returns : string corresponding to gnutls_alert_description_t value.

gnutls_cipher_get ()

```
gnutls_cipher_algorithm_t  gnutls_cipher_get            (gnutls_session_t session);
```

Get currently used cipher.

session : is a gnutls_session_t structure.

Returns : the currently used cipher, a gnutls_cipher_algorithm_t type.

gnutls_kx_get ()

```
gnutls_kx_algorithm_t  gnutls_kx_get                    (gnutls_session_t session);
```

Get currently used key exchange algorithm.

session : is a gnutls_session_t structure.

Returns : the key exchange algorithm used in the last handshake, a gnutls_kx_algorithm_t value.

gnutls_mac_get ()

```
gnutls_mac_algorithm_t  gnutls_mac_get                  (gnutls_session_t session);
```

Get currently used MAC algorithm.

session : is a gnutls_session_t structure.

Returns : the currently used mac algorithm, a gnutls_mac_algorithm_t value.

gnutls_compression_get ()

```
gnutls_compression_method_t  gnutls_compression_get     (gnutls_session_t session);
```

Get currently used compression algorithm.

session : is a gnutls_session_t structure.

Returns : the currently used compression method, a gnutls_compression_method_t value.

gnutls_certificate_type_get ()

```
gnutls_certificate_type_t  gnutls_certificate_type_get  (gnutls_session_t session);
```

The certificate type is by default X.509, unless it is negotiated as a TLS extension.

session : is a gnutls_session_t structure.

Returns : the currently used gnutls_certificate_type_t certificate type.

gnutls_sign_algorithm_get_requested ()

```
int                 gnutls_sign_algorithm_get_requested (gnutls_session_t session,
                                                         size_t indx,
                                                         gnutls_sign_algorithm_t *algo);
```

Returns the signature algorithm specified by index that was requested by the peer. If the specified index has no data available this function returns GNUTLS_E_REQUESTED_DATA_NOT_AVAILABLE. If the negotiated TLS version does not support signature algorithms then GNUTLS_E_REQUESTED_DATA_NOT_AVAILABLE will be returned even for the first index. The first index is 0.

This function is useful in the certificate callback functions to assist in selecting the correct certificate.

session : is a gnutls_session_t structure.

indx : is an index of the signature algorithm to return

algo : the returned certificate type will be stored there

Returns : On success, GNUTLS_E_SUCCESS (0) is returned, otherwise an error code is returned.

Since 2.10.0

gnutls_cipher_get_key_size ()

```
size_t              gnutls_cipher_get_key_size          (gnutls_cipher_algorithm_t  ↩
    algorithm);
```

Get key size for cipher.

algorithm : is an encryption algorithm

Returns : length (in bytes) of the given cipher's key size, or 0 if the given cipher is invalid.

gnutls_mac_get_key_size ()

```
size_t              gnutls_mac_get_key_size             (gnutls_mac_algorithm_t algorithm);
```

Get size of MAC key.

algorithm : is an encryption algorithm

Returns : length (in bytes) of the given MAC key size, or 0 if the given MAC algorithm is invalid.

gnutls_cipher_get_name ()

```
const char *        gnutls_cipher_get_name              (gnutls_cipher_algorithm_t  ↩
    algorithm);
```

Convert a gnutls_cipher_algorithm_t type to a string.

algorithm : is an encryption algorithm

Returns : a pointer to a string that contains the name of the specified cipher, or NULL.

gnutls_mac_get_name ()

```
const char *        gnutls_mac_get_name                 (gnutls_mac_algorithm_t algorithm);
```

Convert a gnutls_mac_algorithm_t value to a string.

algorithm : is a MAC algorithm

Returns : a string that contains the name of the specified MAC algorithm, or NULL.

gnutls_compression_get_name ()

```
const char *        gnutls_compression_get_name         (gnutls_compression_method_t  ↩
    algorithm);
```

Convert a gnutls_compression_method_t value to a string.

algorithm : is a Compression algorithm

Returns : a pointer to a string that contains the name of the specified compression algorithm, or NULL.

gnutls_kx_get_name ()

```
const char *        gnutls_kx_get_name                  (gnutls_kx_algorithm_t algorithm);
```

Convert a gnutls_kx_algorithm_t value to a string.

algorithm : is a key exchange algorithm

Returns : a pointer to a string that contains the name of the specified key exchange algorithm, or NULL.

gnutls_certificate_type_get_name ()

```
const char *        gnutls_certificate_type_get_name    (gnutls_certificate_type_t type);
```

Convert a gnutls_certificate_type_t type to a string.

type : is a certificate type

Returns : a string that contains the name of the specified certificate type, or NULL in case of unknown types.

gnutls_pk_get_name ()

```
const char *        gnutls_pk_get_name                  (gnutls_pk_algorithm_t algorithm);
```

Convert a gnutls_pk_algorithm_t value to a string.

algorithm : is a public key algorithm

Returns : a pointer to a string that contains the name of the specified public key algorithm, or NULL.

Since 2.6.0

gnutls_sign_get_name ()

```
const char *        gnutls_sign_get_name                (gnutls_sign_algorithm_t algorithm) ↩
    ;
```

Convert a gnutls_sign_algorithm_t value to a string.

algorithm : is a public key signature algorithm

Returns : a pointer to a string that contains the name of the specified public key signature algorithm, or NULL.

Since 2.6.0

gnutls_mac_get_id ()

```
gnutls_mac_algorithm_t  gnutls_mac_get_id               (const char *name);
```

Convert a string to a gnutls_mac_algorithm_t value. The names are compared in a case insensitive way.

name : is a MAC algorithm name

Returns : a gnutls_mac_algorithm_t id of the specified MAC algorithm string, or GNUTLS_MAC_UNKNOWN on failures.

gnutls_compression_get_id ()

```
gnutls_compression_method_t  gnutls_compression_get_id  (const char *name);
```

The names are compared in a case insensitive way.

name : is a compression method name

Returns : an id of the specified in a string compression method, or GNUTLS_COMP_UNKNOWN on error.

gnutls_cipher_get_id ()

```
gnutls_cipher_algorithm_t  gnutls_cipher_get_id         (const char *name);
```

The names are compared in a case insensitive way.

name : is a MAC algorithm name

Returns : return a gnutls_cipher_algorithm_t value corresponding to the specified cipher, or GNUTLS_CIPHER_UNKNOWN on error.

gnutls_kx_get_id ()

```
gnutls_kx_algorithm_t  gnutls_kx_get_id                 (const char *name);
```

Convert a string to a gnutls_kx_algorithm_t value. The names are compared in a case insensitive way.

name : is a KX name

Returns : an id of the specified KX algorithm, or GNUTLS_KX_UNKNOWN on error.

gnutls_protocol_get_id ()

```
gnutls_protocol_t   gnutls_protocol_get_id              (const char *name);
```

The names are compared in a case insensitive way.

name : is a protocol name

Returns : an id of the specified protocol, or GNUTLS_VERSION_UNKNOWN on error.

gnutls_certificate_type_get_id ()

```
gnutls_certificate_type_t  gnutls_certificate_type_get_id
                                                        (const char *name);
```

The names are compared in a case insensitive way.

name : is a certificate type name

Returns : a gnutls_certificate_type_t for the specified in a string certificate type, or GNUTLS_CRT_UNKNOWN on error.

gnutls_pk_get_id ()

```
gnutls_pk_algorithm_t  gnutls_pk_get_id                (const char *name);
```

Convert a string to a gnutls_pk_algorithm_t value. The names are compared in a case insensitive way. For example, gnutls_pk_get_id("RS
will return GNUTLS_PK_RSA.

name : is a string containing a public key algorithm name.

Returns : a gnutls_pk_algorithm_t id of the specified public key algorithm string, or GNUTLS_PK_UNKNOWN on failures.

Since 2.6.0

gnutls_sign_get_id ()

```
gnutls_sign_algorithm_t  gnutls_sign_get_id            (const char *name);
```

The names are compared in a case insensitive way.

name : is a MAC algorithm name

Returns : return a gnutls_sign_algorithm_t value corresponding to the specified cipher, or GNUTLS_SIGN_UNKNOWN on error.

gnutls_cipher_list ()

```
const gnutls_cipher_algorithm_t * gnutls_cipher_list    (void);
```

Get a list of supported cipher algorithms. Note that not necessarily all ciphers are supported as TLS cipher suites. For example, DES is not supported as a cipher suite, but is supported for other purposes (e.g., PKCS#8 or similar).

Returns : a zero-terminated list of gnutls_cipher_algorithm_t integers indicating the available ciphers.

gnutls_mac_list ()

```
const gnutls_mac_algorithm_t * gnutls_mac_list          (void);
```

Get a list of hash algorithms for use as MACs. Note that not necessarily all MACs are supported in TLS cipher suites. For example, MD2 is not supported as a cipher suite, but is supported for other purposes (e.g., X.509 signature verification or similar).

Returns : Return a zero-terminated list of gnutls_mac_algorithm_t integers indicating the available MACs.

gnutls_compression_list ()

```
const gnutls_compression_method_t * gnutls_compression_list
                                                        (void);
```

Get a list of compression methods. Note that to be able to use LZO compression, you must link to libgnutls-extra and call gnutls_global_init_extra().

Returns : a zero-terminated list of gnutls_compression_method_t integers indicating the available compression methods.

gnutls_protocol_list ()

```
const gnutls_protocol_t * gnutls_protocol_list          (void);
```

Get a list of supported protocols, e.g. SSL 3.0, TLS 1.0 etc.

Returns **:** a zero-terminated list of gnutls_protocol_t integers indicating the available protocols.

gnutls_certificate_type_list ()

```
const gnutls_certificate_type_t * gnutls_certificate_type_list
                                                        (void);
```

Get a list of certificate types. Note that to be able to use OpenPGP certificates, you must link to libgnutls-extra and call gnutls_global_init_extra().

Returns **:** a zero-terminated list of gnutls_certificate_type_t integers indicating the available certificate types.

gnutls_kx_list ()

```
const gnutls_kx_algorithm_t * gnutls_kx_list            (void);
```

Get a list of supported key exchange algorithms.

Returns **:** a zero-terminated list of gnutls_kx_algorithm_t integers indicating the available key exchange algorithms.

gnutls_pk_list ()

```
const gnutls_pk_algorithm_t * gnutls_pk_list            (void);
```

Get a list of supported public key algorithms.

Returns **:** a zero-terminated list of gnutls_pk_algorithm_t integers indicating the available ciphers.

Since 2.6.0

gnutls_sign_list ()

```
const gnutls_sign_algorithm_t * gnutls_sign_list        (void);
```

Get a list of supported public key signature algorithms.

Returns **:** a zero-terminated list of gnutls_sign_algorithm_t integers indicating the available ciphers.

gnutls_cipher_suite_info ()

```
const char *        gnutls_cipher_suite_info            (size_t idx,
                                                         char *cs_id,
                                                         gnutls_kx_algorithm_t *kx,
                                                         gnutls_cipher_algorithm_t *cipher,
                                                         gnutls_mac_algorithm_t *mac,
                                                         gnutls_protocol_t *version);
```

Get information about supported cipher suites. Use the function iteratively to get information about all supported cipher suites. Call with idx=0 to get information about first cipher suite, then idx=1 and so on until the function returns NULL.

idx : index of cipher suite to get information about, starts on 0.

cs_id : output buffer with room for 2 bytes, indicating cipher suite value

kx : output variable indicating key exchange algorithm, or NULL.

cipher : output variable indicating cipher, or NULL.

mac : output variable indicating MAC algorithm, or NULL.

version : output variable indicating TLS protocol version, or NULL.

Returns : the name of *idx* cipher suite, and set the information about the cipher suite in the output variables. If *idx* is out of bounds, NULL is returned.

gnutls_error_is_fatal ()

```
int                 gnutls_error_is_fatal               (int error);
```

If a GnuTLS function returns a negative value you may feed that value to this function to see if the error condition is fatal.

Note that you may want to check the error code manually, since some non-fatal errors to the protocol may be fatal for you program.

This function is only useful if you are dealing with errors from the record layer or the handshake layer.

error : is a GnuTLS error code, a negative value

Returns : 1 if the error code is fatal, for positive *error* values, 0 is returned. For unknown *error* values, -1 is returned.

gnutls_error_to_alert ()

```
int                 gnutls_error_to_alert               (int err,
                                                         int *level);
```

Get an alert depending on the error code returned by a gnutls function. All alerts sent by this function should be considered fatal. The only exception is when *err* is GNUTLS_E_REHANDSHAKE, where a warning alert should be sent to the peer indicating that no renegotiation will be performed.

If there is no mapping to a valid alert the alert to indicate internal error is returned.

err : is a negative integer

level : the alert level will be stored there

Returns : the alert code to use for a particular error code.

gnutls_perror ()

```
void                gnutls_perror                       (int error);
```

This function is like perror(). The only difference is that it accepts an error number returned by a gnutls function.

error : is a GnuTLS error code, a negative value

gnutls_strerror ()

```
const char *        gnutls_strerror                     (int error);
```

This function is similar to strerror. The difference is that it accepts an error number returned by a gnutls function; In case of an unknown error a descriptive string is sent instead of NULL.

Error codes are always a negative value.

error : is a GnuTLS error code, a negative value

Returns : A string explaining the GnuTLS error message.

gnutls_strerror_name ()

```
const char *        gnutls_strerror_name                (int error);
```

Return the GnuTLS error code define as a string. For example, gnutls_strerror_name (GNUTLS_E_DH_PRIME_UNACCEPTABLE) will return the string "GNUTLS_E_DH_PRIME_UNACCEPTABLE".

error : is an error returned by a gnutls function.

Returns : A string corresponding to the symbol name of the error code.

Since 2.6.0

gnutls_handshake_set_private_extensions ()

```
void                gnutls_handshake_set_private_extensions
                                                        (gnutls_session_t session,
                                                         int allow);
```

This function will enable or disable the use of private cipher suites (the ones that start with 0xFF). By default or if *allow* is 0 then these cipher suites will not be advertized nor used.

Unless this function is called with the option to allow (1), then no compression algorithms, like LZO. That is because these algorithms are not yet defined in any RFC or even internet draft.

Enabling the private ciphersuites when talking to other than gnutls servers and clients may cause interoperability problems.

session : is a gnutls_session_t structure.

allow : is an integer (0 or 1)

gnutls_handshake_get_last_out ()

```
gnutls_handshake_description_t  gnutls_handshake_get_last_out
                                                    (gnutls_session_t session);
```

This function is only useful to check where the last performed handshake failed. If the previous handshake succeed or was not performed at all then no meaningful value will be returned.

Check gnutls_handshake_description_t in gnutls.h for the available handshake descriptions.

session : is a gnutls_session_t structure.

Returns : the last handshake message type sent, a gnutls_handshake_description_t.

gnutls_handshake_get_last_in ()

```
gnutls_handshake_description_t  gnutls_handshake_get_last_in
                                                    (gnutls_session_t session);
```

This function is only useful to check where the last performed handshake failed. If the previous handshake succeed or was not performed at all then no meaningful value will be returned.

Check gnutls_handshake_description_t in gnutls.h for the available handshake descriptions.

session : is a gnutls_session_t structure.

Returns : the last handshake message type received, a gnutls_handshake_description_t.

gnutls_record_send ()

```
ssize_t             gnutls_record_send                  (gnutls_session_t session,
                                                         const void *data,
                                                         size_t sizeofdata);
```

This function has the similar semantics with send(). The only difference is that it accepts a GnuTLS session, and uses different error codes.

Note that if the send buffer is full, send() will block this function. See the send() documentation for full information. You can replace the default push function by using gnutls_transport_set_ptr2() with a call to send() with a MSG_DONTWAIT flag if blocking is a problem.

If the EINTR is returned by the internal push function (the default is send()} then GNUTLS_E_INTERRUPTED will be returned. If GNUTLS_E_INTERRUPTED or GNUTLS_E_AGAIN is returned, you must call this function again, with the same parameters; alternatively you could provide a NULL pointer for data, and 0 for size. cf. gnutls_record_get_direction().

session : is a gnutls_session_t structure.

data : contains the data to send

sizeofdata : is the length of the data

Returns : the number of bytes sent, or a negative error code. The number of bytes sent might be less than `sizeofdata`. The maximum number of bytes this function can send in a single call depends on the negotiated maximum record size.

gnutls_record_recv ()

```
ssize_t             gnutls_record_recv                  (gnutls_session_t session,
                                                         void *data,
                                                         size_t sizeofdata);
```

This function has the similar semantics with recv(). The only difference is that it accepts a GnuTLS session, and uses different error codes.

In the special case that a server requests a renegotiation, the client may receive an error code of GNUTLS_E_REHANDSHAKE. This message may be simply ignored, replied with an alert GNUTLS_A_NO_RENEGOTIATION, or replied with a new handshake, depending on the client's will.

If EINTR is returned by the internal push function (the default is recv()) then GNUTLS_E_INTERRUPTED will be returned. If GNUTLS_E_INTERRUPTED or GNUTLS_E_AGAIN is returned, you must call this function again to get the data. See also gnutls_record_get_direction().

A server may also receive GNUTLS_E_REHANDSHAKE when a client has initiated a handshake. In that case the server can only initiate a handshake or terminate the connection.

session **:** is a gnutls_session_t structure.

data **:** the buffer that the data will be read into

sizeofdata **:** the number of requested bytes

Returns **:** the number of bytes received and zero on EOF. A negative error code is returned in case of an error. The number of bytes received might be less than *sizeofdata*.

gnutls_read

```
#define gnutls_read gnutls_record_recv
```

gnutls_write

```
#define gnutls_write gnutls_record_send
```

gnutls_session_enable_compatibility_mode ()

```
void                gnutls_session_enable_compatibility_mode
                                                        (gnutls_session_t session);
```

This function can be used to disable certain (security) features in TLS in order to maintain maximum compatibility with buggy clients. It is equivalent to calling: gnutls_record_disable_padding()

Normally only servers that require maximum compatibility with everything out there, need to call this function.

session **:** is a gnutls_session_t structure.

gnutls_record_disable_padding ()

```
void                gnutls_record_disable_padding       (gnutls_session_t session);
```

Used to disabled padding in TLS 1.0 and above. Normally you do not need to use this function, but there are buggy clients that complain if a server pads the encrypted data. This of course will disable protection against statistical attacks on the data.

Normally only servers that require maximum compatibility with everything out there, need to call this function.

session **:** is a gnutls_session_t structure.

gnutls_record_get_direction ()

```
int                 gnutls_record_get_direction         (gnutls_session_t session);
```

This function provides information about the internals of the record protocol and is only useful if a prior gnutls function call (e.g. gnutls_handshake()) was interrupted for some reason, that is, if a function returned GNUTLS_E_INTERRUPTED or GNUTLS_E_AGAIN. In such a case, you might want to call select() or poll() before calling the interrupted gnutls function again. To tell you whether a file descriptor should be selected for either reading or writing, gnutls_record_get_direction() returns 0 if the interrupted function was trying to read data, and 1 if it was trying to write data.

session : is a gnutls_session_t structure.

Returns : 0 if trying to read data, 1 if trying to write data.

gnutls_record_get_max_size ()

```
size_t              gnutls_record_get_max_size          (gnutls_session_t session);
```

Get the record size. The maximum record size is negotiated by the client after the first handshake message.

session : is a gnutls_session_t structure.

Returns : The maximum record packet size in this connection.

gnutls_record_set_max_size ()

```
ssize_t             gnutls_record_set_max_size          (gnutls_session_t session,
                                                         size_t size);
```

This function sets the maximum record packet size in this connection. This property can only be set to clients. The server may choose not to accept the requested size.

Acceptable values are 512(=2^9), 1024(=2^10), 2048(=2^11) and 4096(=2^12). The requested record size does get in effect immediately only while sending data. The receive part will take effect after a successful handshake.

This function uses a TLS extension called 'max record size'. Not all TLS implementations use or even understand this extension.

session : is a gnutls_session_t structure.

size : is the new size

Returns : On success, GNUTLS_E_SUCCESS (zero) is returned, otherwise an error code is returned.

gnutls_record_check_pending ()

```
size_t              gnutls_record_check_pending         (gnutls_session_t session);
```

This function checks if there are any data to receive in the gnutls buffers.

Notice that you may also use select() to check for data in a TCP connection, instead of this function. GnuTLS leaves some data in the tcp buffer in order for select to work.

session : is a gnutls_session_t structure.

Returns : the size of that data or 0.

gnutls_prf ()

```
int                 gnutls_prf                          (gnutls_session_t session,
                                                         size_t label_size,
                                                         const char *label,
                                                         int server_random_first,
                                                         size_t extra_size,
                                                         const char *extra,
                                                         size_t outsize,
                                                         char *out);
```

Apply the TLS Pseudo-Random-Function (PRF) using the master secret on some data, seeded with the client and server random fields.

The *label* variable usually contain a string denoting the purpose for the generated data. The *server_random_first* indicate whether the client random field or the server random field should be first in the seed. Non-0 indicate that the server random field is first, 0 that the client random field is first.

The *extra* variable can be used to add more data to the seed, after the random variables. It can be used to tie make sure the generated output is strongly connected to some additional data (e.g., a string used in user authentication).

The output is placed in **OUT*, which must be pre-allocated.

***session*:** is a gnutls_session_t structure.

***label_size*:** length of the *label* variable.

***label*:** label used in PRF computation, typically a short string.

***server_random_first*:** non-0 if server random field should be first in seed

***extra_size*:** length of the *extra* variable.

***extra*:** optional extra data to seed the PRF with.

***outsize*:** size of pre-allocated output buffer to hold the output.

***out*:** pre-allocate buffer to hold the generated data.

***Returns*:** GNUTLS_E_SUCCESS on success, or an error code.

gnutls_prf_raw ()

```
int                 gnutls_prf_raw                      (gnutls_session_t session,
                                                         size_t label_size,
                                                         const char *label,
                                                         size_t seed_size,
                                                         const char *seed,
                                                         size_t outsize,
                                                         char *out);
```

Apply the TLS Pseudo-Random-Function (PRF) using the master secret on some data.

The *label* variable usually contain a string denoting the purpose for the generated data. The *seed* usually contain data such as the client and server random, perhaps together with some additional data that is added to guarantee uniqueness of the output for a particular purpose.

Because the output is not guaranteed to be unique for a particular session unless *seed* include the client random and server random fields (the PRF would output the same data on another connection resumed from the first one), it is not recommended to use this function directly. The gnutls_prf() function seed the PRF with the client and server random fields directly, and is recommended if you want to generate pseudo random data unique for each session.

session : is a gnutls_session_t structure.

label_size : length of the *label* variable.

label : label used in PRF computation, typically a short string.

seed_size : length of the *seed* variable.

seed : optional extra data to seed the PRF with.

outsize : size of pre-allocated output buffer to hold the output.

out : pre-allocate buffer to hold the generated data.

Returns : GNUTLS_E_SUCCESS on success, or an error code.

gnutls_ext_recv_func ()

```
int                 (*gnutls_ext_recv_func)             (gnutls_session_t session,
                                                         unsigned char *data,
                                                         size_t len);
```

session :

data :

len :

Returns :

gnutls_ext_send_func ()

```
int                 (*gnutls_ext_send_func)             (gnutls_session_t session,
                                                         unsigned char *data,
                                                         size_t len);
```

session :

data :

len :

Returns :

enum gnutls_ext_parse_type_t

```
typedef enum
{
  GNUTLS_EXT_ANY = 0,
  GNUTLS_EXT_APPLICATION = 1,
  GNUTLS_EXT_TLS = 2,
  GNUTLS_EXT_MANDATORY = 3,
  GNUTLS_EXT_NONE = 4,
} gnutls_ext_parse_type_t;
```

Enumeration of different TLS extension types. This flag indicates for an extension whether it is useful to application level or TLS level only. This is (only) used to parse the application level extensions before the "client_hello" callback is called.

GNUTLS_EXT_ANY Any extension type.

GNUTLS_EXT_APPLICATION Application extension.

GNUTLS_EXT_TLS TLS-internal extension.

GNUTLS_EXT_MANDATORY Extension parsed even if resuming (or extensions are disabled).

GNUTLS_EXT_NONE Never parsed

gnutls_ext_register ()

```
int                 gnutls_ext_register               (int type,
                                                       const char *name,
                                                       gnutls_ext_parse_type_t parse_type ↩
                                                           ,
                                                       gnutls_ext_recv_func recv_func,
                                                       gnutls_ext_send_func send_func);
```

This function is used to register a new TLS extension handler.

type : the 16-bit integer referring to the extension type

name : human printable name of the extension used for debugging

parse_type : either GNUTLS_EXT_TLS or GNUTLS_EXT_APPLICATION.

recv_func : a function to receive extension data

send_func : a function to send extension data

Returns : GNUTLS_E_SUCCESS on success, or an error code.

Since 2.6.0

enum gnutls_server_name_type_t

```
typedef enum
{
  GNUTLS_NAME_DNS = 1
} gnutls_server_name_type_t;
```

Enumeration of different server name types.

GNUTLS_NAME_DNS Domain Name System name type.

gnutls_server_name_set ()

```
int                 gnutls_server_name_set            (gnutls_session_t session,
                                                       gnutls_server_name_type_t type,
                                                       const void *name,
                                                       size_t name_length);
```

This function is to be used by clients that want to inform (via a TLS extension mechanism) the server of the name they connected to. This should be used by clients that connect to servers that do virtual hosting.

The value of *name* depends on the *type* type. In case of GNUTLS_NAME_DNS, an ASCII zero-terminated domain name string, without the trailing dot, is expected. IPv4 or IPv6 addresses are not permitted.

session : is a gnutls_session_t structure.

type : specifies the indicator type

name : is a string that contains the server name.

name_length : holds the length of name

Returns : On success, GNUTLS_E_SUCCESS (zero) is returned, otherwise an error code is returned.

gnutls_server_name_get ()

```
int                 gnutls_server_name_get              (gnutls_session_t session,
                                                         void *data,
                                                         size_t *data_length,
                                                         unsigned int *type,
                                                         unsigned int indx);
```

This function will allow you to get the name indication (if any), a client has sent. The name indication may be any of the enumeration gnutls_server_name_type_t.

If *type* is GNUTLS_NAME_DNS, then this function is to be used by servers that support virtual hosting, and the data will be a null terminated UTF-8 string.

If *data* has not enough size to hold the server name GNUTLS_E_SHORT_MEMORY_BUFFER is returned, and *data_length* will hold the required size.

index is used to retrieve more than one server names (if sent by the client). The first server name has an index of 0, the second 1 and so on. If no name with the given index exists GNUTLS_E_REQUESTED_DATA_NOT_AVAILABLE is returned.

session : is a gnutls_session_t structure.

data : will hold the data

data_length : will hold the data length. Must hold the maximum size of data.

type : will hold the server name indicator type

indx : is the index of the server_name

Returns : On success, GNUTLS_E_SUCCESS (zero) is returned, otherwise an error code is returned.

gnutls_safe_renegotiation_status ()

```
int                 gnutls_safe_renegotiation_status    (gnutls_session_t session);
```

Can be used to check whether safe renegotiation is being used in the current session.

session : is a gnutls_session_t structure.

Returns : 0 when safe renegotiation is not used and non zero when safe renegotiation is used.

Since 2.10.0

gnutls_oprfi_enable_client ()

```
void                gnutls_oprfi_enable_client          (gnutls_session_t session,
                                                         size_t len,
                                                         unsigned char *data);
```

Request that the client should attempt to negotiate the Opaque PRF Input TLS extension, using the given data as the client's Opaque PRF input.

The data is copied into the session context after this call, so you may de-allocate it immediately after calling this function.

session : is a gnutls_session_t structure.

len : length of Opaque PRF data to use in client.

data : Opaque PRF data to use in client.

gnutls_oprfi_callback_func ()

```
int                 (*gnutls_oprfi_callback_func)       (gnutls_session_t session,
                                                         void *userdata,
                                                         size_t oprfi_len,
                                                         unsigned char *in_oprfi,
                                                         unsigned char *out_oprfi);
```

session :

userdata :

oprfi_len :

in_oprfi :

out_oprfi :

Returns :

gnutls_oprfi_enable_server ()

```
void                gnutls_oprfi_enable_server          (gnutls_session_t session,
                                                         gnutls_oprfi_callback_func cb,
                                                         void *userdata);
```

Request that the server should attempt to accept the Opaque PRF Input TLS extension. If the client requests the extension, the provided callback `cb` will be invoked. The callback must have the following prototype:

int callback (gnutls_session_t session, void *userdata, size_t oprfi_len, const unsigned char *in_oprfi, unsigned char *out_oprfi);

The callback can inspect the client-provided data in the input parameters, and specify its own opaque prf input data in the output variable. The function must return 0 on success, otherwise the handshake will be aborted.

session : is a gnutls_session_t structure.

cb : function pointer to Opaque PRF extension server callback.

userdata : hook passed to callback function for passing application state.

enum gnutls_supplemental_data_format_type_t

```
typedef enum
{
  GNUTLS_SUPPLEMENTAL_USER_MAPPING_DATA = 0
} gnutls_supplemental_data_format_type_t;
```

Enumeration of different supplemental data types (RFC 4680).

GNUTLS_SUPPLEMENTAL_USER_MAPPING_DATA Supplemental user mapping data.

gnutls_session_ticket_key_generate ()

```
int                 gnutls_session_ticket_key_generate  (gnutls_datum_t *key);
```

Generate a random key to encrypt security parameters within SessionTicket.

key **:** is a pointer to a gnutls_datum_t which will contain a newly created key.

Returns **:** On success, GNUTLS_E_SUCCESS (0) is returned, or an error code.

Since 2.10.0

gnutls_session_ticket_enable_client ()

```
int                 gnutls_session_ticket_enable_client (gnutls_session_t session);
```

Request that the client should attempt session resumption using SessionTicket.

session **:** is a gnutls_session_t structure.

Returns **:** On success, GNUTLS_E_SUCCESS (0) is returned, or an error code.

Since 2.10.0

gnutls_session_ticket_enable_server ()

```
int                 gnutls_session_ticket_enable_server (gnutls_session_t session,
                                                         const gnutls_datum_t *key);
```

Request that the server should attempt session resumption using SessionTicket. *key* must be initialized with gnutls_session_ticket_key_g

session **:** is a gnutls_session_t structure.

key **:** key to encrypt session parameters.

Returns **:** On success, GNUTLS_E_SUCCESS (0) is returned, or an error code.

Since 2.10.0

gnutls_cipher_set_priority ()

```
int                 gnutls_cipher_set_priority          (gnutls_session_t session,
                                                         const int *list);
```

Sets the priority on the ciphers supported by gnutls. Priority is higher for elements specified before others. After specifying the ciphers you want, you must append a 0. Note that the priority is set on the client. The server does not use the algorithm's priority except for disabling algorithms that were not specified.

session : is a gnutls_session_t structure.

list : is a 0 terminated list of gnutls_cipher_algorithm_t elements.

Returns : GNUTLS_E_SUCCESS on success, or an error code.

gnutls_mac_set_priority ()

```
int                 gnutls_mac_set_priority             (gnutls_session_t session,
                                                         const int *list);
```

Sets the priority on the mac algorithms supported by gnutls. Priority is higher for elements specified before others. After specifying the algorithms you want, you must append a 0. Note that the priority is set on the client. The server does not use the algorithm's priority except for disabling algorithms that were not specified.

session : is a gnutls_session_t structure.

list : is a 0 terminated list of gnutls_mac_algorithm_t elements.

Returns : GNUTLS_E_SUCCESS on success, or an error code.

gnutls_compression_set_priority ()

```
int                 gnutls_compression_set_priority     (gnutls_session_t session,
                                                         const int *list);
```

Sets the priority on the compression algorithms supported by gnutls. Priority is higher for elements specified before others. After specifying the algorithms you want, you must append a 0. Note that the priority is set on the client. The server does not use the algorithm's priority except for disabling algorithms that were not specified.

TLS 1.0 does not define any compression algorithms except NULL. Other compression algorithms are to be considered as gnutls extensions.

session : is a gnutls_session_t structure.

list : is a 0 terminated list of gnutls_compression_method_t elements.

Returns : GNUTLS_E_SUCCESS on success, or an error code.

gnutls_kx_set_priority ()

```
int                 gnutls_kx_set_priority              (gnutls_session_t session,
                                                         const int *list);
```

Sets the priority on the key exchange algorithms supported by gnutls. Priority is higher for elements specified before others. After specifying the algorithms you want, you must append a 0. Note that the priority is set on the client. The server does not use the algorithm's priority except for disabling algorithms that were not specified.

session : is a gnutls_session_t structure.

list : is a 0 terminated list of gnutls_kx_algorithm_t elements.

Returns : GNUTLS_E_SUCCESS on success, or an error code.

gnutls_protocol_set_priority ()

```
int                 gnutls_protocol_set_priority        (gnutls_session_t session,
                                                         const int *list);
```

Sets the priority on the protocol versions supported by gnutls. This function actually enables or disables protocols. Newer protocol versions always have highest priority.

session : is a gnutls_session_t structure.

list : is a 0 terminated list of gnutls_protocol_t elements.

Returns : GNUTLS_E_SUCCESS on success, or an error code.

gnutls_certificate_type_set_priority ()

```
int                 gnutls_certificate_type_set_priority
                                                        (gnutls_session_t session,
                                                         const int *list);
```

Sets the priority on the certificate types supported by gnutls. Priority is higher for elements specified before others. After specifying the types you want, you must append a 0. Note that the certificate type priority is set on the client. The server does not use the cert type priority except for disabling types that were not specified.

session : is a gnutls_session_t structure.

list : is a 0 terminated list of gnutls_certificate_type_t elements.

Returns : GNUTLS_E_SUCCESS on success, or an error code.

gnutls_priority_init ()

```
int                 gnutls_priority_init                (gnutls_priority_t *priority_cache,
                                                         const char *priorities,
                                                         const char **err_pos);
```

Sets priorities for the ciphers, key exchange methods, MACs and compression methods. This provides a more flexible interface compared to the gnutls_*_priority functions.

The *priorities* parameter allows you to specify a colon separated list of the cipher priorities to enable.

Unless the first keyword is "NONE" the defaults (in preference order) are for TLS protocols TLS1.1, TLS1.0, SSL3.0; for compression NULL; for certificate types X.509, OpenPGP.

For key exchange algorithms when in NORMAL or SECURE levels the perfect forward secrecy algorithms take precedence of the other protocols. In all cases all the supported key exchange algorithms are enabled (except for the RSA-EXPORT which is only enabled in EXPORT level).

Note that although one can select very long key sizes (such as 256 bits) for symmetric algorithms, to actually increase security the public key algorithms have to use longer key sizes as well.

For all the current available algorithms and protocols use "gnutls-cli -l" to get a listing.

Common keywords: Some keywords are defined to provide quick access to common preferences.

"PERFORMANCE" means all the "secure" ciphersuites are enabled, limited to 128 bit ciphers and sorted by terms of speed performance.

"NORMAL" means all "secure" ciphersuites. The 256-bit ciphers are included as a fallback only. The ciphers are sorted by security margin.

"SECURE128" means all "secure" ciphersuites with ciphers up to 128 bits, sorted by security margin.

"SECURE256" means all "secure" ciphersuites including the 256 bit ciphers, sorted by security margin.

"EXPORT" means all ciphersuites are enabled, including the low-security 40 bit ciphers.

"NONE" means nothing is enabled. This disables even protocols and compression methods.

Special keywords: "!" or "-" appended with an algorithm will remove this algorithm.

"+" appended with an algorithm will add this algorithm.

"COMPAT" will enable compatibility features for a server.

"DISABLE_SAFE_RENEGOTIATION" will disable safe renegotiation completely. Do not use unless you know what you are doing. Testing purposes only.

"UNSAFE_RENEGOTIATION" will allow handshakes and rehandshakes without the safe renegotiation extension. Note that for clients this mode is insecure (you may be under attack), and for servers it will allow insecure clients to connect (which could be fooled by an attacker). Do not use unless you know what you are doing and want maximum compatibility.

"PARTIAL_RENEGOTIATION" will allow initial handshakes to proceed, but not rehandshakes. This leaves the client vulnerable to attack, and servers will be compatible with non-upgraded clients for initial handshakes. This is currently the default for clients and servers, for compatibility reasons.

"SAFE_RENEGOTIATION" will enforce safe renegotiation. Clients and servers will refuse to talk to an insecure peer. Currently this causes operability problems, but is required for full protection.

"SSL3_RECORD_VERSION" will use SSL3.0 record version in client hello.

"VERIFY_ALLOW_SIGN_RSA_MD5" will allow RSA-MD5 signatures in certificate chains.

"VERIFY_ALLOW_X509_V1_CA_CRT" will allow V1 CAs in chains.

Namespace: To avoid collisions in order to specify a compression algorithm in this string you have to prefix it with "COMP-", protocol versions with "VERS-", signature algorithms with "SIGN-" and certificate types with "CTYPE-". Other algorithms don't need a prefix.

Examples: "NORMAL:!AES-128-CBC" means normal ciphers except for AES-128.

"EXPORT:!VERS-TLS1.0:+COMP-DEFLATE" means that export ciphers are enabled, TLS 1.0 is disabled, and libz compression enabled.

"NONE:+VERS-TLS1.0:+AES-128-CBC:+RSA:+SHA1:+COMP-NULL", "NORMAL", "COMPAT".

***priority_cache*:** is a gnutls_priorittiy_t structure.

***priorities*:** is a string describing priorities

***err_pos*:** In case of an error this will have the position in the string the error occured

***Returns*:** On syntax error GNUTLS_E_INVALID_REQUEST is returned, GNUTLS_E_SUCCESS on success, or an error code.

gnutls_priority_deinit ()

```
void                gnutls_priority_deinit              (gnutls_priority_t priority_cache);
```

Deinitializes the priority cache.

***priority_cache*:** is a gnutls_priorittiy_t structure.

gnutls_priority_set ()

```
int                 gnutls_priority_set                 (gnutls_session_t session,
                                                         gnutls_priority_t priority);
```

Sets the priorities to use on the ciphers, key exchange methods, macs and compression methods.

session : is a gnutls_session_t structure.

priority : is a gnutls_priority_t structure.

Returns : GNUTLS_E_SUCCESS on success, or an error code.

gnutls_priority_set_direct ()

```
int                 gnutls_priority_set_direct          (gnutls_session_t session,
                                                         const char *priorities,
                                                         const char **err_pos);
```

Sets the priorities to use on the ciphers, key exchange methods, macs and compression methods. This function avoids keeping a priority cache and is used to directly set string priorities to a TLS session. For documentation check the gnutls_priority_init().

session : is a gnutls_session_t structure.

priorities : is a string describing priorities

err_pos : In case of an error this will have the position in the string the error occured

Returns : On syntax error GNUTLS_E_INVALID_REQUEST is returned, GNUTLS_E_SUCCESS on success, or an error code.

gnutls_set_default_priority ()

```
int                 gnutls_set_default_priority         (gnutls_session_t session);
```

Sets some default priority on the ciphers, key exchange methods, macs and compression methods.

This is the same as calling:

gnutls_priority_set_direct (session, "NORMAL", NULL);

This function is kept around for backwards compatibility, but because of its wide use it is still fully supported. If you wish to allow users to provide a string that specify which ciphers to use (which is recommended), you should use gnutls_priority_set_direct() or gnutls_priority_set() instead.

session : is a gnutls_session_t structure.

Returns : GNUTLS_E_SUCCESS on success, or an error code.

gnutls_set_default_export_priority ()

```
int                 gnutls_set_default_export_priority  (gnutls_session_t session);
```

Sets some default priority on the ciphers, key exchange methods, macs and compression methods. This function also includes weak algorithms.

This is the same as calling:

gnutls_priority_set_direct (session, "EXPORT", NULL);

This function is kept around for backwards compatibility, but because of its wide use it is still fully supported. If you wish to allow users to provide a string that specify which ciphers to use (which is recommended), you should use gnutls_priority_set_direct() or gnutls_priority_set() instead.

session : is a gnutls_session_t structure.

Returns : GNUTLS_E_SUCCESS on success, or an error code.

gnutls_cipher_suite_get_name ()

```
const char *        gnutls_cipher_suite_get_name        (gnutls_kx_algorithm_t kx_algorithm ↩
    ,
                                                         gnutls_cipher_algorithm_t  ↩
                                                             cipher_algorithm,
                                                         gnutls_mac_algorithm_t  ↩
                                                             mac_algorithm);
```

Note that the full cipher suite name must be prepended by TLS or SSL depending of the protocol in use.

kx_algorithm : is a Key exchange algorithm

cipher_algorithm : is a cipher algorithm

mac_algorithm : is a MAC algorithm

Returns : a string that contains the name of a TLS cipher suite, specified by the given algorithms, or NULL.

gnutls_protocol_get_version ()

```
gnutls_protocol_t   gnutls_protocol_get_version         (gnutls_session_t session);
```

Get TLS version, a gnutls_protocol_t value.

session : is a gnutls_session_t structure.

Returns : the version of the currently used protocol.

gnutls_protocol_get_name ()

```
const char *        gnutls_protocol_get_name            (gnutls_protocol_t version);
```

Convert a gnutls_protocol_t value to a string.

version : is a (gnutls) version number

Returns : a string that contains the name of the specified TLS version (e.g., "TLS1.0"), or NULL.

gnutls_session_set_data ()

```
int                 gnutls_session_set_data             (gnutls_session_t session,
                                                         const void *session_data,
                                                         size_t session_data_size);
```

Sets all session parameters, in order to resume a previously established session. The session data given must be the one returned by gnutls_session_get_data(). This function should be called before gnutls_handshake().

Keep in mind that session resuming is advisory. The server may choose not to resume the session, thus a full handshake will be performed.

session : is a gnutls_session_t structure.

session_data : is a pointer to space to hold the session.

session_data_size : is the session's size

Returns : On success, GNUTLS_E_SUCCESS (0) is returned, otherwise an error code is returned.

gnutls_session_get_data ()

```
int                 gnutls_session_get_data             (gnutls_session_t session,
                                                         void *session_data,
                                                         size_t *session_data_size);
```

Returns all session parameters, in order to support resuming. The client should call this, and keep the returned session, if he wants to resume that current version later by calling gnutls_session_set_data() This function must be called after a successful handshake.

Resuming sessions is really useful and speedups connections after a successful one.

session : is a gnutls_session_t structure.

session_data : is a pointer to space to hold the session.

session_data_size : is the session_data's size, or it will be set by the function.

Returns : On success, GNUTLS_E_SUCCESS (0) is returned, otherwise an error code is returned.

gnutls_session_get_data2 ()

```
int                 gnutls_session_get_data2            (gnutls_session_t session,
                                                         gnutls_datum_t *data);
```

Returns all session parameters, in order to support resuming. The client should call this, and keep the returned session, if he wants to resume that current version later by calling gnutls_session_set_data(). This function must be called after a successful handshake. The returned datum must be freed with gnutls_free().

Resuming sessions is really useful and speedups connections after a successful one.

session : is a gnutls_session_t structure.

data : is a pointer to a datum that will hold the session.

Returns : On success, GNUTLS_E_SUCCESS (0) is returned, otherwise an error code is returned.

GNUTLS_MAX_SESSION_ID

```
#define GNUTLS_MAX_SESSION_ID 32
```

gnutls_session_get_id ()

```
int                 gnutls_session_get_id               (gnutls_session_t session,
                                                         void *session_id,
                                                         size_t *session_id_size);
```

Returns the current session id. This can be used if you want to check if the next session you tried to resume was actually resumed. This is because resumed sessions have the same sessionID with the original session.

Session id is some data set by the server, that identify the current session. In TLS 1.0 and SSL 3.0 session id is always less than 32 bytes.

session : is a gnutls_session_t structure.

session_id : is a pointer to space to hold the session id.

session_id_size : is the session id's size, or it will be set by the function.

Returns : On success, GNUTLS_E_SUCCESS (0) is returned, otherwise an error code is returned.

GNUTLS_MASTER_SIZE

```
#define GNUTLS_MASTER_SIZE 48
```

GNUTLS_RANDOM_SIZE

```
#define GNUTLS_RANDOM_SIZE 32
```

gnutls_session_get_server_random ()

```
const void *        gnutls_session_get_server_random    (gnutls_session_t session);
```

Return a pointer to the 32-byte server random field used in the session. The pointer must not be modified or deallocated.

If a server random value has not yet been established, the output will be garbage; in particular, a NULL return value should not be expected.

session : is a gnutls_session_t structure.

Returns : pointer to server random data.

gnutls_session_get_client_random ()

```
const void *        gnutls_session_get_client_random    (gnutls_session_t session);
```

Return a pointer to the 32-byte client random field used in the session. The pointer must not be modified or deallocated.

If a client random value has not yet been established, the output will be garbage; in particular, a NULL return value should not be expected.

session : is a gnutls_session_t structure.

Returns : pointer to client random data.

gnutls_session_get_master_secret ()

```
const void *        gnutls_session_get_master_secret    (gnutls_session_t session);
```

Return a pointer to the 48-byte master secret in the session. The pointer must not be modified or deallocated.

If a master secret value has not yet been established, the output will be garbage; in particular, a NULL return value should not be expected.

Consider using gnutls_prf() rather than extracting the master secret and use it to derive further data.

session : is a gnutls_session_t structure.

Returns : pointer to master secret data.

gnutls_finished_callback_func ()

```
void                (*gnutls_finished_callback_func)    (gnutls_session_t session,
                                                         const void *finished,
                                                         size_t len);
```

session :

finished :

len :

gnutls_session_set_finished_function ()

```
void                gnutls_session_set_finished_function
                                                        (gnutls_session_t session,
                                                         gnutls_finished_callback_func func ↩
                                                             );
```

Register a callback function for the session that will be called when a TLS Finished message has been generated. The function is typically used to copy away the TLS finished message for later use as a channel binding or similar purpose.

The callback should follow this prototype:

void callback (gnutls_session_t *session*, const void **finished*, size_t *len*);

The *finished* parameter will contain the binary TLS finished message, and *len* will contains its length. For SSLv3 connections, the *len* parameter will be 36 and for TLS connections it will be 12.

It is recommended that the function returns quickly in order to not delay the handshake. Use the function to store a copy of the TLS finished message for later use.

session : is a gnutls_session_t structure.

func : a gnutls_finished_callback_func callback.

Since 2.6.0

gnutls_session_is_resumed ()

```
int                 gnutls_session_is_resumed           (gnutls_session_t session);
```

Check whether session is resumed or not.

session : is a gnutls_session_t structure.

Returns : non zero if this session is resumed, or a zero if this is a new session.

gnutls_db_store_func ()

```
int                 (*gnutls_db_store_func)             (void *Param1,
                                                         gnutls_datum_t key,
                                                         gnutls_datum_t data);
```

Param1 :

key :

data :

Returns :

gnutls_db_remove_func ()

```
int                 (*gnutls_db_remove_func)            (void *Param1,
                                                         gnutls_datum_t key);
```

Param1 :

key :

Returns :

gnutls_db_retr_func ()

```
gnutls_datum_t      (*gnutls_db_retr_func)              (void *Param1,
                                                         gnutls_datum_t key);
```

Param1 :

key :

Returns :

gnutls_db_set_cache_expiration ()

```
void                gnutls_db_set_cache_expiration      (gnutls_session_t session,
                                                         int seconds);
```

Set the expiration time for resumed sessions. The default is 3600 (one hour) at the time writing this.

session : is a gnutls_session_t structure.

seconds : is the number of seconds.

gnutls_db_remove_session ()

```
void                gnutls_db_remove_session            (gnutls_session_t session);
```

This function will remove the current session data from the session database. This will prevent future handshakes reusing these session data. This function should be called if a session was terminated abnormally, and before gnutls_deinit() is called.

Normally gnutls_deinit() will remove abnormally terminated sessions.

session : is a gnutls_session_t structure.

gnutls_db_set_retrieve_function ()

```
void                gnutls_db_set_retrieve_function     (gnutls_session_t session,
                                                         gnutls_db_retr_func retr_func);
```

Sets the function that will be used to retrieve data from the resumed sessions database. This function must return a gnutls_datum_t containing the data on success, or a gnutls_datum_t containing null and 0 on failure.

The datum's data must be allocated using the function gnutls_malloc().

The first argument to *retr_func* will be null unless gnutls_db_set_ptr() has been called.

session : is a gnutls_session_t structure.

retr_func : is the function.

gnutls_db_set_remove_function ()

```
void                gnutls_db_set_remove_function       (gnutls_session_t session,
                                                         gnutls_db_remove_func rem_func);
```

Sets the function that will be used to remove data from the resumed sessions database. This function must return 0 on success.

The first argument to *rem_func* will be null unless gnutls_db_set_ptr() has been called.

session : is a gnutls_session_t structure.

rem_func : is the function.

gnutls_db_set_store_function ()

```
void                gnutls_db_set_store_function        (gnutls_session_t session,
                                                         gnutls_db_store_func store_func);
```

Sets the function that will be used to store data from the resumed sessions database. This function must remove 0 on success.

The first argument to store_func() will be null unless gnutls_db_set_ptr() has been called.

session : is a gnutls_session_t structure.

store_func : is the function

gnutls_db_set_ptr ()

```
void                gnutls_db_set_ptr                   (gnutls_session_t session,
                                                         void *ptr);
```

Sets the pointer that will be provided to db store, retrieve and delete functions, as the first argument.

session : is a gnutls_session_t structure.

ptr : is the pointer

gnutls_db_get_ptr ()

```
void *              gnutls_db_get_ptr                   (gnutls_session_t session);
```

Get db function pointer.

session : is a gnutls_session_t structure.

Returns : the pointer that will be sent to db store, retrieve and delete functions, as the first argument.

gnutls_db_check_entry ()

```
int                 gnutls_db_check_entry               (gnutls_session_t session,
                                                         gnutls_datum_t session_entry);
```

Check if database entry has expired. This function is to be used when you want to clear unnesessary session which occupy space in your backend.

session : is a gnutls_session_t structure.

session_entry : is the session data (not key)

Returns : Returns GNUTLS_E_EXPIRED, if the database entry has expired or 0 otherwise.

gnutls_handshake_post_client_hello_func ()

```
int                 (*gnutls_handshake_post_client_hello_func)
                                                    (gnutls_session_t Param1);
```

Param1 :

Returns :

gnutls_handshake_set_post_client_hello_function ()

```
void                gnutls_handshake_set_post_client_hello_function
                                                    (gnutls_session_t session,
                                                     gnutls_handshake_post_client_hello_func ↩
                                                          func);
```

This function will set a callback to be called after the client hello has been received (callback valid in server side only). This allows the server to adjust settings based on received extensions.

Those settings could be ciphersuites, requesting certificate, or anything else except for version negotiation (this is done before the hello message is parsed).

This callback must return 0 on success or a gnutls error code to terminate the handshake.

Warning: You should not use this function to terminate the handshake based on client input unless you know what you are doing. Before the handshake is finished there is no way to know if there is a man-in-the-middle attack being performed.

session : is a gnutls_session_t structure.

func : is the function to be called

gnutls_handshake_set_max_packet_length ()

```
void                gnutls_handshake_set_max_packet_length
                                                    (gnutls_session_t session,
                                                     size_t max);
```

This function will set the maximum size of all handshake messages. Handshakes over this size are rejected with GNUTLS_E_HANDSHA error code. The default value is 48kb which is typically large enough. Set this to 0 if you do not want to set an upper limit.

The reason for restricting the handshake message sizes are to limit Denial of Service attacks.

session : is a gnutls_session_t structure.

max : is the maximum number.

gnutls_check_version ()

```
const char *        gnutls_check_version            (const char *req_version);
```

Check GnuTLS Library version.

See GNUTLS_VERSION for a suitable *req_version* string.

req_version : version string to compare with, or NULL.

Returns : Check that the version of the library is at minimum the one given as a string in *req_version* and return the actual version string of the library; return NULL if the condition is not met. If NULL is passed to this function no check is done and only the version string is returned.

gnutls_credentials_clear ()

```
void                gnutls_credentials_clear            (gnutls_session_t session);
```

Clears all the credentials previously set in this session.

session : is a gnutls_session_t structure.

gnutls_credentials_set ()

```
int                 gnutls_credentials_set              (gnutls_session_t session,
                                                         gnutls_credentials_type_t type,
                                                         void *cred);
```

Sets the needed credentials for the specified type. Eg username, password - or public and private keys etc. The *cred* parameter is a structure that depends on the specified type and on the current session (client or server).

In order to minimize memory usage, and share credentials between several threads gnutls keeps a pointer to cred, and not the whole cred structure. Thus you will have to keep the structure allocated until you call gnutls_deinit().

For GNUTLS_CRD_ANON, *cred* should be gnutls_anon_client_credentials_t in case of a client. In case of a server it should be gnutls_anon_server_credentials_t.

For GNUTLS_CRD_SRP, *cred* should be gnutls_srp_client_credentials_t in case of a client, and gnutls_srp_server_credentials_t, in case of a server.

For GNUTLS_CRD_CERTIFICATE, *cred* should be gnutls_certificate_credentials_t.

session : is a gnutls_session_t structure.

type : is the type of the credentials

cred : is a pointer to a structure.

Returns : On success, GNUTLS_E_SUCCESS (zero) is returned, otherwise an error code is returned.

gnutls_cred_set

```
#define gnutls_cred_set~gnutls_credentials_set
```

struct gnutls_certificate_credentials_st

```
struct gnutls_certificate_credentials_st;
```

gnutls_anon_free_server_credentials ()

```
void                gnutls_anon_free_server_credentials (gnutls_anon_server_credentials_t ↩
    sc);
```

This structure is complex enough to manipulate directly thus this helper function is provided in order to free (deallocate) it.

sc : is a gnutls_anon_server_credentials_t structure.

gnutls_anon_allocate_server_credentials ()

```
int                 gnutls_anon_allocate_server_credentials
                                                    (gnutls_anon_server_credentials_t * ↩
                                                        sc);
```

This structure is complex enough to manipulate directly thus this helper function is provided in order to allocate it.

sc : is a pointer to a gnutls_anon_server_credentials_t structure.

Returns : GNUTLS_E_SUCCESS on success, or an error code.

gnutls_anon_set_server_dh_params ()

```
void                gnutls_anon_set_server_dh_params    (gnutls_anon_server_credentials_t  ↩
    res,
                                                         gnutls_dh_params_t dh_params);
```

This function will set the Diffie-Hellman parameters for an anonymous server to use. These parameters will be used in Anonymous Diffie-Hellman cipher suites.

res : is a gnutls_anon_server_credentials_t structure

dh_params : is a structure that holds Diffie-Hellman parameters.

gnutls_anon_set_server_params_function ()

```
void                gnutls_anon_set_server_params_function
                                                    (gnutls_anon_server_credentials_t  ↩
                                                        res,
                                                     gnutls_params_function *func);
```

This function will set a callback in order for the server to get the Diffie-Hellman parameters for anonymous authentication. The callback should return zero on success.

res : is a gnutls_certificate_credentials_t structure

func : is the function to be called

gnutls_anon_free_client_credentials ()

```
void                gnutls_anon_free_client_credentials (gnutls_anon_client_credentials_t  ↩
    sc);
```

This structure is complex enough to manipulate directly thus this helper function is provided in order to free (deallocate) it.

sc : is a gnutls_anon_client_credentials_t structure.

gnutls_anon_allocate_client_credentials ()

```
int                 gnutls_anon_allocate_client_credentials
                                                    (gnutls_anon_client_credentials_t * ↩
                                                        sc);
```

This structure is complex enough to manipulate directly thus this helper function is provided in order to allocate it.

sc : is a pointer to a gnutls_anon_client_credentials_t structure.

Returns : GNUTLS_E_SUCCESS on success, or an error code.

gnutls_certificate_free_credentials ()

```
void                gnutls_certificate_free_credentials (gnutls_certificate_credentials_t ↩
    sc);
```

This structure is complex enough to manipulate directly thus this helper function is provided in order to free (deallocate) it.

This function does not free any temporary parameters associated with this structure (ie RSA and DH parameters are not freed by this function).

sc : is a gnutls_certificate_credentials_t structure.

gnutls_certificate_allocate_credentials ()

```
int                 gnutls_certificate_allocate_credentials
                                                    (gnutls_certificate_credentials_t * ↩
                                                       res);
```

This structure is complex enough to manipulate directly thus this helper function is provided in order to allocate it.

res : is a pointer to a gnutls_certificate_credentials_t structure.

Returns : GNUTLS_E_SUCCESS on success, or an error code.

gnutls_certificate_free_keys ()

```
void                gnutls_certificate_free_keys        (gnutls_certificate_credentials_t ↩
    sc);
```

This function will delete all the keys and the certificates associated with the given credentials. This function must not be called when a TLS negotiation that uses the credentials is in progress.

sc : is a gnutls_certificate_credentials_t structure.

gnutls_certificate_free_cas ()

```
void                gnutls_certificate_free_cas         (gnutls_certificate_credentials_t ↩
    sc);
```

This function will delete all the CAs associated with the given credentials. Servers that do not use gnutls_certificate_verify_peers2() may call this to save some memory.

sc : is a gnutls_certificate_credentials_t structure.

gnutls_certificate_free_ca_names ()

```
void                gnutls_certificate_free_ca_names    (gnutls_certificate_credentials_t ↩
    sc);
```

This function will delete all the CA name in the given credentials. Clients may call this to save some memory since in client side the CA names are not used.

CA names are used by servers to advertize the CAs they support to clients.

sc : is a gnutls_certificate_credentials_t structure.

gnutls_certificate_free_crls ()

```
void                gnutls_certificate_free_crls        (gnutls_certificate_credentials_t  ↩
    sc);
```

This function will delete all the CRLs associated with the given credentials.

sc : is a gnutls_certificate_credentials_t structure.

gnutls_certificate_set_dh_params ()

```
void                gnutls_certificate_set_dh_params    (gnutls_certificate_credentials_t  ↩
    res,
                                                         gnutls_dh_params_t dh_params);
```

This function will set the Diffie-Hellman parameters for a certificate server to use. These parameters will be used in Ephemeral Diffie-Hellman cipher suites. Note that only a pointer to the parameters are stored in the certificate handle, so if you deallocate the parameters before the certificate is deallocated, you must change the parameters stored in the certificate first.

res : is a gnutls_certificate_credentials_t structure

dh_params : is a structure that holds Diffie-Hellman parameters.

gnutls_certificate_set_rsa_export_params ()

```
void                gnutls_certificate_set_rsa_export_params
                                                        (gnutls_certificate_credentials_t  ↩
                                                           res,
                                                         gnutls_rsa_params_t rsa_params);
```

This function will set the temporary RSA parameters for a certificate server to use. These parameters will be used in RSA-EXPORT cipher suites.

res : is a gnutls_certificate_credentials_t structure

rsa_params : is a structure that holds temporary RSA parameters.

gnutls_certificate_set_verify_flags ()

```
void                gnutls_certificate_set_verify_flags (gnutls_certificate_credentials_t  ↩
    res,
                                                         unsigned int flags);
```

This function will set the flags to be used at verification of the certificates. Flags must be OR of the gnutls_certificate_verify_flags enumerations.

res : is a gnutls_certificate_credentials_t structure

flags : are the flags

gnutls_certificate_set_verify_limits ()

```
void                gnutls_certificate_set_verify_limits
                                                    (gnutls_certificate_credentials_t ↩
                                                        res,
                                                     unsigned int max_bits,
                                                     unsigned int max_depth);
```

This function will set some upper limits for the default verification function, gnutls_certificate_verify_peers2(), to avoid denial of service attacks. You can set them to zero to disable limits.

res : is a gnutls_certificate_credentials structure

max_bits : is the number of bits of an acceptable certificate (default 8200)

max_depth : is maximum depth of the verification of a certificate chain (default 5)

gnutls_certificate_set_x509_trust_file ()

```
int                 gnutls_certificate_set_x509_trust_file
                                                    (gnutls_certificate_credentials_t ↩
                                                        res,
                                                     const char *cafile,
                                                     gnutls_x509_crt_fmt_t type);
```

This function adds the trusted CAs in order to verify client or server certificates. In case of a client this is not required to be called if the certificates are not verified using gnutls_certificate_verify_peers2(). This function may be called multiple times.

In case of a server the names of the CAs set here will be sent to the client if a certificate request is sent. This can be disabled using gnutls_certificate_send_x509_rdn_sequence().

res : is a gnutls_certificate_credentials_t structure.

cafile : is a file containing the list of trusted CAs (DER or PEM list)

type : is PEM or DER

Returns : number of certificates processed, or a negative value on error.

gnutls_certificate_set_x509_trust_mem ()

```
int                 gnutls_certificate_set_x509_trust_mem
                                                    (gnutls_certificate_credentials_t ↩
                                                        res,
                                                     const gnutls_datum_t *ca,
                                                     gnutls_x509_crt_fmt_t type);
```

This function adds the trusted CAs in order to verify client or server certificates. In case of a client this is not required to be called if the certificates are not verified using gnutls_certificate_verify_peers2(). This function may be called multiple times.

In case of a server the CAs set here will be sent to the client if a certificate request is sent. This can be disabled using gnutls_certificate_send_x509_rdn_sequence().

res : is a gnutls_certificate_credentials_t structure.

ca : is a list of trusted CAs or a DER certificate

type : is DER or PEM

Returns : the number of certificates processed or a negative value on error.

gnutls_certificate_set_x509_crl_file ()

```
int                 gnutls_certificate_set_x509_crl_file
                                                    (gnutls_certificate_credentials_t  ↩
                                                       res,
                                                     const char *crlfile,
                                                     gnutls_x509_crt_fmt_t type);
```

This function adds the trusted CRLs in order to verify client or server certificates. In case of a client this is not required to be called if the certificates are not verified using gnutls_certificate_verify_peers2(). This function may be called multiple times.

res : is a gnutls_certificate_credentials_t structure.

crlfile : is a file containing the list of verified CRLs (DER or PEM list)

type : is PEM or DER

Returns : number of CRLs processed or a negative value on error.

gnutls_certificate_set_x509_crl_mem ()

```
int                 gnutls_certificate_set_x509_crl_mem (gnutls_certificate_credentials_t  ↩
    res,
                                                     const gnutls_datum_t *CRL,
                                                     gnutls_x509_crt_fmt_t type);
```

This function adds the trusted CRLs in order to verify client or server certificates. In case of a client this is not required to be called if the certificates are not verified using gnutls_certificate_verify_peers2(). This function may be called multiple times.

res : is a gnutls_certificate_credentials_t structure.

CRL : is a list of trusted CRLs. They should have been verified before.

type : is DER or PEM

Returns : number of CRLs processed, or a negative value on error.

gnutls_certificate_set_x509_key_file ()

```
int                 gnutls_certificate_set_x509_key_file
                                                    (gnutls_certificate_credentials_t  ↩
                                                       res,
                                                     const char *certfile,
                                                     const char *keyfile,
                                                     gnutls_x509_crt_fmt_t type);
```

This function sets a certificate/private key pair in the gnutls_certificate_credentials_t structure. This function may be called more than once (in case multiple keys/certificates exist for the server). For clients that wants to send more than its own end entity certificate (e.g., also an intermediate CA cert) then put the certificate chain in `certfile`.

Currently only PKCS-1 encoded RSA and DSA private keys are accepted by this function.

res : is a gnutls_certificate_credentials_t structure.

certfile : is a file that containing the certificate list (path) for the specified private key, in PKCS7 format, or a list of certificates

keyfile : is a file that contains the private key

type : is PEM or DER

Returns : GNUTLS_E_SUCCESS on success, or an error code.

gnutls_certificate_set_x509_key_mem ()

```
int                 gnutls_certificate_set_x509_key_mem (gnutls_certificate_credentials_t  ↩
    res,
                                                         const gnutls_datum_t *cert,
                                                         const gnutls_datum_t *key,
                                                         gnutls_x509_crt_fmt_t type);
```

This function sets a certificate/private key pair in the gnutls_certificate_credentials_t structure. This function may be called more than once (in case multiple keys/certificates exist for the server).

Currently are supported: RSA PKCS-1 encoded private keys, DSA private keys.

DSA private keys are encoded the OpenSSL way, which is an ASN.1 DER sequence of 6 INTEGERs - version, p, q, g, pub, priv.

Note that the keyUsage (2.5.29.15) PKIX extension in X.509 certificates is supported. This means that certificates intended for signing cannot be used for ciphersuites that require encryption.

If the certificate and the private key are given in PEM encoding then the strings that hold their values must be null terminated.

The *key* may be NULL if you are using a sign callback, see gnutls_sign_callback_set().

res : is a gnutls_certificate_credentials_t structure.

cert : contains a certificate list (path) for the specified private key

key : is the private key, or NULL

type : is PEM or DER

Returns : GNUTLS_E_SUCCESS on success, or an error code.

gnutls_certificate_send_x509_rdn_sequence ()

```
void                gnutls_certificate_send_x509_rdn_sequence
                                                        (gnutls_session_t session,
                                                         int status);
```

If status is non zero, this function will order gnutls not to send the rdnSequence in the certificate request message. That is the server will not advertize it's trusted CAs to the peer. If status is zero then the default behaviour will take effect, which is to advertize the server's trusted CAs.

This function has no effect in clients, and in authentication methods other than certificate with X.509 certificates.

session : is a pointer to a gnutls_session_t structure.

status : is 0 or 1

gnutls_certificate_set_x509_simple_pkcs12_file ()

```
int                 gnutls_certificate_set_x509_simple_pkcs12_file
                                                        (gnutls_certificate_credentials_t  ↩
                                                            res,
                                                         const char *pkcs12file,
                                                         gnutls_x509_crt_fmt_t type,
                                                         const char *password);
```

This function sets a certificate/private key pair and/or a CRL in the gnutls_certificate_credentials_t structure. This function may be called more than once (in case multiple keys/certificates exist for the server).

MAC:ed PKCS12 files are supported. Encrypted PKCS12 bags are supported. Encrypted PKCS#8 private keys are supported. However, only password based security, and the same password for all operations, are supported.

The private keys may be RSA PKCS#1 or DSA private keys encoded in the OpenSSL way.

PKCS12 file may contain many keys and/or certificates, and there is no way to identify which key/certificate pair you want. You should make sure the PKCS12 file only contain one key/certificate pair and/or one CRL.

It is believed that the limitations of this function is acceptable for most usage, and that any more flexibility would introduce complexity that would make it harder to use this functionality at all.

res : is a gnutls_certificate_credentials_t structure.

pkcs12file : filename of file containing PKCS12 blob.

type : is PEM or DER of the *pkcs12file*.

password : optional password used to decrypt PKCS12 file, bags and keys.

Returns : GNUTLS_E_SUCCESS on success, or an error code.

gnutls_certificate_set_x509_simple_pkcs12_mem ()

```
int                 gnutls_certificate_set_x509_simple_pkcs12_mem
                                                    (gnutls_certificate_credentials_t  ↩
                                                       res,
                                                     const gnutls_datum_t *p12blob,
                                                     gnutls_x509_crt_fmt_t type,
                                                     const char *password);
```

This function sets a certificate/private key pair and/or a CRL in the gnutls_certificate_credentials_t structure. This function may be called more than once (in case multiple keys/certificates exist for the server).

MAC:ed PKCS12 files are supported. Encrypted PKCS12 bags are supported. Encrypted PKCS#8 private keys are supported. However, only password based security, and the same password for all operations, are supported.

The private keys may be RSA PKCS#1 or DSA private keys encoded in the OpenSSL way.

PKCS12 file may contain many keys and/or certificates, and there is no way to identify which key/certificate pair you want. You should make sure the PKCS12 file only contain one key/certificate pair and/or one CRL.

It is believed that the limitations of this function is acceptable for most usage, and that any more flexibility would introduce complexity that would make it harder to use this functionality at all.

res : is a gnutls_certificate_credentials_t structure.

p12blob : the PKCS12 blob.

type : is PEM or DER of the *pkcs12file*.

password : optional password used to decrypt PKCS12 file, bags and keys.

Returns : GNUTLS_E_SUCCESS on success, or an error code.

Since 2.8.0

gnutls_x509_privkey_t

```
typedef struct gnutls_x509_privkey_int *gnutls_x509_privkey_t;
```

struct gnutls_x509_crl_int

```
struct gnutls_x509_crl_int;
```

gnutls_x509_crl_t

```
  typedef struct gnutls_x509_crl_int *gnutls_x509_crl_t;
```

struct gnutls_x509_crt_int

```
struct gnutls_x509_crt_int;
```

gnutls_x509_crt_t

```
  typedef struct gnutls_x509_crt_int *gnutls_x509_crt_t;
```

struct gnutls_openpgp_keyring_int

```
struct gnutls_openpgp_keyring_int;
```

gnutls_openpgp_keyring_t

```
  typedef struct gnutls_openpgp_keyring_int *gnutls_openpgp_keyring_t;
```

gnutls_certificate_set_x509_key ()

```
int                 gnutls_certificate_set_x509_key     (gnutls_certificate_credentials_t  ↩
    res,
                                                         gnutls_x509_crt_t *cert_list,
                                                         int cert_list_size,
                                                         gnutls_x509_privkey_t key);
```

This function sets a certificate/private key pair in the gnutls_certificate_credentials_t structure. This function may be called more than once (in case multiple keys/certificates exist for the server). For clients that wants to send more than its own end entity certificate (e.g., also an intermediate CA cert) then put the certificate chain in *cert_list*.

res : is a gnutls_certificate_credentials_t structure.

cert_list : contains a certificate list (path) for the specified private key

cert_list_size : holds the size of the certificate list

key : is a gnutls_x509_privkey_t key

Returns : GNUTLS_E_SUCCESS on success, or an error code.

Since 2.4.0

gnutls_certificate_set_x509_trust ()

```
int                 gnutls_certificate_set_x509_trust   (gnutls_certificate_credentials_t  ↩
    res,
                                                         gnutls_x509_crt_t *ca_list,
                                                         int ca_list_size);
```

This function adds the trusted CAs in order to verify client or server certificates. In case of a client this is not required to be called if the certificates are not verified using gnutls_certificate_verify_peers2(). This function may be called multiple times.

In case of a server the CAs set here will be sent to the client if a certificate request is sent. This can be disabled using gnutls_certificate_send_x509_rdn_sequence().

res : is a gnutls_certificate_credentials_t structure.

ca_list : is a list of trusted CAs

ca_list_size : holds the size of the CA list

Returns : GNUTLS_E_SUCCESS on success, or an error code.

Since 2.4.0

gnutls_certificate_set_x509_crl ()

```
int                 gnutls_certificate_set_x509_crl     (gnutls_certificate_credentials_t  ↩
    res,
                                                         gnutls_x509_crl_t *crl_list,
                                                         int crl_list_size);
```

This function adds the trusted CRLs in order to verify client or server certificates. In case of a client this is not required to be called if the certificates are not verified using gnutls_certificate_verify_peers2(). This function may be called multiple times.

res : is a gnutls_certificate_credentials_t structure.

crl_list : is a list of trusted CRLs. They should have been verified before.

crl_list_size : holds the size of the crl_list

Returns : GNUTLS_E_SUCCESS on success, or an error code.

Since 2.4.0

gnutls_certificate_get_x509_cas ()

```
void                gnutls_certificate_get_x509_cas     (gnutls_certificate_credentials_t  ↩
    sc,
                                                         gnutls_x509_crt_t **x509_ca_list,
                                                         unsigned int *ncas);
```

This function will export all the CAs associated with the given credentials.

sc : is a gnutls_certificate_credentials_t structure.

x509_ca_list : will point to the CA list. Should be treated as constant

ncas : the number of CAs

Since 2.4.0

gnutls_certificate_get_x509_crls ()

```
void                gnutls_certificate_get_x509_crls    (gnutls_certificate_credentials_t  ↩
    sc,
                                                         gnutls_x509_crl_t **x509_crl_list,
                                                         unsigned int *ncrls);
```

This function will export all the CRLs associated with the given credentials.

sc : is a gnutls_certificate_credentials_t structure.

x509_crl_list : the exported CRL list. Should be treated as constant

ncrls : the number of exported CRLs

Since 2.4.0

gnutls_certificate_get_openpgp_keyring ()

```
void                gnutls_certificate_get_openpgp_keyring
                                                        (gnutls_certificate_credentials_t  ↩
                                                            sc,
                                                         gnutls_openpgp_keyring_t *keyring) ↩
                                                            ;
```

This function will export the OpenPGP keyring associated with the given credentials.

sc : is a gnutls_certificate_credentials_t structure.

keyring : the exported keyring. Should be treated as constant

Since 2.4.0

gnutls_global_init ()

```
int                 gnutls_global_init                  (void);
```

This function initializes the global data to defaults. Every gnutls application has a global data which holds common parameters shared by gnutls session structures. You should call gnutls_global_deinit() when gnutls usage is no longer needed

Note that this function will also initialize libgcrypt, if it has not been initialized before. Thus if you want to manually initialize libgcrypt you must do it before calling this function. This is useful in cases you want to disable libgcrypt's internal lockings etc.

This function increment a global counter, so that gnutls_global_deinit() only releases resources when it has been called as many times as gnutls_global_init(). This is useful when GnuTLS is used by more than one library in an application. This function can be called many times, but will only do something the first time.

Note! This function is not thread safe. If two threads call this function simultaneously, they can cause a race between checking the global counter and incrementing it, causing both threads to execute the library initialization code. That would lead to a memory leak. To handle this, your application could invoke this function after aquiring a thread mutex. To ignore the potential memory leak is also an option.

Returns : On success, GNUTLS_E_SUCCESS (zero) is returned, otherwise an error code is returned.

gnutls_global_deinit ()

```
void                gnutls_global_deinit                (void);
```

This function deinitializes the global data, that were initialized using gnutls_global_init().

Note! This function is not thread safe. See the discussion for gnutls_global_init() for more information.

gnutls_alloc_function ()

```
void *              (*gnutls_alloc_function)            (size_t Param1);
```

Param1 :

Returns :

gnutls_calloc_function ()

```
void *              (*gnutls_calloc_function)           (size_t Param1,
                                                         size_t Param2);
```

Param1 :

Param2 :

Returns :

gnutls_is_secure_function ()

```
int                 (*gnutls_is_secure_function)        (const void *Param1);
```

Param1 :

Returns :

gnutls_free_function ()

```
void                (*gnutls_free_function)             (void *Param1);
```

Param1 :

gnutls_realloc_function ()

```
void *              (*gnutls_realloc_function)          (void *Param1,
                                                         size_t Param2);
```

Param1 :

Param2 :

Returns :

gnutls_global_set_mem_functions ()

```
void                gnutls_global_set_mem_functions     (gnutls_alloc_function alloc_func,
                                                         gnutls_alloc_function  ↩
                                                             secure_alloc_func,
                                                         gnutls_is_secure_function  ↩
                                                             is_secure_func,
                                                         gnutls_realloc_function  ↩
                                                             realloc_func,
                                                         gnutls_free_function free_func);
```

This is the function were you set the memory allocation functions gnutls is going to use. By default the libc's allocation functions (malloc(), free()), are used by gnutls, to allocate both sensitive and not sensitive data. This function is provided to set the memory allocation functions to something other than the defaults (ie the gcrypt allocation functions).

This function must be called before gnutls_global_init() is called. This function is not thread safe.

***alloc_func*:** it's the default memory allocation function. Like malloc().

***secure_alloc_func*:** This is the memory allocation function that will be used for sensitive data.

***is_secure_func*:** a function that returns 0 if the memory given is not secure. May be NULL.

***realloc_func*:** A realloc function

***free_func*:** The function that frees allocated data. Must accept a NULL pointer.

gnutls_malloc

```
extern gnutls_alloc_function gnutls_malloc;
```

This function will allocate 's' bytes data, and return a pointer to memory. This function is supposed to be used by callbacks.

The allocation function used is the one set by gnutls_global_set_mem_functions().

gnutls_secure_malloc

```
extern gnutls_alloc_function gnutls_secure_malloc;
```

gnutls_realloc

```
extern gnutls_realloc_function gnutls_realloc;
```

gnutls_calloc

```
extern gnutls_calloc_function gnutls_calloc;
```

gnutls_free

```
extern gnutls_free_function gnutls_free;
```

This function will free data pointed by ptr.

The deallocation function used is the one set by gnutls_global_set_mem_functions().

gnutls_strdup ()

```
char *                (*gnutls_strdup)                        (const char *Param1);
```

Param1 :

Returns :

gnutls_log_func ()

```
void                  (*gnutls_log_func)                      (...,
                                                               const char *Param2);
```

... :

Param2 :

gnutls_global_set_log_function ()

```
void                  gnutls_global_set_log_function          (gnutls_log_func log_func);
```

This is the function where you set the logging function gnutls is going to use. This function only accepts a character array. Normally you may not use this function since it is only used for debugging purposes.

gnutls_log_func is of the form, void (*gnutls_log_func)(int level, const char*);

log_func : it's a log function

gnutls_global_set_log_level ()

```
void                  gnutls_global_set_log_level             (int level);
```

This is the function that allows you to set the log level. The level is an integer between 0 and 9. Higher values mean more verbosity. The default value is 0. Larger values should only be used with care, since they may reveal sensitive information.

Use a log level over 10 to enable all debugging options.

level : it's an integer from 0 to 9.

gnutls_dh_params_init ()

```
int                   gnutls_dh_params_init                   (gnutls_dh_params_t *dh_params);
```

This function will initialize the DH parameters structure.

dh_params : Is a structure that will hold the prime numbers

Returns : On success, GNUTLS_E_SUCCESS (zero) is returned, otherwise an error code is returned.

gnutls_dh_params_deinit ()

```
void                  gnutls_dh_params_deinit                 (gnutls_dh_params_t dh_params);
```

This function will deinitialize the DH parameters structure.

dh_params : Is a structure that holds the prime numbers

gnutls_dh_params_import_raw ()

```
int                 gnutls_dh_params_import_raw         (gnutls_dh_params_t dh_params,
                                                         const gnutls_datum_t *prime,
                                                         const gnutls_datum_t *generator);
```

This function will replace the pair of prime and generator for use in the Diffie-Hellman key exchange. The new parameters should be stored in the appropriate gnutls_datum.

dh_params : Is a structure that will hold the prime numbers

prime : holds the new prime

generator : holds the new generator

Returns : On success, GNUTLS_E_SUCCESS (zero) is returned, otherwise an error code is returned.

gnutls_dh_params_import_pkcs3 ()

```
int                 gnutls_dh_params_import_pkcs3       (gnutls_dh_params_t params,
                                                         const gnutls_datum_t *pkcs3_params ↩
                                                             ,
                                                         gnutls_x509_crt_fmt_t format);
```

This function will extract the DHParams found in a PKCS3 formatted structure. This is the format generated by "openssl dhparam" tool.

If the structure is PEM encoded, it should have a header of "BEGIN DH PARAMETERS".

params : A structure where the parameters will be copied to

pkcs3_params : should contain a PKCS3 DHParams structure PEM or DER encoded

format : the format of params. PEM or DER.

Returns : On success, GNUTLS_E_SUCCESS (zero) is returned, otherwise an error code is returned.

gnutls_dh_params_generate2 ()

```
int                 gnutls_dh_params_generate2          (gnutls_dh_params_t params,
                                                         unsigned int bits);
```

This function will generate a new pair of prime and generator for use in the Diffie-Hellman key exchange. The new parameters will be allocated using gnutls_malloc() and will be stored in the appropriate datum. This function is normally slow.

Note that the bits value should be one of 768, 1024, 2048, 3072 or 4096. Also note that the DH parameters are only useful to servers. Since clients use the parameters sent by the server, it's of no use to call this in client side.

params : Is the structure that the DH parameters will be stored

bits : is the prime's number of bits

Returns : On success, GNUTLS_E_SUCCESS (zero) is returned, otherwise an error code is returned.

gnutls_dh_params_export_pkcs3 ()

```
int                 gnutls_dh_params_export_pkcs3       (gnutls_dh_params_t params,
                                                         gnutls_x509_crt_fmt_t format,
                                                         unsigned char *params_data,
                                                         size_t *params_data_size);
```

This function will export the given dh parameters to a PKCS3 DHParams structure. This is the format generated by "openssl dhparam" tool. If the buffer provided is not long enough to hold the output, then GNUTLS_E_SHORT_MEMORY_BUFFER will be returned.

If the structure is PEM encoded, it will have a header of "BEGIN DH PARAMETERS".

params : Holds the DH parameters

format : the format of output params. One of PEM or DER.

params_data : will contain a PKCS3 DHParams structure PEM or DER encoded

params_data_size : holds the size of params_data (and will be replaced by the actual size of parameters)

Returns : On success, GNUTLS_E_SUCCESS (zero) is returned, otherwise an error code is returned.

gnutls_dh_params_export_raw ()

```
int                 gnutls_dh_params_export_raw         (gnutls_dh_params_t params,
                                                         gnutls_datum_t *prime,
                                                         gnutls_datum_t *generator,
                                                         unsigned int *bits);
```

This function will export the pair of prime and generator for use in the Diffie-Hellman key exchange. The new parameters will be allocated using gnutls_malloc() and will be stored in the appropriate datum.

params : Holds the DH parameters

prime : will hold the new prime

generator : will hold the new generator

bits : if non null will hold is the prime's number of bits

Returns : On success, GNUTLS_E_SUCCESS (zero) is returned, otherwise an error code is returned.

gnutls_dh_params_cpy ()

```
int                 gnutls_dh_params_cpy                (gnutls_dh_params_t dst,
                                                         gnutls_dh_params_t src);
```

This function will copy the DH parameters structure from source to destination.

dst : Is the destination structure, which should be initialized.

src : Is the source structure

Returns : On success, GNUTLS_E_SUCCESS (zero) is returned, otherwise an error code is returned.

gnutls_rsa_params_init ()

```
int                 gnutls_rsa_params_init              (gnutls_rsa_params_t *rsa_params);
```

This function will initialize the temporary RSA parameters structure.

rsa_params : Is a structure that will hold the parameters

Returns : GNUTLS_E_SUCCESS on success, or an negative error code.

gnutls_rsa_params_deinit ()

```
void                gnutls_rsa_params_deinit            (gnutls_rsa_params_t rsa_params);
```

This function will deinitialize the RSA parameters structure.

rsa_params : Is a structure that holds the parameters

gnutls_rsa_params_cpy ()

```
int                 gnutls_rsa_params_cpy               (gnutls_rsa_params_t dst,
                                                         gnutls_rsa_params_t src);
```

This function will copy the RSA parameters structure from source to destination.

dst : Is the destination structure, which should be initialized.

src : Is the source structure

Returns : GNUTLS_E_SUCCESS on success, or an negative error code.

gnutls_rsa_params_import_raw ()

```
int                 gnutls_rsa_params_import_raw        (gnutls_rsa_params_t rsa_params,
                                                         const gnutls_datum_t *m,
                                                         const gnutls_datum_t *e,
                                                         const gnutls_datum_t *d,
                                                         const gnutls_datum_t *p,
                                                         const gnutls_datum_t *q,
                                                         const gnutls_datum_t *u);
```

This function will replace the parameters in the given structure. The new parameters should be stored in the appropriate gnutls_datum.

rsa_params : Is a structure will hold the parameters

m : holds the modulus

e : holds the public exponent

d : holds the private exponent

p : holds the first prime (p)

q : holds the second prime (q)

u : holds the coefficient

Returns : GNUTLS_E_SUCCESS on success, or an negative error code.

gnutls_rsa_params_generate2 ()

```
int                 gnutls_rsa_params_generate2         (gnutls_rsa_params_t params,
                                                         unsigned int bits);
```

This function will generate new temporary RSA parameters for use in RSA-EXPORT ciphersuites. This function is normally slow.

Note that if the parameters are to be used in export cipher suites the bits value should be 512 or less. Also note that the generation of new RSA parameters is only useful to servers. Clients use the parameters sent by the server, thus it's no use calling this in client side.

params : The structure where the parameters will be stored

bits : is the prime's number of bits

Returns : GNUTLS_E_SUCCESS on success, or an negative error code.

gnutls_rsa_params_export_raw ()

```
int                 gnutls_rsa_params_export_raw        (gnutls_rsa_params_t params,
                                                         gnutls_datum_t *m,
                                                         gnutls_datum_t *e,
                                                         gnutls_datum_t *d,
                                                         gnutls_datum_t *p,
                                                         gnutls_datum_t *q,
                                                         gnutls_datum_t *u,
                                                         unsigned int *bits);
```

This function will export the RSA parameters found in the given structure. The new parameters will be allocated using gnutls_malloc() and will be stored in the appropriate datum.

params : a structure that holds the rsa parameters

m : will hold the modulus

e : will hold the public exponent

d : will hold the private exponent

p : will hold the first prime (p)

q : will hold the second prime (q)

u : will hold the coefficient

bits : if non null will hold the prime's number of bits

Returns : GNUTLS_E_SUCCESS on success, or an negative error code.

gnutls_rsa_params_export_pkcs1 ()

```
int                 gnutls_rsa_params_export_pkcs1      (gnutls_rsa_params_t params,
                                                         gnutls_x509_crt_fmt_t format,
                                                         unsigned char *params_data,
                                                         size_t *params_data_size);
```

This function will export the given RSA parameters to a PKCS1 RSAPublicKey structure. If the buffer provided is not long enough to hold the output, then GNUTLS_E_SHORT_MEMORY_BUFFER will be returned.

If the structure is PEM encoded, it will have a header of "BEGIN RSA PRIVATE KEY".

params : Holds the RSA parameters

format : the format of output params. One of PEM or DER.

params_data : will contain a PKCS1 RSAPublicKey structure PEM or DER encoded

params_data_size : holds the size of params_data (and will be replaced by the actual size of parameters)

Returns : GNUTLS_E_SUCCESS on success, or an negative error code.

gnutls_rsa_params_import_pkcs1 ()

```
int                 gnutls_rsa_params_import_pkcs1      (gnutls_rsa_params_t params,
                                                         const gnutls_datum_t *pkcs1_params ↩
                                                             ,
                                                         gnutls_x509_crt_fmt_t format);
```

This function will extract the RSAPublicKey found in a PKCS1 formatted structure.
If the structure is PEM encoded, it should have a header of "BEGIN RSA PRIVATE KEY".

params : A structure where the parameters will be copied to

pkcs1_params : should contain a PKCS1 RSAPublicKey structure PEM or DER encoded

format : the format of params. PEM or DER.

Returns : GNUTLS_E_SUCCESS on success, or an negative error code.

gnutls_pull_func ()

```
ssize_t             (*gnutls_pull_func)                 (gnutls_transport_ptr_t Param1,
                                                         void *Param2,
                                                         size_t Param3);
```

Param1 :

Param2 :

Param3 :

Returns :

gnutls_push_func ()

```
ssize_t             (*gnutls_push_func)                 (gnutls_transport_ptr_t Param1,
                                                         const void *Param2,
                                                         size_t Param3);
```

Param1 :

Param2 :

Param3 :

Returns :

gnutls_transport_set_ptr ()

```
void                gnutls_transport_set_ptr            (gnutls_session_t session,
                                                         gnutls_transport_ptr_t ptr);
```

Used to set the first argument of the transport function (like PUSH and PULL). In berkeley style sockets this function will set the connection handle.

session : is a gnutls_session_t structure.

ptr : is the value.

gnutls_transport_set_ptr2 ()

```
void                gnutls_transport_set_ptr2           (gnutls_session_t session,
                                                         gnutls_transport_ptr_t recv_ptr,
                                                         gnutls_transport_ptr_t send_ptr);
```

Used to set the first argument of the transport function (like PUSH and PULL). In berkeley style sockets this function will set the connection handle. With this function you can use two different pointers for receiving and sending.

session : is a gnutls_session_t structure.

recv_ptr : is the value for the pull function

send_ptr : is the value for the push function

gnutls_transport_get_ptr ()

```
gnutls_transport_ptr_t  gnutls_transport_get_ptr        (gnutls_session_t session);
```

Used to get the first argument of the transport function (like PUSH and PULL). This must have been set using gnutls_transport_set_ptr().

session : is a gnutls_session_t structure.

Returns : first argument of the transport function.

gnutls_transport_get_ptr2 ()

```
void                gnutls_transport_get_ptr2           (gnutls_session_t session,
                                                         gnutls_transport_ptr_t *recv_ptr,
                                                         gnutls_transport_ptr_t *send_ptr);
```

Used to get the arguments of the transport functions (like PUSH and PULL). These should have been set using gnutls_transport_set_ptr2()

session : is a gnutls_session_t structure.

recv_ptr : will hold the value for the pull function

send_ptr : will hold the value for the push function

gnutls_transport_set_lowat ()

```
void                gnutls_transport_set_lowat          (gnutls_session_t session,
                                                         int num);
```

Used to set the lowat value in order for select to check if there are pending data to socket buffer. Used only if you have changed the default low water value (default is 1). Normally you will not need that function. This function is only useful if using berkeley style sockets. Otherwise it must be called and set lowat to zero.

session : is a gnutls_session_t structure.

num : is the low water value.

gnutls_transport_set_push_function ()

```
void                gnutls_transport_set_push_function  (gnutls_session_t session,
                                                         gnutls_push_func push_func);
```

This is the function where you set a push function for gnutls to use in order to send data. If you are going to use berkeley style sockets, you do not need to use this function since the default (send(2)) will probably be ok. Otherwise you should specify this function for gnutls to be able to send data.

PUSH_FUNC is of the form, ssize_t (*gnutls_push_func)(gnutls_transport_ptr_t, const void*, size_t);

session : gnutls session

push_func : a callback function similar to write()

gnutls_transport_set_pull_function ()

```
void                gnutls_transport_set_pull_function  (gnutls_session_t session,
                                                         gnutls_pull_func pull_func);
```

This is the function where you set a function for gnutls to receive data. Normally, if you use berkeley style sockets, do not need to use this function since the default (recv(2)) will probably be ok.

PULL_FUNC is of the form, ssize_t (*gnutls_pull_func)(gnutls_transport_ptr_t, void*, size_t);

session : gnutls session

pull_func : a callback function similar to read()

gnutls_transport_set_errno ()

```
void                gnutls_transport_set_errno          (gnutls_session_t session,
                                                         int err);
```

Store *err* in the session-specific errno variable. Useful values for *err* is EAGAIN and EINTR, other values are treated will be treated as real errors in the push/pull function.

This function is useful in replacement push/pull functions set by gnutls_transport_set_push_function and gnutls_transport_set_pullpush_ under Windows, where the replacement push/pull may not have access to the same *errno* variable that is used by GnuTLS (e.g., the application is linked to msvcr71.dll and gnutls is linked to msvcrt.dll).

If you don't have the *session* variable easily accessible from the push/pull function, and don't worry about thread conflicts, you can also use gnutls_transport_set_global_errno().

session : is a gnutls_session_t structure.

err : error value to store in session-specific errno variable.

gnutls_transport_set_global_errno ()

```
void                gnutls_transport_set_global_errno   (int err);
```

Store *err* in the global errno variable. Useful values for *err* is EAGAIN and EINTR, other values are treated will be treated as real errors in the push/pull function.

This function is useful in replacement push/pull functions set by gnutls_transport_set_push_function and gnutls_transport_set_pullpush_ under Windows, where the replacement push/pull may not have access to the same *errno* variable that is used by GnuTLS (e.g., the application is linked to msvcr71.dll and gnutls is linked to msvcrt.dll).

Whether this function is thread safe or not depends on whether the global variable errno is thread safe, some system libraries make it a thread-local variable. When feasible, using the guaranteed thread-safe gnutls_transport_set_errno() may be better.

err : error value to store in global errno variable.

gnutls_session_set_ptr ()

```
void                gnutls_session_set_ptr              (gnutls_session_t session,
                                                         void *ptr);
```

This function will set (associate) the user given pointer *ptr* to the session structure. This is pointer can be accessed with gnutls_session_get_ptr().

session : is a gnutls_session_t structure.

ptr : is the user pointer

gnutls_session_get_ptr ()

```
void *              gnutls_session_get_ptr              (gnutls_session_t session);
```

Get user pointer for session. Useful in callbacks. This is the pointer set with gnutls_session_set_ptr().

session : is a gnutls_session_t structure.

Returns : the user given pointer from the session structure, or NULL if it was never set.

gnutls_openpgp_send_cert ()

```
void                gnutls_openpgp_send_cert            (gnutls_session_t session,
                                                         gnutls_openpgp_crt_status_t status ↩
                                                             );
```

This function will order gnutls to send the key fingerprint instead of the key in the initial handshake procedure. This should be used with care and only when there is indication or knowledge that the server can obtain the client's key.

session : is a pointer to a gnutls_session_t structure.

status : is one of GNUTLS_OPENPGP_CERT, or GNUTLS_OPENPGP_CERT_FINGERPRINT

gnutls_fingerprint ()

```
int                 gnutls_fingerprint                  (gnutls_digest_algorithm_t algo,
                                                         const gnutls_datum_t *data,
                                                         void *result,
                                                         size_t *result_size);
```

This function will calculate a fingerprint (actually a hash), of the given data. The result is not printable data. You should convert it to hex, or to something else printable.

This is the usual way to calculate a fingerprint of an X.509 DER encoded certificate. Note however that the fingerprint of an OpenPGP is not just a hash and cannot be calculated with this function.

algo : is a digest algorithm

data : is the data

result : is the place where the result will be copied (may be null).

result_size : should hold the size of the result. The actual size of the returned result will also be copied there.

Returns : On success, GNUTLS_E_SUCCESS (0) is returned, otherwise an error code is returned.

gnutls_srp_free_client_credentials ()

```
void                gnutls_srp_free_client_credentials  (gnutls_srp_client_credentials_t sc ↩
    );
```

This structure is complex enough to manipulate directly thus this helper function is provided in order to free (deallocate) it.

sc : is a gnutls_srp_client_credentials_t structure.

gnutls_srp_allocate_client_credentials ()

```
int                 gnutls_srp_allocate_client_credentials
                                                        (gnutls_srp_client_credentials_t * ↩
                                                            sc);
```

This structure is complex enough to manipulate directly thus this helper function is provided in order to allocate it.

sc : is a pointer to a gnutls_srp_server_credentials_t structure.

Returns : On success, GNUTLS_E_SUCCESS (0) is returned, or an error code.

gnutls_srp_set_client_credentials ()

```
int                 gnutls_srp_set_client_credentials   (gnutls_srp_client_credentials_t  ↩
    res,
                                                         const char *username,
                                                         const char *password);
```

This function sets the username and password, in a gnutls_srp_client_credentials_t structure. Those will be used in SRP authentication. *username* and *password* should be ASCII strings or UTF-8 strings prepared using the "SASLprep" profile of "stringprep".

res : is a gnutls_srp_client_credentials_t structure.

username : is the user's userid

password : is the user's password

Returns : On success, GNUTLS_E_SUCCESS (0) is returned, or an error code.

gnutls_srp_free_server_credentials ()

```
void                gnutls_srp_free_server_credentials  (gnutls_srp_server_credentials_t sc
    );
```

This structure is complex enough to manipulate directly thus this helper function is provided in order to free (deallocate) it.

sc : is a gnutls_srp_server_credentials_t structure.

gnutls_srp_allocate_server_credentials ()

```
int                 gnutls_srp_allocate_server_credentials
                                                        (gnutls_srp_server_credentials_t * sc);
```

This structure is complex enough to manipulate directly thus this helper function is provided in order to allocate it.

sc : is a pointer to a gnutls_srp_server_credentials_t structure.

Returns : On success, GNUTLS_E_SUCCESS (0) is returned, or an error code.

gnutls_srp_set_server_credentials_file ()

```
int                 gnutls_srp_set_server_credentials_file
                                                        (gnutls_srp_server_credentials_t
                                                            res,
                                                         const char *password_file,
                                                         const char *password_conf_file);
```

This function sets the password files, in a gnutls_srp_server_credentials_t structure. Those password files hold usernames and verifiers and will be used for SRP authentication.

res : is a gnutls_srp_server_credentials_t structure.

password_file : is the SRP password file (tpasswd)

password_conf_file : is the SRP password conf file (tpasswd.conf)

Returns : On success, GNUTLS_E_SUCCESS (0) is returned, or an error code.

gnutls_srp_server_get_username ()

```
const char *        gnutls_srp_server_get_username      (gnutls_session_t session);
```

This function will return the username of the peer. This should only be called in case of SRP authentication and in case of a server. Returns NULL in case of an error.

session : is a gnutls session

Returns : SRP username of the peer, or NULL in case of error.

gnutls_srp_set_prime_bits ()

```
void                gnutls_srp_set_prime_bits           (gnutls_session_t session,
                                                         unsigned int bits);
```

This function sets the minimum accepted number of bits, for use in an SRP key exchange. If zero, the default 2048 bits will be used.

In the client side it sets the minimum accepted number of bits. If a server sends a prime with less bits than that GNUTLS_E_RECEIVED_ will be returned by the handshake.

This function has no effect in server side.

session : is a gnutls_session_t structure.

bits : is the number of bits

Since 2.6.0

gnutls_srp_verifier ()

```
int                 gnutls_srp_verifier                 (const char *username,
                                                         const char *password,
                                                         const gnutls_datum_t *salt,
                                                         const gnutls_datum_t *generator,
                                                         const gnutls_datum_t *prime,
                                                         gnutls_datum_t *res);
```

This function will create an SRP verifier, as specified in RFC2945. The *prime* and *generator* should be one of the static parameters defined in gnutls/extra.h or may be generated using the libgcrypt functions gcry_prime_generate() and gcry_prime_group_generator

The verifier will be allocated with *malloc* and will be stored in *res* using binary format.

username : is the user's name

password : is the user's password

salt : should be some randomly generated bytes

generator : is the generator of the group

prime : is the group's prime

res : where the verifier will be stored.

Returns : On success, GNUTLS_E_SUCCESS (0) is returned, or an error code.

gnutls_srp_2048_group_prime

```
extern const gnutls_datum_t gnutls_srp_2048_group_prime;
```

gnutls_srp_2048_group_generator

```
extern const gnutls_datum_t gnutls_srp_2048_group_generator;
```

gnutls_srp_1536_group_prime

```
extern const gnutls_datum_t gnutls_srp_1536_group_prime;
```

gnutls_srp_1536_group_generator

```
extern const gnutls_datum_t gnutls_srp_1536_group_generator;
```

gnutls_srp_1024_group_prime

```
extern const gnutls_datum_t gnutls_srp_1024_group_prime;
```

gnutls_srp_1024_group_generator

```
extern const gnutls_datum_t gnutls_srp_1024_group_generator;
```

gnutls_srp_set_server_credentials_function ()

```
void                gnutls_srp_set_server_credentials_function
                                                    (gnutls_srp_server_credentials_t  ↩
                                                        cred,
                                                     gnutls_srp_server_credentials_function ↩
                                                         *func);
```

This function can be used to set a callback to retrieve the user's SRP credentials. The callback's function form is:

int (*callback)(gnutls_session_t, const char* username, gnutls_datum_t* salt, gnutls_datum_t *verifier, gnutls_datum_t* g, gnutls_datum_t* n);

username contains the actual username. The *salt*, *verifier*, *generator* and *prime* must be filled in using the gnutls_malloc(). For convenience *prime* and *generator* may also be one of the static parameters defined in extra.h.

In case the callback returned a negative number then gnutls will assume that the username does not exist.

In order to prevent attackers from guessing valid usernames, if a user does not exist, g and n values should be filled in using a random user's parameters. In that case the callback must return the special value (1).

The callback function will only be called once per handshake. The callback function should return 0 on success, while -1 indicates an error.

***cred*:** is a gnutls_srp_server_credentials_t structure.

***func*:** is the callback function

gnutls_srp_set_client_credentials_function ()

```
void                gnutls_srp_set_client_credentials_function
                                                    (gnutls_srp_client_credentials_t  ↩
                                                        cred,
                                                     gnutls_srp_client_credentials_function ↩
                                                         *func);
```

This function can be used to set a callback to retrieve the username and password for client SRP authentication. The callback's function form is:

int (*callback)(gnutls_session_t, char** username, char**password);

The *username* and *password* must be allocated using gnutls_malloc(). *username* and *password* should be ASCII strings or UTF-8 strings prepared using the "SASLprep" profile of "stringprep".

The callback function will be called once per handshake before the initial hello message is sent.

The callback should not return a negative error code the second time called, since the handshake procedure will be aborted.

The callback function should return 0 on success. -1 indicates an error.

cred : is a gnutls_srp_server_credentials_t structure.

func : is the callback function

gnutls_srp_base64_encode ()

```
int                 gnutls_srp_base64_encode            (const gnutls_datum_t *data,
                                                         char *result,
                                                         size_t *result_size);
```

This function will convert the given data to printable data, using the base64 encoding, as used in the libsrp. This is the encoding used in SRP password files. If the provided buffer is not long enough GNUTLS_E_SHORT_MEMORY_BUFFER is returned.

Warning! This base64 encoding is not the "standard" encoding, so do not use it for non-SRP purposes.

data : contain the raw data

result : the place where base64 data will be copied

result_size : holds the size of the result

Returns : GNUTLS_E_SHORT_MEMORY_BUFFER if the buffer given is not long enough, or 0 on success.

gnutls_srp_base64_encode_alloc ()

```
int                 gnutls_srp_base64_encode_alloc      (const gnutls_datum_t *data,
                                                         gnutls_datum_t *result);
```

This function will convert the given data to printable data, using the base64 encoding. This is the encoding used in SRP password files. This function will allocate the required memory to hold the encoded data.

You should use gnutls_free() to free the returned data.

Warning! This base64 encoding is not the "standard" encoding, so do not use it for non-SRP purposes.

data : contains the raw data

result : will hold the newly allocated encoded data

Returns : 0 on success, or an error code.

gnutls_srp_base64_decode ()

```
int                 gnutls_srp_base64_decode            (const gnutls_datum_t *b64_data,
                                                         char *result,
                                                         size_t *result_size);
```

This function will decode the given encoded data, using the base64 encoding found in libsrp.

Note that *b64_data* should be null terminated.

Warning! This base64 encoding is not the "standard" encoding, so do not use it for non-SRP purposes.

b64_data : contain the encoded data

result : the place where decoded data will be copied

result_size : holds the size of the result

Returns : GNUTLS_E_SHORT_MEMORY_BUFFER if the buffer given is not long enough, or 0 on success.

gnutls_srp_base64_decode_alloc ()

```
int                 gnutls_srp_base64_decode_alloc      (const gnutls_datum_t *b64_data,
                                                         gnutls_datum_t *result);
```

This function will decode the given encoded data. The decoded data will be allocated, and stored into result. It will decode using the base64 algorithm as used in libsrp.

You should use gnutls_free() to free the returned data.

Warning! This base64 encoding is not the "standard" encoding, so do not use it for non-SRP purposes.

b64_data : contains the encoded data

result : the place where decoded data lie

Returns : 0 on success, or an error code.

enum gnutls_psk_key_flags

```
typedef enum gnutls_psk_key_flags
{
  GNUTLS_PSK_KEY_RAW = 0,
  GNUTLS_PSK_KEY_HEX
} gnutls_psk_key_flags;
```

Enumeration of different PSK key flags.

GNUTLS_PSK_KEY_RAW PSK-key in raw format.

GNUTLS_PSK_KEY_HEX PSK-key in hex format.

gnutls_psk_free_client_credentials ()

```
void                gnutls_psk_free_client_credentials  (gnutls_psk_client_credentials_t sc ↩
    );
```

This structure is complex enough to manipulate directly thus this helper function is provided in order to free (deallocate) it.

sc : is a gnutls_psk_client_credentials_t structure.

gnutls_psk_allocate_client_credentials ()

```
int                 gnutls_psk_allocate_client_credentials
                                                    (gnutls_psk_client_credentials_t * ↩
                                                        sc);
```

This structure is complex enough to manipulate directly thus this helper function is provided in order to allocate it.

sc : is a pointer to a gnutls_psk_server_credentials_t structure.

Returns : GNUTLS_E_SUCCESS on success, or an error code.

gnutls_psk_set_client_credentials ()

```
int                 gnutls_psk_set_client_credentials   (gnutls_psk_client_credentials_t  ↩
    res,
                                                         const char *username,
                                                         const gnutls_datum_t *key,
                                                         gnutls_psk_key_flags format);
```

This function sets the username and password, in a gnutls_psk_client_credentials_t structure. Those will be used in PSK authentication. *username* should be an ASCII string or UTF-8 strings prepared using the "SASLprep" profile of "stringprep". The key can be either in raw byte format or in Hex format (without the 0x prefix).

res : is a gnutls_psk_client_credentials_t structure.

username : is the user's zero-terminated userid

key : is the user's key

format : indicate the format of the key, either GNUTLS_PSK_KEY_RAW or GNUTLS_PSK_KEY_HEX.

Returns : GNUTLS_E_SUCCESS on success, or an error code.

gnutls_psk_free_server_credentials ()

```
void                gnutls_psk_free_server_credentials  (gnutls_psk_server_credentials_t sc ↩
    );
```

This structure is complex enough to manipulate directly thus this helper function is provided in order to free (deallocate) it.

sc : is a gnutls_psk_server_credentials_t structure.

gnutls_psk_allocate_server_credentials ()

```
int                 gnutls_psk_allocate_server_credentials
                                                    (gnutls_psk_server_credentials_t * ↩
                                                        sc);
```

This structure is complex enough to manipulate directly thus this helper function is provided in order to allocate it.

sc : is a pointer to a gnutls_psk_server_credentials_t structure.

Returns : GNUTLS_E_SUCCESS on success, or an error code.

gnutls_psk_set_server_credentials_file ()

```
int                 gnutls_psk_set_server_credentials_file
                                                    (gnutls_psk_server_credentials_t  ↩
                                                       res,
                                                     const char *password_file);
```

This function sets the password file, in a gnutls_psk_server_credentials_t structure. This password file holds usernames and keys and will be used for PSK authentication.

res : is a gnutls_psk_server_credentials_t structure.

password_file : is the PSK password file (passwd.psk)

Returns : GNUTLS_E_SUCCESS on success, or an error code.

gnutls_psk_set_server_credentials_hint ()

```
int                 gnutls_psk_set_server_credentials_hint
                                                    (gnutls_psk_server_credentials_t  ↩
                                                       res,
                                                     const char *hint);
```

This function sets the identity hint, in a gnutls_psk_server_credentials_t structure. This hint is sent to the client to help it chose a good PSK credential (i.e., username and password).

res : is a gnutls_psk_server_credentials_t structure.

hint : is the PSK identity hint string

Returns : GNUTLS_E_SUCCESS on success, or an error code.

Since 2.4.0

gnutls_psk_server_get_username ()

```
const char *        gnutls_psk_server_get_username      (gnutls_session_t session);
```

This should only be called in case of PSK authentication and in case of a server.

session : is a gnutls session

Returns : the username of the peer, or NULL in case of an error.

gnutls_psk_client_get_hint ()

```
const char *        gnutls_psk_client_get_hint          (gnutls_session_t session);
```

The PSK identity hint may give the client help in deciding which username to use. This should only be called in case of PSK authentication and in case of a client.

session : is a gnutls session

Returns : the identity hint of the peer, or NULL in case of an error.

Since 2.4.0

gnutls_psk_set_server_credentials_function ()

```
void                gnutls_psk_set_server_credentials_function
                                                    (gnutls_psk_server_credentials_t  ↩
                                                        cred,
                                                     gnutls_psk_server_credentials_function ↩
                                                         *func);
```

This function can be used to set a callback to retrieve the user's PSK credentials. The callback's function form is: int (*callback)(gnutls_session_t, const char* username, gnutls_datum_t* key);

username contains the actual username. The *key* must be filled in using the gnutls_malloc().

In case the callback returned a negative number then gnutls will assume that the username does not exist.

The callback function will only be called once per handshake. The callback function should return 0 on success, while -1 indicates an error.

cred **:** is a gnutls_psk_server_credentials_t structure.

func **:** is the callback function

gnutls_psk_set_client_credentials_function ()

```
void                gnutls_psk_set_client_credentials_function
                                                    (gnutls_psk_client_credentials_t  ↩
                                                        cred,
                                                     gnutls_psk_client_credentials_function ↩
                                                         *func);
```

This function can be used to set a callback to retrieve the username and password for client PSK authentication. The callback's function form is: int (*callback)(gnutls_session_t, char** username, gnutls_datum_t* key);

The *username* and *key->data* must be allocated using gnutls_malloc(). *username* should be ASCII strings or UTF-8 strings prepared using the "SASLprep" profile of "stringprep".

The callback function will be called once per handshake.

The callback function should return 0 on success. -1 indicates an error.

cred **:** is a gnutls_psk_server_credentials_t structure.

func **:** is the callback function

gnutls_hex_encode ()

```
int                 gnutls_hex_encode               (const gnutls_datum_t *data,
                                                     char *result,
                                                     size_t *result_size);
```

This function will convert the given data to printable data, using the hex encoding, as used in the PSK password files.

data **:** contain the raw data

result **:** the place where hex data will be copied

result_size **:** holds the size of the result

Returns **:** GNUTLS_E_SHORT_MEMORY_BUFFER if the buffer given is not long enough, or 0 on success.

gnutls_hex_decode ()

```
int                 gnutls_hex_decode                   (const gnutls_datum_t *hex_data,
                                                         char *result,
                                                         size_t *result_size);
```

This function will decode the given encoded data, using the hex encoding used by PSK password files.

Note that hex_data should be null terminated.

hex_data : contain the encoded data

result : the place where decoded data will be copied

result_size : holds the size of the result

Returns : GNUTLS_E_SHORT_MEMORY_BUFFER if the buffer given is not long enough, or 0 on success.

gnutls_psk_set_server_dh_params ()

```
void                gnutls_psk_set_server_dh_params     (gnutls_psk_server_credentials_t ↩
    res,
                                                         gnutls_dh_params_t dh_params);
```

This function will set the Diffie-Hellman parameters for an anonymous server to use. These parameters will be used in Diffie-Hellman exchange with PSK cipher suites.

res : is a gnutls_psk_server_credentials_t structure

dh_params : is a structure that holds Diffie-Hellman parameters.

gnutls_psk_set_server_params_function ()

```
void                gnutls_psk_set_server_params_function
                                                        (gnutls_psk_server_credentials_t ↩
                                                           res,
                                                         gnutls_params_function *func);
```

This function will set a callback in order for the server to get the Diffie-Hellman parameters for PSK authentication. The callback should return zero on success.

res : is a gnutls_certificate_credentials_t structure

func : is the function to be called

gnutls_psk_netconf_derive_key ()

```
int                 gnutls_psk_netconf_derive_key       (const char *password,
                                                         const char *psk_identity,
                                                         const char *psk_identity_hint,
                                                         gnutls_datum_t *output_key);
```

This function will derive a PSK key from a password, for use with the Netconf protocol.

password : zero terminated string containing password.

psk_identity : zero terminated string with PSK identity.

psk_identity_hint **:** zero terminated string with PSK identity hint.

output_key **:** output variable, contains newly allocated *data pointer.

Returns **:** GNUTLS_E_SUCCESS on success, or an error code.

Since 2.4.0

enum gnutls_x509_subject_alt_name_t

```
typedef enum gnutls_x509_subject_alt_name_t
{
  GNUTLS_SAN_DNSNAME = 1,
  GNUTLS_SAN_RFC822NAME = 2,
  GNUTLS_SAN_URI = 3,
  GNUTLS_SAN_IPADDRESS = 4,
  GNUTLS_SAN_OTHERNAME = 5,
  GNUTLS_SAN_DN = 6,
  /* The following are "virtual" subject alternative name types, in
     that they are represented by an otherName value and an OID.
     Used by gnutls_x509_crt_get_subject_alt_othername_oid().  */
  GNUTLS_SAN_OTHERNAME_XMPP = 1000
} gnutls_x509_subject_alt_name_t;
```

Enumeration of different subject alternative names types.

GNUTLS_SAN_DNSNAME DNS-name SAN.

GNUTLS_SAN_RFC822NAME E-mail address SAN.

GNUTLS_SAN_URI URI SAN.

GNUTLS_SAN_IPADDRESS IP address SAN.

GNUTLS_SAN_OTHERNAME OtherName SAN.

GNUTLS_SAN_DN DN SAN.

GNUTLS_SAN_OTHERNAME_XMPP Virtual SAN, used by gnutls_x509_crt_get_subject_alt_othername_oid().

struct gnutls_openpgp_crt_int

```
struct gnutls_openpgp_crt_int;
```

gnutls_openpgp_crt_t

```
typedef struct gnutls_openpgp_crt_int *gnutls_openpgp_crt_t;
```

struct gnutls_openpgp_privkey_int

```
struct gnutls_openpgp_privkey_int;
```

gnutls_openpgp_privkey_t

```
typedef struct gnutls_openpgp_privkey_int *gnutls_openpgp_privkey_t;
```

gnutls_auth_get_type ()

```
gnutls_credentials_type_t  gnutls_auth_get_type         (gnutls_session_t session);
```

Returns type of credentials for the current authentication schema. The returned information is to be used to distinguish the function used to access authentication data.

Eg. for CERTIFICATE ciphersuites (key exchange algorithms: GNUTLS_KX_RSA, GNUTLS_KX_DHE_RSA), the same function are to be used to access the authentication data.

session : is a gnutls_session_t structure.

Returns : The type of credentials for the current authentication schema, a gnutls_credentials_type_t type.

gnutls_auth_server_get_type ()

```
gnutls_credentials_type_t  gnutls_auth_server_get_type  (gnutls_session_t session);
```

Returns the type of credentials that were used for server authentication. The returned information is to be used to distinguish the function used to access authentication data.

session : is a gnutls_session_t structure.

Returns : The type of credentials for the server authentication schema, a gnutls_credentials_type_t type.

gnutls_auth_client_get_type ()

```
gnutls_credentials_type_t  gnutls_auth_client_get_type  (gnutls_session_t session);
```

Returns the type of credentials that were used for client authentication. The returned information is to be used to distinguish the function used to access authentication data.

session : is a gnutls_session_t structure.

Returns : The type of credentials for the client authentication schema, a gnutls_credentials_type_t type.

gnutls_dh_set_prime_bits ()

```
void                gnutls_dh_set_prime_bits            (gnutls_session_t session,
                                                         unsigned int bits);
```

This function sets the number of bits, for use in an Diffie-Hellman key exchange. This is used both in DH ephemeral and DH anonymous cipher suites. This will set the minimum size of the prime that will be used for the handshake.

In the client side it sets the minimum accepted number of bits. If a server sends a prime with less bits than that GNUTLS_E_DH_PRIME_
will be returned by the handshake.

This function has no effect in server side.

session : is a gnutls_session_t structure.

bits : is the number of bits

gnutls_dh_get_secret_bits ()

```
int                 gnutls_dh_get_secret_bits           (gnutls_session_t session);
```

This function will return the bits used in the last Diffie-Hellman key exchange with the peer. Should be used for both anonymous and ephemeral Diffie-Hellman.

session : is a gnutls session

Returns : On success, GNUTLS_E_SUCCESS (0) is returned, otherwise an error code is returned.

gnutls_dh_get_peers_public_bits ()

```
int                 gnutls_dh_get_peers_public_bits     (gnutls_session_t session);
```

Get the Diffie-Hellman public key bit size. Can be used for both anonymous and ephemeral Diffie-Hellman.

session : is a gnutls session

Returns : the public key bit size used in the last Diffie-Hellman key exchange with the peer, or a negative value in case of error.

gnutls_dh_get_prime_bits ()

```
int                 gnutls_dh_get_prime_bits            (gnutls_session_t session);
```

This function will return the bits of the prime used in the last Diffie-Hellman key exchange with the peer. Should be used for both anonymous and ephemeral Diffie-Hellman. Note that some ciphers, like RSA and DSA without DHE, does not use a Diffie-Hellman key exchange, and then this function will return 0.

session : is a gnutls session

Returns : The Diffie-Hellman bit strength is returned, or 0 if no Diffie-Hellman key exchange was done, or a negative error code on failure.

gnutls_dh_get_group ()

```
int                 gnutls_dh_get_group                 (gnutls_session_t session,
                                                         gnutls_datum_t *raw_gen,
                                                         gnutls_datum_t *raw_prime);
```

This function will return the group parameters used in the last Diffie-Hellman key exchange with the peer. These are the prime and the generator used. This function should be used for both anonymous and ephemeral Diffie-Hellman. The output parameters must be freed with gnutls_free().

session : is a gnutls session

raw_gen : will hold the generator.

raw_prime : will hold the prime.

Returns : On success, GNUTLS_E_SUCCESS (0) is returned, otherwise an error code is returned.

gnutls_dh_get_pubkey ()

```
int                 gnutls_dh_get_pubkey                (gnutls_session_t session,
                                                         gnutls_datum_t *raw_key);
```

This function will return the peer's public key used in the last Diffie-Hellman key exchange. This function should be used for both anonymous and ephemeral Diffie-Hellman. The output parameters must be freed with gnutls_free().

session : is a gnutls session

raw_key : will hold the public key.

Returns : On success, GNUTLS_E_SUCCESS (0) is returned, otherwise an error code is returned.

gnutls_rsa_export_get_pubkey ()

```
int                 gnutls_rsa_export_get_pubkey        (gnutls_session_t session,
                                                         gnutls_datum_t *exponent,
                                                         gnutls_datum_t *modulus);
```

This function will return the peer's public key exponent and modulus used in the last RSA-EXPORT authentication. The output parameters must be freed with gnutls_free().

session : is a gnutls session

exponent : will hold the exponent.

modulus : will hold the modulus.

Returns : On success, GNUTLS_E_SUCCESS (0) is returned, otherwise an error code is returned.

gnutls_rsa_export_get_modulus_bits ()

```
int                 gnutls_rsa_export_get_modulus_bits  (gnutls_session_t session);
```

Get the export RSA parameter's modulus size.

session : is a gnutls session

Returns : the bits used in the last RSA-EXPORT key exchange with the peer, or a negative value in case of error.

gnutls_sign_func ()

```
int                 (*gnutls_sign_func)                 (gnutls_session_t session,
                                                         void *userdata,
                                                         gnutls_certificate_type_t ↩
                                                             cert_type,
                                                         const gnutls_datum_t *cert,
                                                         const gnutls_datum_t *hash,
                                                         gnutls_datum_t *signature);
```

session :

userdata :

cert_type :

cert :

hash :

signature :

Returns :

gnutls_sign_callback_set ()

```
void                gnutls_sign_callback_set            (gnutls_session_t session,
                                                         gnutls_sign_func sign_func,
                                                         void *userdata);
```

Set the callback function. The function must have this prototype:

typedef int (*gnutls_sign_func) (gnutls_session_t session, void *userdata, gnutls_certificate_type_t cert_type, const gnutls_datum_t * cert, const gnutls_datum_t * hash, gnutls_datum_t * signature);

The *userdata* parameter is passed to the *sign_func* verbatim, and can be used to store application-specific data needed in the callback function. See also gnutls_sign_callback_get().

session : is a gnutls session

sign_func : function pointer to application's sign callback.

userdata : void pointer that will be passed to sign callback.

gnutls_sign_callback_get ()

```
gnutls_sign_func    gnutls_sign_callback_get            (gnutls_session_t session,
                                                         void **userdata);
```

Retrieve the callback function, and its userdata pointer.

session : is a gnutls session

userdata : if non-NULL, will be set to abstract callback pointer.

Returns : The function pointer set by gnutls_sign_callback_set(), or if not set, NULL.

gnutls_certificate_client_set_retrieve_function ()

```
void                gnutls_certificate_client_set_retrieve_function
                                                        (gnutls_certificate_credentials_t ↩
                                                            cred,
                                                         gnutls_certificate_client_retrieve_function
                                                             *func);
```

This function sets a callback to be called in order to retrieve the certificate to be used in the handshake.

The callback's function prototype is: int (*callback)(gnutls_session_t, const gnutls_datum_t* req_ca_dn, int nreqs, const gnutls_pk_algo pk_algos, int pk_algos_length, gnutls_retr_st* st);

req_ca_cert is only used in X.509 certificates. Contains a list with the CA names that the server considers trusted. Normally we should send a certificate that is signed by one of these CAs. These names are DER encoded. To get a more meaningful value use the function gnutls_x509_rdn_get().

pk_algos contains a list with server's acceptable signature algorithms. The certificate returned should support the server's given algorithms.

st should contain the certificates and private keys.

If the callback function is provided then gnutls will call it, in the handshake, after the certificate request message has been received.

The callback function should set the certificate list to be sent, and return 0 on success. If no certificate was selected then the number of certificates should be set to zero. The value (-1) indicates error and the handshake will be terminated.

cred : is a gnutls_certificate_credentials_t structure.

func : is the callback function

gnutls_certificate_server_set_retrieve_function ()

```
void                gnutls_certificate_server_set_retrieve_function
                                                    (gnutls_certificate_credentials_t ↩
                                                        cred,
                                                     gnutls_certificate_server_retrieve_function
                                                         *func);
```

This function sets a callback to be called in order to retrieve the certificate to be used in the handshake.

The callback's function prototype is: int (*callback)(gnutls_session_t, gnutls_retr_st* st);

st should contain the certificates and private keys.

If the callback function is provided then gnutls will call it, in the handshake, after the certificate request message has been received.

The callback function should set the certificate list to be sent, and return 0 on success. The value (-1) indicates error and the handshake will be terminated.

***cred*:** is a gnutls_certificate_credentials_t structure.

***func*:** is the callback function

gnutls_certificate_set_verify_function ()

```
void                gnutls_certificate_set_verify_function
                                                    (gnutls_certificate_credentials_t ↩
                                                        cred,
                                                     gnutls_certificate_verify_function ↩
                                                         *func);
```

This function sets a callback to be called when peer's certificate has been received in order to verify it on receipt rather than doing after the handshake is completed.

The callback's function prototype is: int (*callback)(gnutls_session_t);

If the callback function is provided then gnutls will call it, in the handshake, just after the certificate message has been received. To verify or obtain the certificate the gnutls_certificate_verify_peers2(), gnutls_certificate_type_get(), gnutls_certificate_get_peers() functions can be used.

The callback function should return 0 for the handshake to continue or non-zero to terminate.

***cred*:** is a gnutls_certificate_credentials_t structure.

***func*:** is the callback function

Since 2.10.0

gnutls_certificate_server_set_request ()

```
void                gnutls_certificate_server_set_request
                                                    (gnutls_session_t session,
                                                     gnutls_certificate_request_t req);
```

This function specifies if we (in case of a server) are going to send a certificate request message to the client. If *req* is GNUTLS_CERT_REQUIRE then the server will return an error if the peer does not provide a certificate. If you do not call this function then the client will not be asked to send a certificate.

***session*:** is a gnutls_session_t structure.

***req*:** is one of GNUTLS_CERT_REQUEST, GNUTLS_CERT_REQUIRE

gnutls_certificate_get_peers ()

```
const gnutls_datum_t * gnutls_certificate_get_peers     (gnutls_session_t session,
                                                         unsigned int *list_size);
```

Get the peer's raw certificate (chain) as sent by the peer. These certificates are in raw format (DER encoded for X.509). In case of a X.509 then a certificate list may be present. The first certificate in the list is the peer's certificate, following the issuer's certificate, then the issuer's issuer etc.

In case of OpenPGP keys a single key will be returned in raw format.

session : is a gnutls session

list_size : is the length of the certificate list

Returns : return a pointer to a gnutls_datum_t containing our certificates, or NULL in case of an error or if no certificate was used.

gnutls_certificate_get_ours ()

```
const gnutls_datum_t * gnutls_certificate_get_ours      (gnutls_session_t session);
```

Get the certificate as sent to the peer, in the last handshake. These certificates are in raw format. In X.509 this is a certificate list. In OpenPGP this is a single certificate.

session : is a gnutls session

Returns : return a pointer to a gnutls_datum_t containing our certificates, or NULL in case of an error or if no certificate was used.

gnutls_certificate_activation_time_peers ()

```
time_t              gnutls_certificate_activation_time_peers
                                                    (gnutls_session_t session);
```

Warning

`gnutls_certificate_activation_time_peers` is deprecated and should not be used in newly-written code. gnutls_certificate_verify_peers2() now verifies activation times.

This function will return the peer's certificate activation time. This is the creation time for openpgp keys.

session : is a gnutls session

Returns : (time_t)-1 on error.

gnutls_certificate_expiration_time_peers ()

```
time_t              gnutls_certificate_expiration_time_peers
                                                    (gnutls_session_t session);
```

Warning
`gnutls_certificate_expiration_time_peers` is deprecated and should not be used in newly-written code. gnutls_certificate_verify_peers2() now verifies expiration times.

This function will return the peer's certificate expiration time.

session **:** is a gnutls session

Returns **:** (time_t)-1 on error.

gnutls_certificate_client_get_request_status ()

```
int                 gnutls_certificate_client_get_request_status
                                                    (gnutls_session_t session);
```

Get whether client certificate is requested or not.

session **:** is a gnutls session

Returns **:** 0 if the peer (server) did not request client authentication or 1 otherwise, or a negative value in case of error.

gnutls_certificate_verify_peers2 ()

```
int                 gnutls_certificate_verify_peers2    (gnutls_session_t session,
                                                         unsigned int *status);
```

This function will try to verify the peer's certificate and return its status (trusted, invalid etc.). The value of *status* should be one or more of the gnutls_certificate_status_t enumerated elements bitwise or'd. To avoid denial of service attacks some default upper limits regarding the certificate key size and chain size are set. To override them use gnutls_certificate_set_verify_limits().

Note that you must also check the peer's name in order to check if the verified certificate belongs to the actual peer.

This function uses gnutls_x509_crt_list_verify() with the CAs in the credentials as trusted CAs.

Note that some commonly used X.509 Certificate Authorities are still using Version 1 certificates. If you want to accept them, you need to call gnutls_certificate_set_verify_flags() with, e.g., GNUTLS_VERIFY_ALLOW_X509_V1_CA_CRT parameter.

session **:** is a gnutls session

status **:** is the output of the verification

Returns **:** a negative error code on error and zero on success.

gnutls_certificate_verify_peers ()

```
int                 gnutls_certificate_verify_peers     (gnutls_session_t session);
```

Warning
`gnutls_certificate_verify_peers` is deprecated and should not be used in newly-written code. Use gnutls_certificate_verify_peers2() instead.

This function will try to verify the peer's certificate and return its status (trusted, invalid etc.). However you must also check the peer's name in order to check if the verified certificate belongs to the actual peer.

This function uses gnutls_x509_crt_list_verify().

session : is a gnutls session

Returns : one or more of the gnutls_certificate_status_t enumerated elements bitwise or'd, or a negative value on error.

gnutls_pem_base64_encode ()

```
int                 gnutls_pem_base64_encode            (const char *msg,
                                                         const gnutls_datum_t *data,
                                                         char *result,
                                                         size_t *result_size);
```

This function will convert the given data to printable data, using the base64 encoding. This is the encoding used in PEM messages.

The output string will be null terminated, although the size will not include the terminating null.

msg : is a message to be put in the header

data : contain the raw data

result : the place where base64 data will be copied

result_size : holds the size of the result

Returns : On success GNUTLS_E_SUCCESS (0) is returned, GNUTLS_E_SHORT_MEMORY_BUFFER is returned if the buffer given is not long enough, or 0 on success.

gnutls_pem_base64_decode ()

```
int                 gnutls_pem_base64_decode            (const char *header,
                                                         const gnutls_datum_t *b64_data,
                                                         unsigned char *result,
                                                         size_t *result_size);
```

This function will decode the given encoded data. If the header given is non null this function will search for "-----BEGIN header" and decode only this part. Otherwise it will decode the first PEM packet found.

header : A null terminated string with the PEM header (eg. CERTIFICATE)

b64_data : contain the encoded data

result : the place where decoded data will be copied

result_size : holds the size of the result

Returns : On success GNUTLS_E_SUCCESS (0) is returned, GNUTLS_E_SHORT_MEMORY_BUFFER is returned if the buffer given is not long enough, or 0 on success.

gnutls_pem_base64_encode_alloc ()

```
int                 gnutls_pem_base64_encode_alloc      (const char *msg,
                                                         const gnutls_datum_t *data,
                                                         gnutls_datum_t *result);
```

This function will convert the given data to printable data, using the base64 encoding. This is the encoding used in PEM messages. This function will allocate the required memory to hold the encoded data.

You should use gnutls_free() to free the returned data.

msg : is a message to be put in the encoded header

data : contains the raw data

result : will hold the newly allocated encoded data

Returns : On success, GNUTLS_E_SUCCESS (0) is returned, otherwise an error code is returned.

gnutls_pem_base64_decode_alloc ()

```
int                 gnutls_pem_base64_decode_alloc      (const char *header,
                                                         const gnutls_datum_t *b64_data,
                                                         gnutls_datum_t *result);
```

This function will decode the given encoded data. The decoded data will be allocated, and stored into result. If the header given is non null this function will search for "-----BEGIN header" and decode only this part. Otherwise it will decode the first PEM packet found.

You should use gnutls_free() to free the returned data.

header : The PEM header (eg. CERTIFICATE)

b64_data : contains the encoded data

result : the place where decoded data lie

Returns : On success, GNUTLS_E_SUCCESS (0) is returned, otherwise an error code is returned.

GNUTLS_KEY_DIGITAL_SIGNATURE

```
#define GNUTLS_KEY_DIGITAL_SIGNATURE~128
```

GNUTLS_KEY_NON_REPUDIATION

```
#define GNUTLS_KEY_NON_REPUDIATION~64
```

GNUTLS_KEY_KEY_ENCIPHERMENT

```
#define GNUTLS_KEY_KEY_ENCIPHERMENT~32
```

GNUTLS_KEY_DATA_ENCIPHERMENT

```
#define GNUTLS_KEY_DATA_ENCIPHERMENT~16
```

GNUTLS_KEY_KEY_AGREEMENT

```
#define GNUTLS_KEY_KEY_AGREEMENT~8
```

GNUTLS_KEY_KEY_CERT_SIGN

```
#define GNUTLS_KEY_KEY_CERT_SIGN~4
```

GNUTLS_KEY_CRL_SIGN

```
#define GNUTLS_KEY_CRL_SIGN   2
```

GNUTLS_KEY_ENCIPHER_ONLY

```
#define GNUTLS_KEY_ENCIPHER_ONLY~1
```

GNUTLS_KEY_DECIPHER_ONLY

```
#define GNUTLS_KEY_DECIPHER_ONLY~32768
```

gnutls_certificate_set_params_function ()

```
void                gnutls_certificate_set_params_function
                                                    (gnutls_certificate_credentials_t  ↩
                                                       res,
                                                     gnutls_params_function *func);
```

This function will set a callback in order for the server to get the Diffie-Hellman or RSA parameters for certificate authentication. The callback should return zero on success.

res **:** is a gnutls_certificate_credentials_t structure

func **:** is the function to be called

gnutls_anon_set_params_function ()

```
void                gnutls_anon_set_params_function     (gnutls_anon_server_credentials_t  ↩
    res,
                                                         gnutls_params_function *func);
```

This function will set a callback in order for the server to get the Diffie-Hellman or RSA parameters for anonymous authentication. The callback should return zero on success.

res **:** is a gnutls_anon_server_credentials_t structure

func **:** is the function to be called

gnutls_psk_set_params_function ()

```
void                gnutls_psk_set_params_function      (gnutls_psk_server_credentials_t  ↩
    res,
                                                         gnutls_params_function *func);
```

This function will set a callback in order for the server to get the Diffie-Hellman or RSA parameters for PSK authentication. The callback should return zero on success.

res : is a gnutls_psk_server_credentials_t structure

func : is the function to be called

gnutls_hex2bin ()

```
int                 gnutls_hex2bin                      (const char *hex_data,
                                                         size_t hex_size,
                                                         char *bin_data,
                                                         size_t *bin_size);
```

Convert a buffer with hex data to binary data.

hex_data : string with data in hex format

hex_size : size of hex data

bin_data : output array with binary data

bin_size : when calling *`bin_size` should hold size of `bin_data`, on return will hold actual size of `bin_data`.

Returns : GNUTLS_E_SUCCESS on success, otherwise an error.

Since 2.4.0

GNUTLS_E_SUCCESS

```
#define GNUTLS_E_SUCCESS 0
```

GNUTLS_E_UNKNOWN_COMPRESSION_ALGORITHM

```
#define~GNUTLS_E_UNKNOWN_COMPRESSION_ALGORITHM -3
```

GNUTLS_E_UNKNOWN_CIPHER_TYPE

```
#define~GNUTLS_E_UNKNOWN_CIPHER_TYPE -6
```

GNUTLS_E_LARGE_PACKET

```
#define~GNUTLS_E_LARGE_PACKET -7
```

GNUTLS_E_UNSUPPORTED_VERSION_PACKET

```
#define GNUTLS_E_UNSUPPORTED_VERSION_PACKET -8~/* GNUTLS_A_PROTOCOL_VERSION */
```

GNUTLS_E_UNEXPECTED_PACKET_LENGTH

```
#define GNUTLS_E_UNEXPECTED_PACKET_LENGTH -9~/* GNUTLS_A_RECORD_OVERFLOW */
```

GNUTLS_E_INVALID_SESSION

```
#define GNUTLS_E_INVALID_SESSION -10
```

GNUTLS_E_FATAL_ALERT_RECEIVED

```
#define GNUTLS_E_FATAL_ALERT_RECEIVED -12
```

GNUTLS_E_UNEXPECTED_PACKET

```
#define GNUTLS_E_UNEXPECTED_PACKET -15~/* GNUTLS_A_UNEXPECTED_MESSAGE */
```

GNUTLS_E_WARNING_ALERT_RECEIVED

```
#define GNUTLS_E_WARNING_ALERT_RECEIVED -16
```

GNUTLS_E_ERROR_IN_FINISHED_PACKET

```
#define GNUTLS_E_ERROR_IN_FINISHED_PACKET -18
```

GNUTLS_E_UNEXPECTED_HANDSHAKE_PACKET

```
#define GNUTLS_E_UNEXPECTED_HANDSHAKE_PACKET -19
```

GNUTLS_E_UNKNOWN_CIPHER_SUITE

```
#define~GNUTLS_E_UNKNOWN_CIPHER_SUITE -21~/* GNUTLS_A_HANDSHAKE_FAILURE */
```

GNUTLS_E_UNWANTED_ALGORITHM

```
#define~GNUTLS_E_UNWANTED_ALGORITHM -22
```

GNUTLS_E_MPI_SCAN_FAILED

```
#define~GNUTLS_E_MPI_SCAN_FAILED -23
```

GNUTLS_E_DECRYPTION_FAILED

```
#define GNUTLS_E_DECRYPTION_FAILED -24~/* GNUTLS_A_DECRYPTION_FAILED,  ↩
    GNUTLS_A_BAD_RECORD_MAC */
```

GNUTLS_E_MEMORY_ERROR

```
#define GNUTLS_E_MEMORY_ERROR -25
```

GNUTLS_E_DECOMPRESSION_FAILED

```
#define GNUTLS_E_DECOMPRESSION_FAILED -26~/* GNUTLS_A_DECOMPRESSION_FAILURE */
```

GNUTLS_E_COMPRESSION_FAILED

```
#define GNUTLS_E_COMPRESSION_FAILED -27
```

GNUTLS_E_AGAIN

```
#define GNUTLS_E_AGAIN -28
```

GNUTLS_E_EXPIRED

```
#define GNUTLS_E_EXPIRED -29
```

GNUTLS_E_DB_ERROR

```
#define GNUTLS_E_DB_ERROR -30
```

GNUTLS_E_SRP_PWD_ERROR

```
#define GNUTLS_E_SRP_PWD_ERROR -31
```

GNUTLS_E_INSUFFICIENT_CREDENTIALS

```
#define GNUTLS_E_INSUFFICIENT_CREDENTIALS -32
```

GNUTLS_E_INSUFICIENT_CREDENTIALS

```
#define GNUTLS_E_INSUFICIENT_CREDENTIALS GNUTLS_E_INSUFFICIENT_CREDENTIALS~/* for backwards ↩
    compatibility only */
```

GNUTLS_E_INSUFFICIENT_CRED

```
#define GNUTLS_E_INSUFFICIENT_CRED GNUTLS_E_INSUFFICIENT_CREDENTIALS
```

GNUTLS_E_INSUFICIENT_CRED

```
#define GNUTLS_E_INSUFICIENT_CRED GNUTLS_E_INSUFFICIENT_CREDENTIALS~/* for backwards  ↩
    compatibility only */
```

GNUTLS_E_HASH_FAILED

```
#define GNUTLS_E_HASH_FAILED -33
```

GNUTLS_E_BASE64_DECODING_ERROR

```
#define GNUTLS_E_BASE64_DECODING_ERROR -34
```

GNUTLS_E_MPI_PRINT_FAILED

```
#define~GNUTLS_E_MPI_PRINT_FAILED -35
```

GNUTLS_E_REHANDSHAKE

```
#define GNUTLS_E_REHANDSHAKE -37~/* GNUTLS_A_NO_RENEGOTIATION */
```

GNUTLS_E_GOT_APPLICATION_DATA

```
#define GNUTLS_E_GOT_APPLICATION_DATA -38
```

GNUTLS_E_RECORD_LIMIT_REACHED

```
#define GNUTLS_E_RECORD_LIMIT_REACHED -39
```

GNUTLS_E_ENCRYPTION_FAILED

```
#define GNUTLS_E_ENCRYPTION_FAILED -40
```

GNUTLS_E_PK_ENCRYPTION_FAILED

```
#define GNUTLS_E_PK_ENCRYPTION_FAILED -44
```

GNUTLS_E_PK_DECRYPTION_FAILED

```
#define GNUTLS_E_PK_DECRYPTION_FAILED -45
```

GNUTLS_E_PK_SIGN_FAILED

```
#define GNUTLS_E_PK_SIGN_FAILED -46
```

GNUTLS_E_X509_UNSUPPORTED_CRITICAL_EXTENSION

```
#define GNUTLS_E_X509_UNSUPPORTED_CRITICAL_EXTENSION -47
```

GNUTLS_E_KEY_USAGE_VIOLATION

```
#define GNUTLS_E_KEY_USAGE_VIOLATION -48
```

GNUTLS_E_NO_CERTIFICATE_FOUND

```
#define GNUTLS_E_NO_CERTIFICATE_FOUND -49~/* GNUTLS_A_BAD_CERTIFICATE */
```

GNUTLS_E_INVALID_REQUEST

```
#define GNUTLS_E_INVALID_REQUEST -50
```

GNUTLS_E_SHORT_MEMORY_BUFFER

```
#define GNUTLS_E_SHORT_MEMORY_BUFFER -51
```

GNUTLS_E_INTERRUPTED

```
#define GNUTLS_E_INTERRUPTED -52
```

GNUTLS_E_PUSH_ERROR

```
#define GNUTLS_E_PUSH_ERROR -53
```

GNUTLS_E_PULL_ERROR

```
#define GNUTLS_E_PULL_ERROR -54
```

GNUTLS_E_RECEIVED_ILLEGAL_PARAMETER

```
#define GNUTLS_E_RECEIVED_ILLEGAL_PARAMETER -55~/* GNUTLS_A_ILLEGAL_PARAMETER */
```

GNUTLS_E_REQUESTED_DATA_NOT_AVAILABLE

```
#define GNUTLS_E_REQUESTED_DATA_NOT_AVAILABLE -56
```

GNUTLS_E_PKCS1_WRONG_PAD

```
#define GNUTLS_E_PKCS1_WRONG_PAD -57
```

GNUTLS_E_RECEIVED_ILLEGAL_EXTENSION

```
#define GNUTLS_E_RECEIVED_ILLEGAL_EXTENSION -58
```

GNUTLS_E_INTERNAL_ERROR

```
#define GNUTLS_E_INTERNAL_ERROR -59
```

GNUTLS_E_DH_PRIME_UNACCEPTABLE

```
#define GNUTLS_E_DH_PRIME_UNACCEPTABLE -63
```

GNUTLS_E_FILE_ERROR

```
#define GNUTLS_E_FILE_ERROR -64
```

GNUTLS_E_TOO_MANY_EMPTY_PACKETS

```
#define GNUTLS_E_TOO_MANY_EMPTY_PACKETS -78
```

GNUTLS_E_UNKNOWN_PK_ALGORITHM

```
#define GNUTLS_E_UNKNOWN_PK_ALGORITHM -80
```

GNUTLS_E_INIT_LIBEXTRA

```
#define GNUTLS_E_INIT_LIBEXTRA -82
```

GNUTLS_E_LIBRARY_VERSION_MISMATCH

```
#define GNUTLS_E_LIBRARY_VERSION_MISMATCH -83
```

GNUTLS_E_NO_TEMPORARY_RSA_PARAMS

```
#define GNUTLS_E_NO_TEMPORARY_RSA_PARAMS -84
```

GNUTLS_E_LZO_INIT_FAILED

```
#define GNUTLS_E_LZO_INIT_FAILED -85
```

GNUTLS_E_NO_COMPRESSION_ALGORITHMS

```
#define GNUTLS_E_NO_COMPRESSION_ALGORITHMS -86
```

GNUTLS_E_NO_CIPHER_SUITES

```
#define GNUTLS_E_NO_CIPHER_SUITES -87
```

GNUTLS_E_OPENPGP_GETKEY_FAILED

```
#define GNUTLS_E_OPENPGP_GETKEY_FAILED -88
```

GNUTLS_E_PK_SIG_VERIFY_FAILED

```
#define GNUTLS_E_PK_SIG_VERIFY_FAILED -89
```

GNUTLS_E_ILLEGAL_SRP_USERNAME

```
#define GNUTLS_E_ILLEGAL_SRP_USERNAME -90
```

GNUTLS_E_SRP_PWD_PARSING_ERROR

```
#define GNUTLS_E_SRP_PWD_PARSING_ERROR -91
```

GNUTLS_E_NO_TEMPORARY_DH_PARAMS

```
#define GNUTLS_E_NO_TEMPORARY_DH_PARAMS -93
```

GNUTLS_E_ASN1_ELEMENT_NOT_FOUND

```
#define GNUTLS_E_ASN1_ELEMENT_NOT_FOUND -67
```

GNUTLS_E_ASN1_IDENTIFIER_NOT_FOUND

```
#define GNUTLS_E_ASN1_IDENTIFIER_NOT_FOUND -68
```

GNUTLS_E_ASN1_DER_ERROR

```
#define GNUTLS_E_ASN1_DER_ERROR -69
```

GNUTLS_E_ASN1_VALUE_NOT_FOUND

```
#define GNUTLS_E_ASN1_VALUE_NOT_FOUND -70
```

GNUTLS_E_ASN1_GENERIC_ERROR

```
#define GNUTLS_E_ASN1_GENERIC_ERROR -71
```

GNUTLS_E_ASN1_VALUE_NOT_VALID

```
#define GNUTLS_E_ASN1_VALUE_NOT_VALID -72
```

GNUTLS_E_ASN1_TAG_ERROR

```
#define GNUTLS_E_ASN1_TAG_ERROR -73
```

GNUTLS_E_ASN1_TAG_IMPLICIT

```
#define GNUTLS_E_ASN1_TAG_IMPLICIT -74
```

GNUTLS_E_ASN1_TYPE_ANY_ERROR

```
#define GNUTLS_E_ASN1_TYPE_ANY_ERROR -75
```

GNUTLS_E_ASN1_SYNTAX_ERROR

```
#define GNUTLS_E_ASN1_SYNTAX_ERROR -76
```

GNUTLS_E_ASN1_DER_OVERFLOW

```
#define GNUTLS_E_ASN1_DER_OVERFLOW -77
```

GNUTLS_E_OPENPGP_UID_REVOKED

```
#define GNUTLS_E_OPENPGP_UID_REVOKED -79
```

GNUTLS_E_CERTIFICATE_ERROR

```
#define GNUTLS_E_CERTIFICATE_ERROR -43
```

GNUTLS_E_X509_CERTIFICATE_ERROR

```
#define GNUTLS_E_X509_CERTIFICATE_ERROR GNUTLS_E_CERTIFICATE_ERROR
```

GNUTLS_E_CERTIFICATE_KEY_MISMATCH

```
#define GNUTLS_E_CERTIFICATE_KEY_MISMATCH -60
```

GNUTLS_E_UNSUPPORTED_CERTIFICATE_TYPE

```
#define GNUTLS_E_UNSUPPORTED_CERTIFICATE_TYPE -61~/* GNUTLS_A_UNSUPPORTED_CERTIFICATE */
```

GNUTLS_E_X509_UNKNOWN_SAN

```
#define GNUTLS_E_X509_UNKNOWN_SAN -62
```

GNUTLS_E_OPENPGP_FINGERPRINT_UNSUPPORTED

```
#define GNUTLS_E_OPENPGP_FINGERPRINT_UNSUPPORTED -94
```

GNUTLS_E_X509_UNSUPPORTED_ATTRIBUTE

```
#define GNUTLS_E_X509_UNSUPPORTED_ATTRIBUTE -95
```

GNUTLS_E_UNKNOWN_HASH_ALGORITHM

```
#define GNUTLS_E_UNKNOWN_HASH_ALGORITHM -96
```

GNUTLS_E_UNKNOWN_PKCS_CONTENT_TYPE

```
#define GNUTLS_E_UNKNOWN_PKCS_CONTENT_TYPE -97
```

GNUTLS_E_UNKNOWN_PKCS_BAG_TYPE

```
#define GNUTLS_E_UNKNOWN_PKCS_BAG_TYPE -98
```

GNUTLS_E_INVALID_PASSWORD

```
#define GNUTLS_E_INVALID_PASSWORD -99
```

GNUTLS_E_MAC_VERIFY_FAILED

```
#define GNUTLS_E_MAC_VERIFY_FAILED -100~/* for PKCS #12 MAC */
```

GNUTLS_E_CONSTRAINT_ERROR

```
#define GNUTLS_E_CONSTRAINT_ERROR -101
```

GNUTLS_E_WARNING_IA_IPHF_RECEIVED

```
#define GNUTLS_E_WARNING_IA_IPHF_RECEIVED -102
```

GNUTLS_E_WARNING_IA_FPHF_RECEIVED

```
#define GNUTLS_E_WARNING_IA_FPHF_RECEIVED -103
```

GNUTLS_E_IA_VERIFY_FAILED

```
#define GNUTLS_E_IA_VERIFY_FAILED -104
```

GNUTLS_E_UNKNOWN_ALGORITHM

```
#define GNUTLS_E_UNKNOWN_ALGORITHM -105
```

GNUTLS_E_UNSUPPORTED_SIGNATURE_ALGORITHM

```
#define GNUTLS_E_UNSUPPORTED_SIGNATURE_ALGORITHM -106
```

GNUTLS_E_SAFE_RENEGOTIATION_FAILED

```
#define GNUTLS_E_SAFE_RENEGOTIATION_FAILED -107
```

GNUTLS_E_UNSAFE_RENEGOTIATION_DENIED

```
#define GNUTLS_E_UNSAFE_RENEGOTIATION_DENIED -108
```

GNUTLS_E_UNKNOWN_SRP_USERNAME

```
#define GNUTLS_E_UNKNOWN_SRP_USERNAME -109
```

GNUTLS_E_BASE64_ENCODING_ERROR

```
#define GNUTLS_E_BASE64_ENCODING_ERROR -201
```

GNUTLS_E_INCOMPATIBLE_GCRYPT_LIBRARY

```
#define GNUTLS_E_INCOMPATIBLE_GCRYPT_LIBRARY -202~/* obsolete */
```

GNUTLS_E_INCOMPATIBLE_CRYPTO_LIBRARY

```
#define GNUTLS_E_INCOMPATIBLE_CRYPTO_LIBRARY -202
```

GNUTLS_E_INCOMPATIBLE_LIBTASN1_LIBRARY

```
#define GNUTLS_E_INCOMPATIBLE_LIBTASN1_LIBRARY -203
```

GNUTLS_E_OPENPGP_KEYRING_ERROR

```
#define GNUTLS_E_OPENPGP_KEYRING_ERROR -204
```

GNUTLS_E_X509_UNSUPPORTED_OID

```
#define GNUTLS_E_X509_UNSUPPORTED_OID -205
```

GNUTLS_E_RANDOM_FAILED

```
#define GNUTLS_E_RANDOM_FAILED -206
```

GNUTLS_E_BASE64_UNEXPECTED_HEADER_ERROR

```
#define GNUTLS_E_BASE64_UNEXPECTED_HEADER_ERROR -207
```

GNUTLS_E_OPENPGP_SUBKEY_ERROR

```
#define GNUTLS_E_OPENPGP_SUBKEY_ERROR -208
```

GNUTLS_E_CRYPTO_ALREADY_REGISTERED

```
#define GNUTLS_E_CRYPTO_ALREADY_REGISTERED -209
```

GNUTLS_E_HANDSHAKE_TOO_LARGE

```
#define GNUTLS_E_HANDSHAKE_TOO_LARGE -210
```

GNUTLS_E_CRYPTODEV_IOCTL_ERROR

```
#define GNUTLS_E_CRYPTODEV_IOCTL_ERROR -211
```

GNUTLS_E_CRYPTODEV_DEVICE_ERROR

```
#define GNUTLS_E_CRYPTODEV_DEVICE_ERROR -212
```

GNUTLS_E_UNIMPLEMENTED_FEATURE

```
#define GNUTLS_E_UNIMPLEMENTED_FEATURE -1250
```

GNUTLS_E_APPLICATION_ERROR_MAX

```
#define GNUTLS_E_APPLICATION_ERROR_MAX -65000
```

GNUTLS_E_APPLICATION_ERROR_MIN

```
#define GNUTLS_E_APPLICATION_ERROR_MIN -65500
```

1.2 extra

extra —

Synopsis

```
#define             GNUTLS_EXTRA_VERSION
enum                gnutls_ia_apptype_t;
int                 (*gnutls_ia_avp_func)               (gnutls_session_t session,
                                                         void *ptr,
                                                         const char *last,
                                                         size_t lastlen,
                                                         char **next,
                                                         size_t *nextlen);
void                gnutls_ia_free_client_credentials   (gnutls_ia_client_credentials_t sc
int                 gnutls_ia_allocate_client_credentials
                                                        (gnutls_ia_client_credentials_t *s
void                gnutls_ia_free_server_credentials   (gnutls_ia_server_credentials_t sc
int                 gnutls_ia_allocate_server_credentials
                                                        (gnutls_ia_server_credentials_t *s
void                gnutls_ia_set_client_avp_function   (gnutls_ia_client_credentials_t cr
                                                         gnutls_ia_avp_func avp_func);
void                gnutls_ia_set_client_avp_ptr        (gnutls_ia_client_credentials_t cr
                                                         void *ptr);
void *              gnutls_ia_get_client_avp_ptr        (gnutls_ia_client_credentials_t cr
void                gnutls_ia_set_server_avp_function   (gnutls_ia_server_credentials_t cr
                                                         gnutls_ia_avp_func avp_func);
void                gnutls_ia_set_server_avp_ptr        (gnutls_ia_server_credentials_t cr
                                                         void *ptr);
void *              gnutls_ia_get_server_avp_ptr        (gnutls_ia_server_credentials_t cr
int                 gnutls_ia_handshake_p               (gnutls_session_t session);
int                 gnutls_ia_handshake                 (gnutls_session_t session);
int                 gnutls_ia_permute_inner_secret      (gnutls_session_t session,
                                                         size_t session_keys_size,
                                                         const char *session_keys);
int                 gnutls_ia_endphase_send             (gnutls_session_t session,
                                                         int final_p);
int                 gnutls_ia_verify_endphase           (gnutls_session_t session,
                                                         const char *checksum);
ssize_t             gnutls_ia_send                      (gnutls_session_t session,
                                                         const char *data,
                                                         size_t sizeofdata);
ssize_t             gnutls_ia_recv                      (gnutls_session_t session,
                                                         char *data,
```

```
                                                        size_t sizeofdata);
int                 gnutls_ia_generate_challenge        (gnutls_session_t session,
                                                         size_t buffer_size,
                                                         char *buffer);
void                gnutls_ia_extract_inner_secret      (gnutls_session_t session,
                                                         char *buffer);
void                gnutls_ia_enable                    (gnutls_session_t session,
                                                         int allow_skip_on_resume);
int                 gnutls_global_init_extra            (void);
int                 gnutls_register_md5_handler         (void);
const char *        gnutls_extra_check_version          (const char *req_version);
```

Description

Details

GNUTLS_EXTRA_VERSION

```
#define GNUTLS_EXTRA_VERSION GNUTLS_VERSION
```

enum gnutls_ia_apptype_t

```
typedef enum
{
  GNUTLS_IA_APPLICATION_PAYLOAD = 0,
  GNUTLS_IA_INTERMEDIATE_PHASE_FINISHED = 1,
  GNUTLS_IA_FINAL_PHASE_FINISHED = 2
} gnutls_ia_apptype_t;
```

Enumeration of different certificate encoding formats.

GNUTLS_IA_APPLICATION_PAYLOAD TLS/IA application payload.

GNUTLS_IA_INTERMEDIATE_PHASE_FINISHED TLS/IA intermediate phase finished.

GNUTLS_IA_FINAL_PHASE_FINISHED TLS/IA final phase finished.

gnutls_ia_avp_func ()

```
int                 (*gnutls_ia_avp_func)               (gnutls_session_t session,
                                                         void *ptr,
                                                         const char *last,
                                                         size_t lastlen,
                                                         char **next,
                                                         size_t *nextlen);
```

session :

ptr :

last :

lastlen :

next :

nextlen :

Returns :

gnutls_ia_free_client_credentials ()

```
void                gnutls_ia_free_client_credentials   (gnutls_ia_client_credentials_t sc) ↩
    ;
```

This structure is complex enough to manipulate directly thus this helper function is provided in order to free (deallocate) it.

sc : is a gnutls_ia_client_credentials_t structure.

gnutls_ia_allocate_client_credentials ()

```
int                 gnutls_ia_allocate_client_credentials
                                                        (gnutls_ia_client_credentials_t *sc ↩
                                                            );
```

This structure is complex enough to manipulate directly thus this helper function is provided in order to allocate it.

Adding this credential to a session will enable TLS/IA, and will require an Application Phase after the TLS handshake (if the server support TLS/IA). Use gnutls_ia_enable() to toggle the TLS/IA mode.

sc : is a pointer to a gnutls_ia_server_credentials_t structure.

Returns : On success, GNUTLS_E_SUCCESS (0) is returned, otherwise an error code is returned.

gnutls_ia_free_server_credentials ()

```
void                gnutls_ia_free_server_credentials   (gnutls_ia_server_credentials_t sc) ↩
    ;
```

This structure is complex enough to manipulate directly thus this helper function is provided in order to free (deallocate) it.

sc : is a gnutls_ia_server_credentials_t structure.

gnutls_ia_allocate_server_credentials ()

```
int                 gnutls_ia_allocate_server_credentials
                                                        (gnutls_ia_server_credentials_t *sc ↩
                                                            );
```

This structure is complex enough to manipulate directly thus this helper function is provided in order to allocate it.

Adding this credential to a session will enable TLS/IA, and will require an Application Phase after the TLS handshake (if the client support TLS/IA). Use gnutls_ia_enable() to toggle the TLS/IA mode.

sc : is a pointer to a gnutls_ia_server_credentials_t structure.

Returns : On success, GNUTLS_E_SUCCESS (0) is returned, otherwise an error code is returned.

gnutls_ia_set_client_avp_function ()

```
void                gnutls_ia_set_client_avp_function   (gnutls_ia_client_credentials_t ↩
    cred,
                                                         gnutls_ia_avp_func avp_func);
```

Set the TLS/IA AVP callback handler used for the session.

The AVP callback is called to process AVPs received from the server, and to get a new AVP to send to the server.

The callback's function form is: int (*avp_func) (gnutls_session_t session, void *ptr, const char *last, size_t lastlen, char **next, size_t *nextlen);

The *session* parameter is the gnutls_session_t structure corresponding to the current session. The *ptr* parameter is the application hook pointer, set through gnutls_ia_set_client_avp_ptr(). The AVP received from the server is present in *last* of *lastlen* size, which will be NULL on the first invocation. The newly allocated output AVP to send to the server should be placed in **next* of **nextlen* size.

The callback may invoke gnutls_ia_permute_inner_secret() to mix any generated session keys with the TLS/IA inner secret.

Return 0 (GNUTLS_IA_APPLICATION_PAYLOAD) on success, or a negative error code to abort the TLS/IA handshake.

Note that the callback must use allocate the *next* parameter using gnutls_malloc(), because it is released via gnutls_free() by the TLS/IA handshake function.

cred **:** is a gnutls_ia_client_credentials_t structure.

avp_func **:** is the callback function

gnutls_ia_set_client_avp_ptr ()

```
void                gnutls_ia_set_client_avp_ptr        (gnutls_ia_client_credentials_t ↩
    cred,
                                                         void *ptr);
```

Sets the pointer that will be provided to the TLS/IA callback function as the first argument.

cred **:** is a gnutls_ia_client_credentials_t structure.

ptr **:** is the pointer

gnutls_ia_get_client_avp_ptr ()

```
void *              gnutls_ia_get_client_avp_ptr        (gnutls_ia_client_credentials_t ↩
    cred);
```

Returns the pointer that will be provided to the TLS/IA callback function as the first argument.

cred **:** is a gnutls_ia_client_credentials_t structure.

Returns **:** The client callback data pointer.

gnutls_ia_set_server_avp_function ()

```
void                gnutls_ia_set_server_avp_function   (gnutls_ia_server_credentials_t ↩
    cred,
                                                         gnutls_ia_avp_func avp_func);
```

cred **:**

avp_func **:**

gnutls_ia_set_server_avp_ptr ()

```
void                gnutls_ia_set_server_avp_ptr        (gnutls_ia_server_credentials_t  ↩
    cred,
                                                         void *ptr);
```

Sets the pointer that will be provided to the TLS/IA callback function as the first argument.

cred : is a gnutls_ia_client_credentials_t structure.

ptr : is the pointer

gnutls_ia_get_server_avp_ptr ()

```
void *              gnutls_ia_get_server_avp_ptr        (gnutls_ia_server_credentials_t  ↩
    cred);
```

Returns the pointer that will be provided to the TLS/IA callback function as the first argument.

cred : is a gnutls_ia_client_credentials_t structure.

Returns : The server callback data pointer.

gnutls_ia_handshake_p ()

```
int                 gnutls_ia_handshake_p               (gnutls_session_t session);
```

Predicate to be used after gnutls_handshake() to decide whether to invoke gnutls_ia_handshake(). Usable by both clients and servers.

session : is a gnutls_session_t structure.

Returns : non-zero if TLS/IA handshake is expected, zero otherwise.

gnutls_ia_handshake ()

```
int                 gnutls_ia_handshake                 (gnutls_session_t session);
```

Perform a TLS/IA handshake. This should be called after gnutls_handshake() iff gnutls_ia_handshake_p().

session : is a gnutls_session_t structure.

Returns : On success, GNUTLS_E_SUCCESS (zero) is returned, otherwise an error code is returned.

gnutls_ia_permute_inner_secret ()

```
int                 gnutls_ia_permute_inner_secret      (gnutls_session_t session,
                                                         size_t session_keys_size,
                                                         const char *session_keys);
```

Permute the inner secret using the generated session keys.

This can be called in the TLS/IA AVP callback to mix any generated session keys with the TLS/IA inner secret.

session : is a gnutls_session_t structure.

session_keys_size : Size of generated session keys (0 if none).

session_keys : Generated session keys, used to permute inner secret (NULL if none).

Returns : Return zero on success, or a negative error code.

gnutls_ia_endphase_send ()

```
int                 gnutls_ia_endphase_send             (gnutls_session_t session,
                                                         int final_p);
```

Send a TLS/IA end phase message.

In the client, this should only be used to acknowledge an end phase message sent by the server.

In the server, this can be called instead of gnutls_ia_send() if the server wishes to end an application phase.

session : is a gnutls_session_t structure.

final_p : Set iff this should signal the final phase.

Returns : Return 0 on success, or an error code.

gnutls_ia_verify_endphase ()

```
int                 gnutls_ia_verify_endphase           (gnutls_session_t session,
                                                         const char *checksum);
```

Verify TLS/IA end phase checksum data. If verification fails, the GNUTLS_A_INNER_APPLICATION_VERIFICATION alert is sent to the other sie.

This function is called when gnutls_ia_recv() return GNUTLS_E_WARNING_IA_IPHF_RECEIVED or GNUTLS_E_WARNING_IA_

session : is a gnutls_session_t structure.

checksum : 12-byte checksum data, received from gnutls_ia_recv().

Returns : Return 0 on successful verification, or an error code. If the checksum verification of the end phase message fails, GNUTLS_E_IA_VERIFY_FAILED is returned.

gnutls_ia_send ()

```
ssize_t             gnutls_ia_send                      (gnutls_session_t session,
                                                         const char *data,
                                                         size_t sizeofdata);
```

Send TLS/IA application payload data. This function has the similar semantics with send(). The only difference is that it accepts a GnuTLS session, and uses different error codes.

The TLS/IA protocol is synchronous, so you cannot send more than one packet at a time. The client always send the first packet.

To finish an application phase in the server, use gnutls_ia_endphase_send(). The client cannot end an application phase unilaterally; rather, a client is required to respond with an endphase of its own if gnutls_ia_recv indicates that the server has sent one.

If the EINTR is returned by the internal push function (the default is send()} then GNUTLS_E_INTERRUPTED will be returned. If GNUTLS_E_INTERRUPTED or GNUTLS_E_AGAIN is returned, you must call this function again, with the same parameters; alternatively you could provide a NULL pointer for data, and 0 for size.

session : is a gnutls_session_t structure.

data : contains the data to send

sizeofdata : is the length of the data

Returns : The number of bytes sent, or a negative error code.

gnutls_ia_recv ()

```
ssize_t             gnutls_ia_recv                      (gnutls_session_t session,
                                                         char *data,
                                                         size_t sizeofdata);
```

Receive TLS/IA data. This function has the similar semantics with recv(). The only difference is that it accepts a GnuTLS session, and uses different error codes.

If the server attempt to finish an application phase, this function will return GNUTLS_E_WARNING_IA_IPHF_RECEIVED or GNUTLS_E_WARNING_IA_FPHF_RECEIVED. The caller should then invoke gnutls_ia_verify_endphase(), and if it runs the client side, also send an endphase message of its own using gnutls_ia_endphase_send.

If EINTR is returned by the internal push function (the default is *code*{recv()}) then GNUTLS_E_INTERRUPTED will be returned. If GNUTLS_E_INTERRUPTED or GNUTLS_E_AGAIN is returned, you must call this function again, with the same parameters; alternatively you could provide a NULL pointer for data, and 0 for size.

session : is a gnutls_session_t structure.

data : the buffer that the data will be read into, must hold >= 12 bytes.

sizeofdata : the number of requested bytes, must be >= 12.

Returns : The number of bytes received. A negative error code is returned in case of an error. The GNUTLS_E_WARNING_IA_IPHF_R and GNUTLS_E_WARNING_IA_FPHF_RECEIVED errors are returned when an application phase finished message has been sent by the server.

gnutls_ia_generate_challenge ()

```
int                 gnutls_ia_generate_challenge        (gnutls_session_t session,
                                                         size_t buffer_size,
                                                         char *buffer);
```

Generate an application challenge that the client cannot control or predict, based on the TLS/IA inner secret.

session : is a gnutls_session_t structure.

buffer_size : size of output buffer.

buffer : pre-allocated buffer to contain *buffer_size* bytes of output.

Returns : Returns 0 on success, or an negative error code.

gnutls_ia_extract_inner_secret ()

```
void                gnutls_ia_extract_inner_secret      (gnutls_session_t session,
                                                         char *buffer);
```

Copy the 48 bytes large inner secret into the specified buffer

This function is typically used after the TLS/IA handshake has concluded. The TLS/IA inner secret can be used as input to a PRF to derive session keys. Do not use the inner secret directly as a session key, because for a resumed session that does not include an application phase, the inner secret will be identical to the inner secret in the original session. It is important to include, for example, the client and server randomness when deriving a sesssion key from the inner secret.

session : is a gnutls_session_t structure.

buffer : pre-allocated buffer to hold 48 bytes of inner secret.

gnutls_ia_enable ()

```
void                gnutls_ia_enable                    (gnutls_session_t session,
                                                         int allow_skip_on_resume);
```

Specify whether we must advertise support for the TLS/IA extension during the handshake.

At the client side, we always advertise TLS/IA if gnutls_ia_enable was called before the handshake; at the server side, we also require that the client has advertised that it wants to run TLS/IA before including the advertisement, as required by the protocol.

Similarly, at the client side we always advertise that we allow TLS/IA to be skipped for resumed sessions if *allow_skip_on_resume* is non-zero; at the server side, we also require that the session is indeed resumable and that the client has also advertised that it allows TLS/IA to be skipped for resumed sessions.

After the TLS handshake, call gnutls_ia_handshake_p() to find out whether both parties agreed to do a TLS/IA handshake, before calling gnutls_ia_handshake() or one of the lower level gnutls_ia_* functions.

session : is a gnutls_session_t structure.

allow_skip_on_resume : non-zero if local party allows to skip the TLS/IA application phases for a resumed session.

gnutls_global_init_extra ()

```
int                 gnutls_global_init_extra            (void);
```

This function initializes the global state of gnutls-extra library to defaults.

Note that gnutls_global_init() has to be called before this function. If this function is not called then the gnutls-extra library will not be usable.

This function is not thread safe, see the discussion for gnutls_global_init() on how to deal with that.

Returns : On success, GNUTLS_E_SUCCESS (zero) is returned, otherwise an error code is returned.

gnutls_register_md5_handler ()

```
int                 gnutls_register_md5_handler         (void);
```

Register a non-libgcrypt based MD5 and HMAC-MD5 handler. This is useful if you run Libgcrypt in FIPS-mode. Normally TLS requires use of MD5, so without this you cannot use GnuTLS with libgcrypt in FIPS mode.

Returns : GNUTLS_E_SUCCESS on success, otherwise an error.

Since 2.6.0

gnutls_extra_check_version ()

```
const char *        gnutls_extra_check_version          (const char *req_version);
```

Check GnuTLS Extra Library version.

See GNUTLS_EXTRA_VERSION for a suitable *req_version* string.

req_version : version string to compare with, or NULL.

Returns : Check that the version of the library is at minimum the one given as a string in *req_version* and return the actual version string of the library; return NULL if the condition is not met. If NULL is passed to this function no check is done and only the version string is returned.

1.3 x509

x509 —

Synopsis

```
#define             GNUTLS_OID_X520_COUNTRY_NAME
#define             GNUTLS_OID_X520_ORGANIZATION_NAME
#define             GNUTLS_OID_X520_ORGANIZATIONAL_UNIT_NAME
#define             GNUTLS_OID_X520_COMMON_NAME
#define             GNUTLS_OID_X520_LOCALITY_NAME
#define             GNUTLS_OID_X520_STATE_OR_PROVINCE_NAME
#define             GNUTLS_OID_X520_INITIALS
#define             GNUTLS_OID_X520_GENERATION_QUALIFIER
#define             GNUTLS_OID_X520_SURNAME
#define             GNUTLS_OID_X520_GIVEN_NAME
#define             GNUTLS_OID_X520_TITLE
#define             GNUTLS_OID_X520_DN_QUALIFIER
#define             GNUTLS_OID_X520_PSEUDONYM
#define             GNUTLS_OID_X520_POSTALCODE
#define             GNUTLS_OID_X520_NAME
#define             GNUTLS_OID_LDAP_DC
#define             GNUTLS_OID_LDAP_UID
#define             GNUTLS_OID_PKCS9_EMAIL
#define             GNUTLS_OID_PKIX_DATE_OF_BIRTH
#define             GNUTLS_OID_PKIX_PLACE_OF_BIRTH
#define             GNUTLS_OID_PKIX_GENDER
#define             GNUTLS_OID_PKIX_COUNTRY_OF_CITIZENSHIP
#define             GNUTLS_OID_PKIX_COUNTRY_OF_RESIDENCE
#define             GNUTLS_KP_TLS_WWW_SERVER
#define             GNUTLS_KP_TLS_WWW_CLIENT
#define             GNUTLS_KP_CODE_SIGNING
#define             GNUTLS_KP_EMAIL_PROTECTION
#define             GNUTLS_KP_TIME_STAMPING
#define             GNUTLS_KP_OCSP_SIGNING
#define             GNUTLS_KP_ANY
#define             GNUTLS_FSAN_SET
#define             GNUTLS_FSAN_APPEND
enum                gnutls_certificate_import_flags;
int                 gnutls_x509_crt_init                (gnutls_x509_crt_t *cert);
void                gnutls_x509_crt_deinit              (gnutls_x509_crt_t cert);
int                 gnutls_x509_crt_import              (gnutls_x509_crt_t cert,
                                                         const gnutls_datum_t *data,
                                                         gnutls_x509_crt_fmt_t format);
int                 gnutls_x509_crt_list_import         (gnutls_x509_crt_t *certs,
                                                         unsigned int *cert_max,
                                                         const gnutls_datum_t *data,
                                                         gnutls_x509_crt_fmt_t format,
                                                         unsigned int flags);
int                 gnutls_x509_crt_export              (gnutls_x509_crt_t cert,
                                                         gnutls_x509_crt_fmt_t format,
                                                         void *output_data,
                                                         size_t *output_data_size);
int                 gnutls_x509_crt_get_issuer_dn       (gnutls_x509_crt_t cert,
                                                         char *buf,
```

```
                                                    size_t *sizeof_buf);
int                 gnutls_x509_crt_get_issuer_dn_oid   (gnutls_x509_crt_t cert,
                                                    int indx,
                                                    void *oid,
                                                    size_t *sizeof_oid);
int                 gnutls_x509_crt_get_issuer_dn_by_oid
                                                   (gnutls_x509_crt_t cert,
                                                    const char *oid,
                                                    int indx,
                                                    unsigned int raw_flag,
                                                    void *buf,
                                                    size_t *sizeof_buf);
int                 gnutls_x509_crt_get_dn          (gnutls_x509_crt_t cert,
                                                    char *buf,
                                                    size_t *sizeof_buf);
int                 gnutls_x509_crt_get_dn_oid      (gnutls_x509_crt_t cert,
                                                    int indx,
                                                    void *oid,
                                                    size_t *sizeof_oid);
int                 gnutls_x509_crt_get_dn_by_oid   (gnutls_x509_crt_t cert,
                                                    const char *oid,
                                                    int indx,
                                                    unsigned int raw_flag,
                                                    void *buf,
                                                    size_t *sizeof_buf);
int                 gnutls_x509_crt_check_hostname  (gnutls_x509_crt_t cert,
                                                    const char *hostname);
int                 gnutls_x509_crt_get_signature_algorithm
                                                   (gnutls_x509_crt_t cert);
int                 gnutls_x509_crt_get_signature   (gnutls_x509_crt_t cert,
                                                    char *sig,
                                                    size_t *sizeof_sig);
int                 gnutls_x509_crt_get_version     (gnutls_x509_crt_t cert);
int                 gnutls_x509_crt_get_key_id      (gnutls_x509_crt_t crt,
                                                    unsigned int flags,
                                                    unsigned char *output_data,
                                                    size_t *output_data_size);
int                 gnutls_x509_crt_set_authority_key_id
                                                   (gnutls_x509_crt_t cert,
                                                    const void *id,
                                                    size_t id_size);
int                 gnutls_x509_crt_get_authority_key_id
                                                   (gnutls_x509_crt_t cert,
                                                    void *ret,
                                                    size_t *ret_size,
                                                    unsigned int *critical);
int                 gnutls_x509_crt_get_subject_key_id  (gnutls_x509_crt_t cert,
                                                    void *ret,
                                                    size_t *ret_size,
                                                    unsigned int *critical);
#define             GNUTLS_CRL_REASON_UNUSED
#define             GNUTLS_CRL_REASON_KEY_COMPROMISE
#define             GNUTLS_CRL_REASON_CA_COMPROMISE
#define             GNUTLS_CRL_REASON_AFFILIATION_CHANGED
#define             GNUTLS_CRL_REASON_SUPERSEDED
#define             GNUTLS_CRL_REASON_SUPERSEEDED
#define             GNUTLS_CRL_REASON_CESSATION_OF_OPERATION
```

```
#define             GNUTLS_CRL_REASON_CERTIFICATE_HOLD
#define             GNUTLS_CRL_REASON_PRIVILEGE_WITHDRAWN
#define             GNUTLS_CRL_REASON_AA_COMPROMISE
int                 gnutls_x509_crt_get_crl_dist_points (gnutls_x509_crt_t cert,
                                                         unsigned int seq,
                                                         void *ret,
                                                         size_t *ret_size,
                                                         unsigned int *reason_flags,
                                                         unsigned int *critical);
int                 gnutls_x509_crt_set_crl_dist_points2
                                                        (gnutls_x509_crt_t crt,
                                                         gnutls_x509_subject_alt_name_t ty
                                                         const void *data,
                                                         unsigned int data_size,
                                                         unsigned int reason_flags);
int                 gnutls_x509_crt_set_crl_dist_points (gnutls_x509_crt_t crt,
                                                         gnutls_x509_subject_alt_name_t ty
                                                         const void *data_string,
                                                         unsigned int reason_flags);
int                 gnutls_x509_crt_cpy_crl_dist_points (gnutls_x509_crt_t dst,
                                                         gnutls_x509_crt_t src);
time_t              gnutls_x509_crt_get_activation_time (gnutls_x509_crt_t cert);
time_t              gnutls_x509_crt_get_expiration_time (gnutls_x509_crt_t cert);
int                 gnutls_x509_crt_get_serial          (gnutls_x509_crt_t cert,
                                                         void *result,
                                                         size_t *result_size);
int                 gnutls_x509_crt_get_pk_algorithm    (gnutls_x509_crt_t cert,
                                                         unsigned int *bits);
int                 gnutls_x509_crt_get_pk_rsa_raw      (gnutls_x509_crt_t crt,
                                                         gnutls_datum_t *m,
                                                         gnutls_datum_t *e);
int                 gnutls_x509_crt_get_pk_dsa_raw      (gnutls_x509_crt_t crt,
                                                         gnutls_datum_t *p,
                                                         gnutls_datum_t *q,
                                                         gnutls_datum_t *g,
                                                         gnutls_datum_t *y);
int                 gnutls_x509_crt_get_subject_alt_name
                                                        (gnutls_x509_crt_t cert,
                                                         unsigned int seq,
                                                         void *ret,
                                                         size_t *ret_size,
                                                         unsigned int *critical);
int                 gnutls_x509_crt_get_subject_alt_name2
                                                        (gnutls_x509_crt_t cert,
                                                         unsigned int seq,
                                                         void *ret,
                                                         size_t *ret_size,
                                                         unsigned int *ret_type,
                                                         unsigned int *critical);
int                 gnutls_x509_crt_get_subject_alt_othername_oid
                                                        (gnutls_x509_crt_t cert,
                                                         unsigned int seq,
                                                         void *ret,
                                                         size_t *ret_size);
int                 gnutls_x509_crt_get_issuer_alt_name (gnutls_x509_crt_t cert,
                                                         unsigned int seq,
                                                         void *ret,
```

```
                                                 size_t *ret_size,
                                                 unsigned int *critical);
int                 gnutls_x509_crt_get_issuer_alt_name2
                                                (gnutls_x509_crt_t cert,
                                                 unsigned int seq,
                                                 void *ret,
                                                 size_t *ret_size,
                                                 unsigned int *ret_type,
                                                 unsigned int *critical);
int                 gnutls_x509_crt_get_issuer_alt_othername_oid
                                                (gnutls_x509_crt_t cert,
                                                 unsigned int seq,
                                                 void *ret,
                                                 size_t *ret_size);
int                 gnutls_x509_crt_get_ca_status       (gnutls_x509_crt_t cert,
                                                 unsigned int *critical);
int                 gnutls_x509_crt_get_basic_constraints
                                                (gnutls_x509_crt_t cert,
                                                 unsigned int *critical,
                                                 int *ca,
                                                 int *pathlen);
int                 gnutls_x509_crt_get_key_usage       (gnutls_x509_crt_t cert,
                                                 unsigned int *key_usage,
                                                 unsigned int *critical);
int                 gnutls_x509_crt_set_key_usage       (gnutls_x509_crt_t crt,
                                                 unsigned int usage);
int                 gnutls_x509_crt_get_proxy           (gnutls_x509_crt_t cert,
                                                 unsigned int *critical,
                                                 int *pathlen,
                                                 char **policyLanguage,
                                                 char **policy,
                                                 size_t *sizeof_policy);
int                 gnutls_x509_dn_oid_known            (const char *oid);
int                 gnutls_x509_crt_get_extension_oid   (gnutls_x509_crt_t cert,
                                                 int indx,
                                                 void *oid,
                                                 size_t *sizeof_oid);
int                 gnutls_x509_crt_get_extension_by_oid
                                                (gnutls_x509_crt_t cert,
                                                 const char *oid,
                                                 int indx,
                                                 void *buf,
                                                 size_t *sizeof_buf,
                                                 unsigned int *critical);
int                 gnutls_x509_crt_get_extension_info  (gnutls_x509_crt_t cert,
                                                 int indx,
                                                 void *oid,
                                                 size_t *sizeof_oid,
                                                 int *critical);
int                 gnutls_x509_crt_get_extension_data  (gnutls_x509_crt_t cert,
                                                 int indx,
                                                 void *data,
                                                 size_t *sizeof_data);
int                 gnutls_x509_crt_set_extension_by_oid
                                                (gnutls_x509_crt_t crt,
                                                 const char *oid,
                                                 const void *buf,
```

```
                                                 size_t sizeof_buf,
                                                 unsigned int critical);
int                 gnutls_x509_crt_set_dn_by_oid       (gnutls_x509_crt_t crt,
                                                 const char *oid,
                                                 unsigned int raw_flag,
                                                 const void *name,
                                                 unsigned int sizeof_name);
int                 gnutls_x509_crt_set_issuer_dn_by_oid
                                                (gnutls_x509_crt_t crt,
                                                 const char *oid,
                                                 unsigned int raw_flag,
                                                 const void *name,
                                                 unsigned int sizeof_name);
int                 gnutls_x509_crt_set_version         (gnutls_x509_crt_t crt,
                                                 unsigned int version);
int                 gnutls_x509_crt_set_key             (gnutls_x509_crt_t crt,
                                                 gnutls_x509_privkey_t key);
int                 gnutls_x509_crt_set_ca_status       (gnutls_x509_crt_t crt,
                                                 unsigned int ca);
int                 gnutls_x509_crt_set_basic_constraints
                                                (gnutls_x509_crt_t crt,
                                                 unsigned int ca,
                                                 int pathLenConstraint);
int                 gnutls_x509_crt_set_subject_alternative_name
                                                (gnutls_x509_crt_t crt,
                                                 gnutls_x509_subject_alt_name_t ty
                                                 const char *data_string);
int                 gnutls_x509_crt_set_subject_alt_name
                                                (gnutls_x509_crt_t crt,
                                                 gnutls_x509_subject_alt_name_t ty
                                                 const void *data,
                                                 unsigned int data_size,
                                                 unsigned int flags);
int                 gnutls_x509_crt_sign                (gnutls_x509_crt_t crt,
                                                 gnutls_x509_crt_t issuer,
                                                 gnutls_x509_privkey_t issuer_key)
int                 gnutls_x509_crt_sign2               (gnutls_x509_crt_t crt,
                                                 gnutls_x509_crt_t issuer,
                                                 gnutls_x509_privkey_t issuer_key,
                                                 gnutls_digest_algorithm_t dig,
                                                 unsigned int flags);
int                 gnutls_x509_crt_set_activation_time (gnutls_x509_crt_t cert,
                                                 time_t act_time);
int                 gnutls_x509_crt_set_expiration_time (gnutls_x509_crt_t cert,
                                                 time_t exp_time);
int                 gnutls_x509_crt_set_serial          (gnutls_x509_crt_t cert,
                                                 const void *serial,
                                                 size_t serial_size);
int                 gnutls_x509_crt_set_subject_key_id  (gnutls_x509_crt_t cert,
                                                 const void *id,
                                                 size_t id_size);
int                 gnutls_x509_crt_set_proxy_dn        (gnutls_x509_crt_t crt,
                                                 gnutls_x509_crt_t eecrt,
                                                 unsigned int raw_flag,
                                                 const void *name,
                                                 unsigned int sizeof_name);
int                 gnutls_x509_crt_set_proxy           (gnutls_x509_crt_t crt,
```

```
                                                    int pathLenConstraint,
                                                    const char *policyLanguage,
                                                    const char *policy,
                                                    size_t sizeof_policy);
int                 gnutls_x509_crt_print           (gnutls_x509_crt_t cert,
                                                    gnutls_certificate_print_formats_
                                                    gnutls_datum_t *out);
int                 gnutls_x509_crl_print           (gnutls_x509_crl_t crl,
                                                    gnutls_certificate_print_formats_
                                                    gnutls_datum_t *out);
int                 gnutls_x509_crt_get_raw_issuer_dn (gnutls_x509_crt_t cert,
                                                    gnutls_datum_t *start);
int                 gnutls_x509_crt_get_raw_dn      (gnutls_x509_crt_t cert,
                                                    gnutls_datum_t *start);
int                 gnutls_x509_rdn_get             (const gnutls_datum_t *idn,
                                                    char *buf,
                                                    size_t *sizeof_buf);
int                 gnutls_x509_rdn_get_oid         (const gnutls_datum_t *idn,
                                                    int indx,
                                                    void *buf,
                                                    size_t *sizeof_buf);
int                 gnutls_x509_rdn_get_by_oid      (const gnutls_datum_t *idn,
                                                    const char *oid,
                                                    int indx,
                                                    unsigned int raw_flag,
                                                    void *buf,
                                                    size_t *sizeof_buf);
typedef             gnutls_x509_dn_t;
int                 gnutls_x509_crt_get_subject     (gnutls_x509_crt_t cert,
                                                    gnutls_x509_dn_t *dn);
int                 gnutls_x509_crt_get_issuer      (gnutls_x509_crt_t cert,
                                                    gnutls_x509_dn_t *dn);
int                 gnutls_x509_dn_get_rdn_ava      (gnutls_x509_dn_t dn,
                                                    int irdn,
                                                    int iava,
                                                    gnutls_x509_ava_st *ava);
int                 gnutls_x509_dn_init             (gnutls_x509_dn_t *dn);
int                 gnutls_x509_dn_import           (gnutls_x509_dn_t dn,
                                                    const gnutls_datum_t *data);
int                 gnutls_x509_dn_export           (gnutls_x509_dn_t dn,
                                                    gnutls_x509_crt_fmt_t format,
                                                    void *output_data,
                                                    size_t *output_data_size);
void                gnutls_x509_dn_deinit           (gnutls_x509_dn_t dn);
int                 gnutls_x509_crl_init            (gnutls_x509_crl_t *crl);
void                gnutls_x509_crl_deinit          (gnutls_x509_crl_t crl);
int                 gnutls_x509_crl_import          (gnutls_x509_crl_t crl,
                                                    const gnutls_datum_t *data,
                                                    gnutls_x509_crt_fmt_t format);
int                 gnutls_x509_crl_export          (gnutls_x509_crl_t crl,
                                                    gnutls_x509_crt_fmt_t format,
                                                    void *output_data,
                                                    size_t *output_data_size);
int                 gnutls_x509_crl_get_issuer_dn   (const gnutls_x509_crl_t crl,
                                                    char *buf,
                                                    size_t *sizeof_buf);
int                 gnutls_x509_crl_get_issuer_dn_by_oid
```

```
                                                    (gnutls_x509_crl_t crl,
                                                     const char *oid,
                                                     int indx,
                                                     unsigned int raw_flag,
                                                     void *buf,
                                                     size_t *sizeof_buf);
int                 gnutls_x509_crl_get_dn_oid      (gnutls_x509_crl_t crl,
                                                     int indx,
                                                     void *oid,
                                                     size_t *sizeof_oid);
int                 gnutls_x509_crl_get_signature_algorithm
                                                    (gnutls_x509_crl_t crl);
int                 gnutls_x509_crl_get_signature   (gnutls_x509_crl_t crl,
                                                     char *sig,
                                                     size_t *sizeof_sig);
int                 gnutls_x509_crl_get_version     (gnutls_x509_crl_t crl);
time_t              gnutls_x509_crl_get_this_update (gnutls_x509_crl_t crl);
time_t              gnutls_x509_crl_get_next_update (gnutls_x509_crl_t crl);
int                 gnutls_x509_crl_get_crt_count   (gnutls_x509_crl_t crl);
int                 gnutls_x509_crl_get_crt_serial  (gnutls_x509_crl_t crl,
                                                     int indx,
                                                     unsigned char *serial,
                                                     size_t *serial_size,
                                                     time_t *t);
#define             gnutls_x509_crl_get_certificate_count
#define             gnutls_x509_crl_get_certificate
int                 gnutls_x509_crl_check_issuer    (gnutls_x509_crl_t crl,
                                                     gnutls_x509_crt_t issuer);
int                 gnutls_x509_crl_set_version     (gnutls_x509_crl_t crl,
                                                     unsigned int version);
int                 gnutls_x509_crl_sign            (gnutls_x509_crl_t crl,
                                                     gnutls_x509_crt_t issuer,
                                                     gnutls_x509_privkey_t issuer_key)
int                 gnutls_x509_crl_sign2           (gnutls_x509_crl_t crl,
                                                     gnutls_x509_crt_t issuer,
                                                     gnutls_x509_privkey_t issuer_key,
                                                     gnutls_digest_algorithm_t dig,
                                                     unsigned int flags);
int                 gnutls_x509_crl_set_this_update (gnutls_x509_crl_t crl,
                                                     time_t act_time);
int                 gnutls_x509_crl_set_next_update (gnutls_x509_crl_t crl,
                                                     time_t exp_time);
int                 gnutls_x509_crl_set_crt_serial  (gnutls_x509_crl_t crl,
                                                     const void *serial,
                                                     size_t serial_size,
                                                     time_t revocation_time);
int                 gnutls_x509_crl_set_crt         (gnutls_x509_crl_t crl,
                                                     gnutls_x509_crt_t crt,
                                                     time_t revocation_time);
int                 gnutls_x509_crl_get_authority_key_id
                                                    (gnutls_x509_crl_t crl,
                                                     void *ret,
                                                     size_t *ret_size,
                                                     unsigned int *critical);
int                 gnutls_x509_crl_get_number      (gnutls_x509_crl_t crl,
                                                     void *ret,
                                                     size_t *ret_size,
```

```
                                                 unsigned int *critical);
int                 gnutls_x509_crl_get_extension_oid   (gnutls_x509_crl_t crl,
                                                 int indx,
                                                 void *oid,
                                                 size_t *sizeof_oid);
int                 gnutls_x509_crl_get_extension_info  (gnutls_x509_crl_t crl,
                                                 int indx,
                                                 void *oid,
                                                 size_t *sizeof_oid,
                                                 int *critical);
int                 gnutls_x509_crl_get_extension_data  (gnutls_x509_crl_t crl,
                                                 int indx,
                                                 void *data,
                                                 size_t *sizeof_data);
int                 gnutls_x509_crl_set_authority_key_id
                                                (gnutls_x509_crl_t crl,
                                                 const void *id,
                                                 size_t id_size);
int                 gnutls_x509_crl_set_number          (gnutls_x509_crl_t crl,
                                                 const void *nr,
                                                 size_t nr_size);
struct              gnutls_pkcs7_int;
typedef             gnutls_pkcs7_t;
int                 gnutls_pkcs7_init                   (gnutls_pkcs7_t *pkcs7);
void                gnutls_pkcs7_deinit                 (gnutls_pkcs7_t pkcs7);
int                 gnutls_pkcs7_import                 (gnutls_pkcs7_t pkcs7,
                                                 const gnutls_datum_t *data,
                                                 gnutls_x509_crt_fmt_t format);
int                 gnutls_pkcs7_export                 (gnutls_pkcs7_t pkcs7,
                                                 gnutls_x509_crt_fmt_t format,
                                                 void *output_data,
                                                 size_t *output_data_size);
int                 gnutls_pkcs7_get_crt_count          (gnutls_pkcs7_t pkcs7);
int                 gnutls_pkcs7_get_crt_raw            (gnutls_pkcs7_t pkcs7,
                                                 int indx,
                                                 void *certificate,
                                                 size_t *certificate_size);
int                 gnutls_pkcs7_set_crt_raw            (gnutls_pkcs7_t pkcs7,
                                                 const gnutls_datum_t *crt);
int                 gnutls_pkcs7_set_crt                (gnutls_pkcs7_t pkcs7,
                                                 gnutls_x509_crt_t crt);
int                 gnutls_pkcs7_delete_crt             (gnutls_pkcs7_t pkcs7,
                                                 int indx);
int                 gnutls_pkcs7_get_crl_raw            (gnutls_pkcs7_t pkcs7,
                                                 int indx,
                                                 void *crl,
                                                 size_t *crl_size);
int                 gnutls_pkcs7_get_crl_count          (gnutls_pkcs7_t pkcs7);
int                 gnutls_pkcs7_set_crl_raw            (gnutls_pkcs7_t pkcs7,
                                                 const gnutls_datum_t *crl);
int                 gnutls_pkcs7_set_crl                (gnutls_pkcs7_t pkcs7,
                                                 gnutls_x509_crl_t crl);
int                 gnutls_pkcs7_delete_crl             (gnutls_pkcs7_t pkcs7,
                                                 int indx);
enum                gnutls_certificate_verify_flags;
int                 gnutls_x509_crt_check_issuer        (gnutls_x509_crt_t cert,
                                                 gnutls_x509_crt_t issuer);
```

```
int                 gnutls_x509_crt_list_verify         (const gnutls_x509_crt_t *cert_lis
                                                         int cert_list_length,
                                                         const gnutls_x509_crt_t *CA_list,
                                                         int CA_list_length,
                                                         const gnutls_x509_crl_t *CRL_list
                                                         int CRL_list_length,
                                                         unsigned int flags,
                                                         unsigned int *verify);
int                 gnutls_x509_crt_verify              (gnutls_x509_crt_t cert,
                                                         const gnutls_x509_crt_t *CA_list,
                                                         int CA_list_length,
                                                         unsigned int flags,
                                                         unsigned int *verify);
int                 gnutls_x509_crl_verify              (gnutls_x509_crl_t crl,
                                                         const gnutls_x509_crt_t *CA_list,
                                                         int CA_list_length,
                                                         unsigned int flags,
                                                         unsigned int *verify);
int                 gnutls_x509_crt_check_revocation    (gnutls_x509_crt_t cert,
                                                         const gnutls_x509_crl_t *crl_list
                                                         int crl_list_length);
int                 gnutls_x509_crt_get_fingerprint     (gnutls_x509_crt_t cert,
                                                         gnutls_digest_algorithm_t algo,
                                                         void *buf,
                                                         size_t *sizeof_buf);
int                 gnutls_x509_crt_get_key_purpose_oid (gnutls_x509_crt_t cert,
                                                         int indx,
                                                         void *oid,
                                                         size_t *sizeof_oid,
                                                         unsigned int *critical);
int                 gnutls_x509_crt_set_key_purpose_oid (gnutls_x509_crt_t cert,
                                                         const void *oid,
                                                         unsigned int critical);
enum                gnutls_pkcs_encrypt_flags_t;
int                 gnutls_x509_privkey_init            (gnutls_x509_privkey_t *key);
void                gnutls_x509_privkey_deinit          (gnutls_x509_privkey_t key);
int                 gnutls_x509_privkey_cpy             (gnutls_x509_privkey_t dst,
                                                         gnutls_x509_privkey_t src);
int                 gnutls_x509_privkey_import          (gnutls_x509_privkey_t key,
                                                         const gnutls_datum_t *data,
                                                         gnutls_x509_crt_fmt_t format);
int                 gnutls_x509_privkey_import_pkcs8    (gnutls_x509_privkey_t key,
                                                         const gnutls_datum_t *data,
                                                         gnutls_x509_crt_fmt_t format,
                                                         const char *password,
                                                         unsigned int flags);
int                 gnutls_x509_privkey_import_rsa_raw  (gnutls_x509_privkey_t key,
                                                         const gnutls_datum_t *m,
                                                         const gnutls_datum_t *e,
                                                         const gnutls_datum_t *d,
                                                         const gnutls_datum_t *p,
                                                         const gnutls_datum_t *q,
                                                         const gnutls_datum_t *u);
int                 gnutls_x509_privkey_fix             (gnutls_x509_privkey_t key);
int                 gnutls_x509_privkey_export_dsa_raw  (gnutls_x509_privkey_t key,
                                                         gnutls_datum_t *p,
                                                         gnutls_datum_t *q,
```

```
                                                gnutls_datum_t *g,
                                                gnutls_datum_t *y,
                                                gnutls_datum_t *x);
int                 gnutls_x509_privkey_import_dsa_raw  (gnutls_x509_privkey_t key,
                                                const gnutls_datum_t *p,
                                                const gnutls_datum_t *q,
                                                const gnutls_datum_t *g,
                                                const gnutls_datum_t *y,
                                                const gnutls_datum_t *x);
int                 gnutls_x509_privkey_get_pk_algorithm
                                               (gnutls_x509_privkey_t key);
int                 gnutls_x509_privkey_get_key_id      (gnutls_x509_privkey_t key,
                                                unsigned int flags,
                                                unsigned char *output_data,
                                                size_t *output_data_size);
int                 gnutls_x509_privkey_generate        (gnutls_x509_privkey_t key,
                                                gnutls_pk_algorithm_t algo,
                                                unsigned int bits,
                                                unsigned int flags);
int                 gnutls_x509_privkey_export          (gnutls_x509_privkey_t key,
                                                gnutls_x509_crt_fmt_t format,
                                                void *output_data,
                                                size_t *output_data_size);
int                 gnutls_x509_privkey_export_pkcs8    (gnutls_x509_privkey_t key,
                                                gnutls_x509_crt_fmt_t format,
                                                const char *password,
                                                unsigned int flags,
                                                void *output_data,
                                                size_t *output_data_size);
int                 gnutls_x509_privkey_export_rsa_raw  (gnutls_x509_privkey_t key,
                                                gnutls_datum_t *m,
                                                gnutls_datum_t *e,
                                                gnutls_datum_t *d,
                                                gnutls_datum_t *p,
                                                gnutls_datum_t *q,
                                                gnutls_datum_t *u);
int                 gnutls_x509_privkey_sign_data       (gnutls_x509_privkey_t key,
                                                gnutls_digest_algorithm_t digest,
                                                unsigned int flags,
                                                const gnutls_datum_t *data,
                                                void *signature,
                                                size_t *signature_size);
int                 gnutls_x509_privkey_verify_data     (gnutls_x509_privkey_t key,
                                                unsigned int flags,
                                                const gnutls_datum_t *data,
                                                const gnutls_datum_t *signature);
int                 gnutls_x509_crt_verify_data         (gnutls_x509_crt_t crt,
                                                unsigned int flags,
                                                const gnutls_datum_t *data,
                                                const gnutls_datum_t *signature);
int                 gnutls_x509_crt_verify_hash         (gnutls_x509_crt_t crt,
                                                unsigned int flags,
                                                const gnutls_datum_t *hash,
                                                const gnutls_datum_t *signature);
int                 gnutls_x509_crt_get_verify_algorithm
                                               (gnutls_x509_crt_t crt,
                                                const gnutls_datum_t *signature,
```

```
                                                gnutls_digest_algorithm_t *hash);
int                 gnutls_x509_privkey_sign_hash
                                               (gnutls_x509_privkey_t key,
                                                const gnutls_datum_t *hash,
                                                gnutls_datum_t *signature);
struct              gnutls_x509_crq_int;
typedef             gnutls_x509_crq_t;
int                 gnutls_x509_crq_print      (gnutls_x509_crq_t crq,
                                                gnutls_certificate_print_formats_
                                                gnutls_datum_t *out);
int                 gnutls_x509_crq_init       (gnutls_x509_crq_t *crq);
void                gnutls_x509_crq_deinit     (gnutls_x509_crq_t crq);
int                 gnutls_x509_crq_import     (gnutls_x509_crq_t crq,
                                                const gnutls_datum_t *data,
                                                gnutls_x509_crt_fmt_t format);
int                 gnutls_x509_crq_get_dn     (gnutls_x509_crq_t crq,
                                                char *buf,
                                                size_t *sizeof_buf);
int                 gnutls_x509_crq_get_dn_oid (gnutls_x509_crq_t crq,
                                                int indx,
                                                void *oid,
                                                size_t *sizeof_oid);
int                 gnutls_x509_crq_get_dn_by_oid
                                               (gnutls_x509_crq_t crq,
                                                const char *oid,
                                                int indx,
                                                unsigned int raw_flag,
                                                void *buf,
                                                size_t *sizeof_buf);
int                 gnutls_x509_crq_set_dn_by_oid
                                               (gnutls_x509_crq_t crq,
                                                const char *oid,
                                                unsigned int raw_flag,
                                                const void *data,
                                                unsigned int sizeof_data);
int                 gnutls_x509_crq_set_version
                                               (gnutls_x509_crq_t crq,
                                                unsigned int version);
int                 gnutls_x509_crq_get_version
                                               (gnutls_x509_crq_t crq);
int                 gnutls_x509_crq_set_key    (gnutls_x509_crq_t crq,
                                                gnutls_x509_privkey_t key);
int                 gnutls_x509_crq_sign2      (gnutls_x509_crq_t crq,
                                                gnutls_x509_privkey_t key,
                                                gnutls_digest_algorithm_t dig,
                                                unsigned int flags);
int                 gnutls_x509_crq_sign       (gnutls_x509_crq_t crq,
                                                gnutls_x509_privkey_t key);
int                 gnutls_x509_crq_set_challenge_password
                                               (gnutls_x509_crq_t crq,
                                                const char *pass);
int                 gnutls_x509_crq_get_challenge_password
                                               (gnutls_x509_crq_t crq,
                                                char *pass,
                                                size_t *sizeof_pass);
int                 gnutls_x509_crq_set_attribute_by_oid
                                               (gnutls_x509_crq_t crq,
                                                const char *oid,
                                                void *buf,
                                                size_t sizeof_buf);
int                 gnutls_x509_crq_get_attribute_by_oid
                                               (gnutls_x509_crq_t crq,
```

```
                                                const char *oid,
                                                int indx,
                                                void *buf,
                                                size_t *sizeof_buf);
int                 gnutls_x509_crq_export      (gnutls_x509_crq_t crq,
                                                gnutls_x509_crt_fmt_t format,
                                                void *output_data,
                                                size_t *output_data_size);
int                 gnutls_x509_crt_set_crq     (gnutls_x509_crt_t crt,
                                                gnutls_x509_crq_t crq);
int                 gnutls_x509_crt_set_crq_extensions
                                                (gnutls_x509_crt_t crt,
                                                gnutls_x509_crq_t crq);
int                 gnutls_x509_crq_set_key_rsa_raw
                                                (gnutls_x509_crq_t crq,
                                                const gnutls_datum_t *m,
                                                const gnutls_datum_t *e);
int                 gnutls_x509_crq_set_subject_alt_name
                                                (gnutls_x509_crq_t crq,
                                                gnutls_x509_subject_alt_name_t nt
                                                const void *data,
                                                unsigned int data_size,
                                                unsigned int flags);
int                 gnutls_x509_crq_set_key_usage
                                                (gnutls_x509_crq_t crq,
                                                unsigned int usage);
int                 gnutls_x509_crq_set_basic_constraints
                                                (gnutls_x509_crq_t crq,
                                                unsigned int ca,
                                                int pathLenConstraint);
int                 gnutls_x509_crq_set_key_purpose_oid
                                                (gnutls_x509_crq_t crq,
                                                const void *oid,
                                                unsigned int critical);
int                 gnutls_x509_crq_get_key_purpose_oid
                                                (gnutls_x509_crq_t crq,
                                                int indx,
                                                void *oid,
                                                size_t *sizeof_oid,
                                                unsigned int *critical);
int                 gnutls_x509_crq_get_extension_data
                                                (gnutls_x509_crq_t crq,
                                                int indx,
                                                void *data,
                                                size_t *sizeof_data);
int                 gnutls_x509_crq_get_extension_info
                                                (gnutls_x509_crq_t crq,
                                                int indx,
                                                void *oid,
                                                size_t *sizeof_oid,
                                                int *critical);
int                 gnutls_x509_crq_get_attribute_data
                                                (gnutls_x509_crq_t crq,
                                                int indx,
                                                void *data,
                                                size_t *sizeof_data);
int                 gnutls_x509_crq_get_attribute_info
                                                (gnutls_x509_crq_t crq,
                                                int indx,
                                                void *oid,
                                                size_t *sizeof_oid);
int                 gnutls_x509_crq_get_pk_algorithm
                                                (gnutls_x509_crq_t crq,
                                                unsigned int *bits);
int                 gnutls_x509_crq_get_key_id  (gnutls_x509_crq_t crq,
                                                unsigned int flags,
                                                unsigned char *output_data,
```

```
                                                     size_t *output_data_size);
int                 gnutls_x509_crq_get_key_rsa_raw     (gnutls_x509_crq_t crq,
                                                         gnutls_datum_t *m,
                                                         gnutls_datum_t *e);
int                 gnutls_x509_crq_get_key_usage       (gnutls_x509_crq_t crq,
                                                         unsigned int *key_usage,
                                                         unsigned int *critical);
int                 gnutls_x509_crq_get_basic_constraints
                                                        (gnutls_x509_crq_t crq,
                                                         unsigned int *critical,
                                                         int *ca,
                                                         int *pathlen);
int                 gnutls_x509_crq_get_subject_alt_name
                                                        (gnutls_x509_crq_t crq,
                                                         unsigned int seq,
                                                         void *ret,
                                                         size_t *ret_size,
                                                         unsigned int *ret_type,
                                                         unsigned int *critical);
int                 gnutls_x509_crq_get_subject_alt_othername_oid
                                                        (gnutls_x509_crq_t crq,
                                                         unsigned int seq,
                                                         void *ret,
                                                         size_t *ret_size);
int                 gnutls_x509_crq_get_extension_by_oid
                                                        (gnutls_x509_crq_t crq,
                                                         const char *oid,
                                                         int indx,
                                                         void *buf,
                                                         size_t *sizeof_buf,
                                                         unsigned int *critical);
```

Description

Details

GNUTLS_OID_X520_COUNTRY_NAME

```
#define GNUTLS_OID_X520_COUNTRY_NAME    "2.5.4.6"
```

GNUTLS_OID_X520_ORGANIZATION_NAME

```
#define GNUTLS_OID_X520_ORGANIZATION_NAME~"2.5.4.10"
```

GNUTLS_OID_X520_ORGANIZATIONAL_UNIT_NAME

```
#define GNUTLS_OID_X520_ORGANIZATIONAL_UNIT_NAME "2.5.4.11"
```

GNUTLS_OID_X520_COMMON_NAME

```
#define GNUTLS_OID_X520_COMMON_NAME   "2.5.4.3"
```

GNUTLS_OID_X520_LOCALITY_NAME

```
#define GNUTLS_OID_X520_LOCALITY_NAME   "2.5.4.7"
```

GNUTLS_OID_X520_STATE_OR_PROVINCE_NAME

```
#define GNUTLS_OID_X520_STATE_OR_PROVINCE_NAME~"2.5.4.8"
```

GNUTLS_OID_X520_INITIALS

```
#define GNUTLS_OID_X520_INITIALS    "2.5.4.43"
```

GNUTLS_OID_X520_GENERATION_QUALIFIER

```
#define GNUTLS_OID_X520_GENERATION_QUALIFIER~"2.5.4.44"
```

GNUTLS_OID_X520_SURNAME

```
#define GNUTLS_OID_X520_SURNAME     "2.5.4.4"
```

GNUTLS_OID_X520_GIVEN_NAME

```
#define GNUTLS_OID_X520_GIVEN_NAME    "2.5.4.42"
```

GNUTLS_OID_X520_TITLE

```
#define GNUTLS_OID_X520_TITLE     "2.5.4.12"
```

GNUTLS_OID_X520_DN_QUALIFIER

```
#define GNUTLS_OID_X520_DN_QUALIFIER    "2.5.4.46"
```

GNUTLS_OID_X520_PSEUDONYM

```
#define GNUTLS_OID_X520_PSEUDONYM   "2.5.4.65"
```

GNUTLS_OID_X520_POSTALCODE

```
#define GNUTLS_OID_X520_POSTALCODE              "2.5.4.17"
```

GNUTLS_OID_X520_NAME

```
#define GNUTLS_OID_X520_NAME                    "2.5.4.41"
```

GNUTLS_OID_LDAP_DC

```
#define GNUTLS_OID_LDAP_DC      "0.9.2342.19200300.100.1.25"
```

GNUTLS_OID_LDAP_UID

```
#define GNUTLS_OID_LDAP_UID     "0.9.2342.19200300.100.1.1"
```

GNUTLS_OID_PKCS9_EMAIL

```
#define GNUTLS_OID_PKCS9_EMAIL      "1.2.840.113549.1.9.1"
```

GNUTLS_OID_PKIX_DATE_OF_BIRTH

```
#define GNUTLS_OID_PKIX_DATE_OF_BIRTH   "1.3.6.1.5.5.7.9.1"
```

GNUTLS_OID_PKIX_PLACE_OF_BIRTH

```
#define GNUTLS_OID_PKIX_PLACE_OF_BIRTH    "1.3.6.1.5.5.7.9.2"
```

GNUTLS_OID_PKIX_GENDER

```
#define GNUTLS_OID_PKIX_GENDER      "1.3.6.1.5.5.7.9.3"
```

GNUTLS_OID_PKIX_COUNTRY_OF_CITIZENSHIP

```
#define GNUTLS_OID_PKIX_COUNTRY_OF_CITIZENSHIP~"1.3.6.1.5.5.7.9.4"
```

GNUTLS_OID_PKIX_COUNTRY_OF_RESIDENCE

```
#define GNUTLS_OID_PKIX_COUNTRY_OF_RESIDENCE~"1.3.6.1.5.5.7.9.5"
```

GNUTLS_KP_TLS_WWW_SERVER

```
#define GNUTLS_KP_TLS_WWW_SERVER    "1.3.6.1.5.5.7.3.1"
```

GNUTLS_KP_TLS_WWW_CLIENT

```
#define GNUTLS_KP_TLS_WWW_CLIENT                "1.3.6.1.5.5.7.3.2"
```

GNUTLS_KP_CODE_SIGNING

```
#define GNUTLS_KP_CODE_SIGNING      "1.3.6.1.5.5.7.3.3"
```

GNUTLS_KP_EMAIL_PROTECTION

```
#define GNUTLS_KP_EMAIL_PROTECTION    "1.3.6.1.5.5.7.3.4"
```

GNUTLS_KP_TIME_STAMPING

```
#define GNUTLS_KP_TIME_STAMPING     "1.3.6.1.5.5.7.3.8"
```

GNUTLS_KP_OCSP_SIGNING

```
#define GNUTLS_KP_OCSP_SIGNING      "1.3.6.1.5.5.7.3.9"
```

GNUTLS_KP_ANY

```
#define GNUTLS_KP_ANY       "2.5.29.37.0"
```

GNUTLS_FSAN_SET

```
#define GNUTLS_FSAN_SET 0
```

GNUTLS_FSAN_APPEND

```
#define GNUTLS_FSAN_APPEND 1
```

enum gnutls_certificate_import_flags

```
typedef enum gnutls_certificate_import_flags
{
  GNUTLS_X509_CRT_LIST_IMPORT_FAIL_IF_EXCEED = 1
} gnutls_certificate_import_flags;
```

Enumeration of different certificate import flags.

GNUTLS_X509_CRT_LIST_IMPORT_FAIL_IF_EXCEED Fail if the certificates in the buffer are more than the space allocated for certificates. The error code will be GNUTLS_E_SHORT_MEMORY_BUFFER.

gnutls_x509_crt_init ()

```
int                 gnutls_x509_crt_init                (gnutls_x509_crt_t *cert);
```

This function will initialize an X.509 certificate structure.

cert : The structure to be initialized

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_x509_crt_deinit ()

```
void                gnutls_x509_crt_deinit              (gnutls_x509_crt_t cert);
```

This function will deinitialize a CRL structure.

cert : The structure to be initialized

gnutls_x509_crt_import ()

```
int                 gnutls_x509_crt_import              (gnutls_x509_crt_t cert,
                                                         const gnutls_datum_t *data,
                                                         gnutls_x509_crt_fmt_t format);
```

This function will convert the given DER or PEM encoded Certificate to the native gnutls_x509_crt_t format. The output will be stored in *cert*.

If the Certificate is PEM encoded it should have a header of "X509 CERTIFICATE", or "CERTIFICATE".

cert : The structure to store the parsed certificate.

data : The DER or PEM encoded certificate.

format : One of DER or PEM

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_x509_crt_list_import ()

```
int                 gnutls_x509_crt_list_import         (gnutls_x509_crt_t *certs,
                                                         unsigned int *cert_max,
                                                         const gnutls_datum_t *data,
                                                         gnutls_x509_crt_fmt_t format,
                                                         unsigned int flags);
```

This function will convert the given PEM encoded certificate list to the native gnutls_x509_crt_t format. The output will be stored in *certs*. They will be automatically initialized.

If the Certificate is PEM encoded it should have a header of "X509 CERTIFICATE", or "CERTIFICATE".

certs : The structures to store the parsed certificate. Must not be initialized.

cert_max : Initially must hold the maximum number of certs. It will be updated with the number of certs available.

data : The PEM encoded certificate.

format : One of DER or PEM.

flags : must be zero or an OR'd sequence of gnutls_certificate_import_flags.

Returns : the number of certificates read or a negative error value.

gnutls_x509_crt_export ()

```
int                 gnutls_x509_crt_export              (gnutls_x509_crt_t cert,
                                                         gnutls_x509_crt_fmt_t format,
                                                         void *output_data,
                                                         size_t *output_data_size);
```

This function will export the certificate to DER or PEM format.

If the buffer provided is not long enough to hold the output, then *output_data_size is updated and GNUTLS_E_SHORT_MEMORY_BU will be returned.

If the structure is PEM encoded, it will have a header of "BEGIN CERTIFICATE".

cert **:** Holds the certificate

format **:** the format of output params. One of PEM or DER.

output_data **:** will contain a certificate PEM or DER encoded

output_data_size **:** holds the size of output_data (and will be replaced by the actual size of parameters)

Returns **:** In case of failure a negative value will be returned, and 0 on success.

gnutls_x509_crt_get_issuer_dn ()

```
int                 gnutls_x509_crt_get_issuer_dn       (gnutls_x509_crt_t cert,
                                                         char *buf,
                                                         size_t *sizeof_buf);
```

This function will copy the name of the Certificate issuer in the provided buffer. The name will be in the form "C=xxxx,O=yyyy,CN=zzzz as described in RFC2253. The output string will be ASCII or UTF-8 encoded, depending on the certificate data.

If *buf* is null then only the size will be filled.

cert **:** should contain a gnutls_x509_crt_t structure

buf **:** a pointer to a structure to hold the name (may be null)

sizeof_buf **:** initially holds the size of *buf*

Returns **:** GNUTLS_E_SHORT_MEMORY_BUFFER if the provided buffer is not long enough, and in that case the *sizeof_buf will be updated with the required size. On success 0 is returned.

gnutls_x509_crt_get_issuer_dn_oid ()

```
int                 gnutls_x509_crt_get_issuer_dn_oid   (gnutls_x509_crt_t cert,
                                                         int indx,
                                                         void *oid,
                                                         size_t *sizeof_oid);
```

This function will extract the OIDs of the name of the Certificate issuer specified by the given index.

If *oid* is null then only the size will be filled.

cert **:** should contain a gnutls_x509_crt_t structure

indx **:** This specifies which OID to return. Use zero to get the first one.

oid **:** a pointer to a buffer to hold the OID (may be null)

sizeof_oid **:** initially holds the size of *oid*

Returns **:** GNUTLS_E_SHORT_MEMORY_BUFFER if the provided buffer is not long enough, and in that case the *sizeof_oid will be updated with the required size. On success 0 is returned.

gnutls_x509_crt_get_issuer_dn_by_oid ()

```
int                 gnutls_x509_crt_get_issuer_dn_by_oid
                                                    (gnutls_x509_crt_t cert,
                                                     const char *oid,
                                                     int indx,
                                                     unsigned int raw_flag,
                                                     void *buf,
                                                     size_t *sizeof_buf);
```

This function will extract the part of the name of the Certificate issuer specified by the given OID. The output, if the raw flag is not used, will be encoded as described in RFC2253. Thus a string that is ASCII or UTF-8 encoded, depending on the certificate data.

Some helper macros with popular OIDs can be found in gnutls/x509.h If raw flag is zero, this function will only return known OIDs as text. Other OIDs will be DER encoded, as described in RFC2253 -- in hex format with a '#' prefix. You can check about known OIDs using gnutls_x509_dn_oid_known().

If *buf* is null then only the size will be filled.

cert : should contain a gnutls_x509_crt_t structure

oid : holds an Object Identified in null terminated string

indx : In case multiple same OIDs exist in the RDN, this specifies which to send. Use zero to get the first one.

raw_flag : If non zero returns the raw DER data of the DN part.

buf : a pointer to a structure to hold the name (may be null)

sizeof_buf : initially holds the size of *buf*

Returns : GNUTLS_E_SHORT_MEMORY_BUFFER if the provided buffer is not long enough, and in that case the *sizeof_buf will be updated with the required size. On success 0 is returned.

gnutls_x509_crt_get_dn ()

```
int                 gnutls_x509_crt_get_dn              (gnutls_x509_crt_t cert,
                                                         char *buf,
                                                         size_t *sizeof_buf);
```

This function will copy the name of the Certificate in the provided buffer. The name will be in the form "C=xxxx,O=yyyy,CN=zzzz" as described in RFC2253. The output string will be ASCII or UTF-8 encoded, depending on the certificate data.

If *buf* is null then only the size will be filled.

cert : should contain a gnutls_x509_crt_t structure

buf : a pointer to a structure to hold the name (may be null)

sizeof_buf : initially holds the size of *buf*

Returns : GNUTLS_E_SHORT_MEMORY_BUFFER if the provided buffer is not long enough, and in that case the *sizeof_buf will be updated with the required size. On success 0 is returned.

gnutls_x509_crt_get_dn_oid ()

```
int                 gnutls_x509_crt_get_dn_oid          (gnutls_x509_crt_t cert,
                                                         int indx,
                                                         void *oid,
                                                         size_t *sizeof_oid);
```

This function will extract the OIDs of the name of the Certificate subject specified by the given index.

If oid is null then only the size will be filled.

cert : should contain a gnutls_x509_crt_t structure

indx : This specifies which OID to return. Use zero to get the first one.

oid : a pointer to a buffer to hold the OID (may be null)

sizeof_oid : initially holds the size of *oid*

Returns : GNUTLS_E_SHORT_MEMORY_BUFFER if the provided buffer is not long enough, and in that case the *sizeof_oid will be updated with the required size. On success 0 is returned.

gnutls_x509_crt_get_dn_by_oid ()

```
int                 gnutls_x509_crt_get_dn_by_oid       (gnutls_x509_crt_t cert,
                                                         const char *oid,
                                                         int indx,
                                                         unsigned int raw_flag,
                                                         void *buf,
                                                         size_t *sizeof_buf);
```

This function will extract the part of the name of the Certificate subject specified by the given OID. The output, if the raw flag is not used, will be encoded as described in RFC2253. Thus a string that is ASCII or UTF-8 encoded, depending on the certificate data.

Some helper macros with popular OIDs can be found in gnutls/x509.h If raw flag is zero, this function will only return known OIDs as text. Other OIDs will be DER encoded, as described in RFC2253 -- in hex format with a '#' prefix. You can check about known OIDs using gnutls_x509_dn_oid_known().

If *buf* is null then only the size will be filled.

cert : should contain a gnutls_x509_crt_t structure

oid : holds an Object Identified in null terminated string

indx : In case multiple same OIDs exist in the RDN, this specifies which to send. Use zero to get the first one.

raw_flag : If non zero returns the raw DER data of the DN part.

buf : a pointer where the DN part will be copied (may be null).

sizeof_buf : initially holds the size of *buf*

Returns : GNUTLS_E_SHORT_MEMORY_BUFFER if the provided buffer is not long enough, and in that case the *sizeof_buf will be updated with the required size. On success 0 is returned.

gnutls_x509_crt_check_hostname ()

```
int                 gnutls_x509_crt_check_hostname      (gnutls_x509_crt_t cert,
                                                         const char *hostname);
```

This function will check if the given certificate's subject matches the given hostname. This is a basic implementation of the matching described in RFC2818 (HTTPS), which takes into account wildcards, and the DNSName/IPAddress subject alternative name PKIX extension.

cert : should contain an gnutls_x509_crt_t structure

hostname : A null terminated string that contains a DNS name

Returns : non zero for a successful match, and zero on failure.

gnutls_x509_crt_get_signature_algorithm ()

```
int                 gnutls_x509_crt_get_signature_algorithm
                                                        (gnutls_x509_crt_t cert);
```

This function will return a value of the gnutls_sign_algorithm_t enumeration that is the signature algorithm.

cert : should contain a gnutls_x509_crt_t structure

Returns : a gnutls_sign_algorithm_t value, or a negative value on error.

gnutls_x509_crt_get_signature ()

```
int                 gnutls_x509_crt_get_signature       (gnutls_x509_crt_t cert,
                                                         char *sig,
                                                         size_t *sizeof_sig);
```

This function will extract the signature field of a certificate.

cert : should contain a gnutls_x509_crt_t structure

sig : a pointer where the signature part will be copied (may be null).

sizeof_sig : initially holds the size of *sig*

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value. and a negative value on error.

gnutls_x509_crt_get_version ()

```
int                 gnutls_x509_crt_get_version         (gnutls_x509_crt_t cert);
```

This function will return the version of the specified Certificate.

cert : should contain a gnutls_x509_crt_t structure

Returns : version of certificate, or a negative value on error.

gnutls_x509_crt_get_key_id ()

```
int                 gnutls_x509_crt_get_key_id          (gnutls_x509_crt_t crt,
                                                         unsigned int flags,
                                                         unsigned char *output_data,
                                                         size_t *output_data_size);
```

This function will return a unique ID the depends on the public key parameters. This ID can be used in checking whether a certificate corresponds to the given private key.

If the buffer provided is not long enough to hold the output, then *output_data_size is updated and GNUTLS_E_SHORT_MEMORY_BU will be returned. The output will normally be a SHA-1 hash output, which is 20 bytes.

crt : Holds the certificate

flags : should be 0 for now

output_data : will contain the key ID

output_data_size : holds the size of output_data (and will be replaced by the actual size of parameters)

Returns : In case of failure a negative value will be returned, and 0 on success.

gnutls_x509_crt_set_authority_key_id ()

```
int                 gnutls_x509_crt_set_authority_key_id
                                                        (gnutls_x509_crt_t cert,
                                                         const void *id,
                                                         size_t id_size);
```

This function will set the X.509 certificate's authority key ID extension. Only the keyIdentifier field can be set with this function.

cert : a certificate of type gnutls_x509_crt_t

id : The key ID

id_size : Holds the size of the serial field.

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_x509_crt_get_authority_key_id ()

```
int                 gnutls_x509_crt_get_authority_key_id
                                                        (gnutls_x509_crt_t cert,
                                                         void *ret,
                                                         size_t *ret_size,
                                                         unsigned int *critical);
```

This function will return the X.509v3 certificate authority's key identifier. This is obtained by the X.509 Authority Key identifier extension field (2.5.29.35). Note that this function only returns the keyIdentifier field of the extension.

cert : should contain a gnutls_x509_crt_t structure

ret : The place where the identifier will be copied

ret_size : Holds the size of the result field.

critical : will be non zero if the extension is marked as critical (may be null)

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_x509_crt_get_subject_key_id ()

```
int                 gnutls_x509_crt_get_subject_key_id  (gnutls_x509_crt_t cert,
                                                         void *ret,
                                                         size_t *ret_size,
                                                         unsigned int *critical);
```

This function will return the X.509v3 certificate's subject key identifier. This is obtained by the X.509 Subject Key identifier extension field (2.5.29.14).

cert : should contain a gnutls_x509_crt_t structure

ret : The place where the identifier will be copied

ret_size : Holds the size of the result field.

critical : will be non zero if the extension is marked as critical (may be null)

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

GNUTLS_CRL_REASON_UNUSED

```
#define GNUTLS_CRL_REASON_UNUSED 128
```

GNUTLS_CRL_REASON_KEY_COMPROMISE

```
#define GNUTLS_CRL_REASON_KEY_COMPROMISE 64
```

GNUTLS_CRL_REASON_CA_COMPROMISE

```
#define GNUTLS_CRL_REASON_CA_COMPROMISE 32
```

GNUTLS_CRL_REASON_AFFILIATION_CHANGED

```
#define GNUTLS_CRL_REASON_AFFILIATION_CHANGED 16
```

GNUTLS_CRL_REASON_SUPERSEDED

```
#define GNUTLS_CRL_REASON_SUPERSEDED 8
```

GNUTLS_CRL_REASON_SUPERSEEDED

```
#define GNUTLS_CRL_REASON_SUPERSEEDED GNUTLS_CRL_REASON_SUPERSEDED
```

GNUTLS_CRL_REASON_CESSATION_OF_OPERATION

```
#define GNUTLS_CRL_REASON_CESSATION_OF_OPERATION 4
```

GNUTLS_CRL_REASON_CERTIFICATE_HOLD

```
#define GNUTLS_CRL_REASON_CERTIFICATE_HOLD 2
```

GNUTLS_CRL_REASON_PRIVILEGE_WITHDRAWN

```
#define GNUTLS_CRL_REASON_PRIVILEGE_WITHDRAWN 1
```

GNUTLS_CRL_REASON_AA_COMPROMISE

```
#define GNUTLS_CRL_REASON_AA_COMPROMISE 32768
```

gnutls_x509_crt_get_crl_dist_points ()

```
int                 gnutls_x509_crt_get_crl_dist_points (gnutls_x509_crt_t cert,
                                                         unsigned int seq,
                                                         void *ret,
                                                         size_t *ret_size,
                                                         unsigned int *reason_flags,
                                                         unsigned int *critical);
```

This function retrieves the CRL distribution points (2.5.29.31), contained in the given certificate in the X509v3 Certificate Extensions.

reason_flags should be an ORed sequence of GNUTLS_CRL_REASON_UNUSED, GNUTLS_CRL_REASON_KEY_COMPROM GNUTLS_CRL_REASON_CA_COMPROMISE, GNUTLS_CRL_REASON_AFFILIATION_CHANGED, GNUTLS_CRL_REASON GNUTLS_CRL_REASON_CESSATION_OF_OPERATION, GNUTLS_CRL_REASON_CERTIFICATE_HOLD, GNUTLS_CRL_R GNUTLS_CRL_REASON_AA_COMPROMISE, or zero for all possible reasons.

cert **:** should contain a gnutls_x509_crt_t structure

seq **:** specifies the sequence number of the distribution point (0 for the first one, 1 for the second etc.)

ret **:** is the place where the distribution point will be copied to

ret_size **:** holds the size of ret.

reason_flags **:** Revocation reasons flags.

critical **:** will be non zero if the extension is marked as critical (may be null)

Returns **:** GNUTLS_E_SHORT_MEMORY_BUFFER and updates &*ret_size* if &*ret_size* is not enough to hold the distribution point, or the type of the distribution point if everything was ok. The type is one of the enumerated gnutls_x509_subject_alt_n If the certificate does not have an Alternative name with the specified sequence number then GNUTLS_E_REQUESTED_DATA_N is returned.

gnutls_x509_crt_set_crl_dist_points2 ()

```
int                 gnutls_x509_crt_set_crl_dist_points2
                                                        (gnutls_x509_crt_t crt,
                                                         gnutls_x509_subject_alt_name_t  ↩
                                                             type,
                                                         const void *data,
                                                         unsigned int data_size,
                                                         unsigned int reason_flags);
```

This function will set the CRL distribution points certificate extension.

crt : a certificate of type gnutls_x509_crt_t

type : is one of the gnutls_x509_subject_alt_name_t enumerations

data : The data to be set

data_size : The data size

reason_flags : revocation reasons

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

Since 2.6.0

gnutls_x509_crt_set_crl_dist_points ()

```
int                 gnutls_x509_crt_set_crl_dist_points (gnutls_x509_crt_t crt,
                                                         gnutls_x509_subject_alt_name_t  ↩
                                                             type,
                                                         const void *data_string,
                                                         unsigned int reason_flags);
```

This function will set the CRL distribution points certificate extension.

crt : a certificate of type gnutls_x509_crt_t

type : is one of the gnutls_x509_subject_alt_name_t enumerations

data_string : The data to be set

reason_flags : revocation reasons

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_x509_crt_cpy_crl_dist_points ()

```
int                 gnutls_x509_crt_cpy_crl_dist_points (gnutls_x509_crt_t dst,
                                                         gnutls_x509_crt_t src);
```

This function will copy the CRL distribution points certificate extension, from the source to the destination certificate. This may be useful to copy from a CA certificate to issued ones.

dst : a certificate of type gnutls_x509_crt_t

src : the certificate where the dist points will be copied from

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_x509_crt_get_activation_time ()

```
time_t              gnutls_x509_crt_get_activation_time (gnutls_x509_crt_t cert);
```

This function will return the time this Certificate was or will be activated.

cert : should contain a gnutls_x509_crt_t structure

Returns : activation time, or (time_t)-1 on error.

gnutls_x509_crt_get_expiration_time ()

```
time_t              gnutls_x509_crt_get_expiration_time (gnutls_x509_crt_t cert);
```

This function will return the time this Certificate was or will be expired.

cert : should contain a gnutls_x509_crt_t structure

Returns : expiration time, or (time_t)-1 on error.

gnutls_x509_crt_get_serial ()

```
int                 gnutls_x509_crt_get_serial          (gnutls_x509_crt_t cert,
                                                         void *result,
                                                         size_t *result_size);
```

This function will return the X.509 certificate's serial number. This is obtained by the X509 Certificate serialNumber field. Serial is not always a 32 or 64bit number. Some CAs use large serial numbers, thus it may be wise to handle it as something opaque.

cert : should contain a gnutls_x509_crt_t structure

result : The place where the serial number will be copied

result_size : Holds the size of the result field.

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_x509_crt_get_pk_algorithm ()

```
int                 gnutls_x509_crt_get_pk_algorithm    (gnutls_x509_crt_t cert,
                                                         unsigned int *bits);
```

This function will return the public key algorithm of an X.509 certificate.

If bits is non null, it should have enough size to hold the parameters size in bits. For RSA the bits returned is the modulus. For DSA the bits returned are of the public exponent.

cert : should contain a gnutls_x509_crt_t structure

bits : if bits is non null it will hold the size of the parameters' in bits

Returns : a member of the gnutls_pk_algorithm_t enumeration on success, or a negative value on error.

gnutls_x509_crt_get_pk_rsa_raw ()

```
int                 gnutls_x509_crt_get_pk_rsa_raw      (gnutls_x509_crt_t crt,
                                                         gnutls_datum_t *m,
                                                         gnutls_datum_t *e);
```

This function will export the RSA public key's parameters found in the given structure. The new parameters will be allocated using gnutls_malloc() and will be stored in the appropriate datum.

crt : Holds the certificate

m : will hold the modulus

e : will hold the public exponent

Returns : GNUTLS_E_SUCCESS on success, otherwise an error.

gnutls_x509_crt_get_pk_dsa_raw ()

```
int                 gnutls_x509_crt_get_pk_dsa_raw      (gnutls_x509_crt_t crt,
                                                         gnutls_datum_t *p,
                                                         gnutls_datum_t *q,
                                                         gnutls_datum_t *g,
                                                         gnutls_datum_t *y);
```

This function will export the DSA public key's parameters found in the given certificate. The new parameters will be allocated using gnutls_malloc() and will be stored in the appropriate datum.

crt : Holds the certificate

p : will hold the p

q : will hold the q

g : will hold the g

y : will hold the y

Returns : GNUTLS_E_SUCCESS on success, otherwise an error.

gnutls_x509_crt_get_subject_alt_name ()

```
int                 gnutls_x509_crt_get_subject_alt_name
                                                        (gnutls_x509_crt_t cert,
                                                         unsigned int seq,
                                                         void *ret,
                                                         size_t *ret_size,
                                                         unsigned int *critical);
```

This function retrieves the Alternative Name (2.5.29.17), contained in the given certificate in the X509v3 Certificate Extensions.

When the SAN type is otherName, it will extract the data in the otherName's value field, and GNUTLS_SAN_OTHERNAME is returned. You may use gnutls_x509_crt_get_subject_alt_othername_oid() to get the corresponding OID and the "virtual" SAN types (e.g., GNUTLS_SAN_OTHERNAME_XMPP).

If an otherName OID is known, the data will be decoded. Otherwise the returned data will be DER encoded, and you will have to decode it yourself. Currently, only the RFC 3920 id-on-xmppAddr SAN is recognized.

cert : should contain a gnutls_x509_crt_t structure

seq : specifies the sequence number of the alt name (0 for the first one, 1 for the second etc.)

ret : is the place where the alternative name will be copied to

ret_size : holds the size of ret.

critical : will be non zero if the extension is marked as critical (may be null)

Returns : the alternative subject name type on success, one of the enumerated gnutls_x509_subject_alt_name_t. It will return GNUTLS_E_SHORT_MEMORY_BUFFER if *ret_size* is not large enough to hold the value. In that case *ret_size* will be updated with the required size. If the certificate does not have an Alternative name with the specified sequence number then GNUTLS_E_REQUESTED_DATA_NOT_AVAILABLE is returned.

gnutls_x509_crt_get_subject_alt_name2 ()

```
int                 gnutls_x509_crt_get_subject_alt_name2
                                                        (gnutls_x509_crt_t cert,
                                                         unsigned int seq,
                                                         void *ret,
                                                         size_t *ret_size,
                                                         unsigned int *ret_type,
                                                         unsigned int *critical);
```

This function will return the alternative names, contained in the given certificate. It is the same as gnutls_x509_crt_get_subject_alt_name except for the fact that it will return the type of the alternative name in *ret_type* even if the function fails for some reason (i.e. the buffer provided is not enough).

cert : should contain a gnutls_x509_crt_t structure

seq : specifies the sequence number of the alt name (0 for the first one, 1 for the second etc.)

ret : is the place where the alternative name will be copied to

ret_size : holds the size of ret.

ret_type : holds the type of the alternative name (one of gnutls_x509_subject_alt_name_t).

critical : will be non zero if the extension is marked as critical (may be null)

Returns : the alternative subject name type on success, one of the enumerated gnutls_x509_subject_alt_name_t. It will return GNUTLS_E_SHORT_MEMORY_BUFFER if *ret_size* is not large enough to hold the value. In that case *ret_size* will be updated with the required size. If the certificate does not have an Alternative name with the specified sequence number then GNUTLS_E_REQUESTED_DATA_NOT_AVAILABLE is returned.

gnutls_x509_crt_get_subject_alt_othername_oid ()

```
int                 gnutls_x509_crt_get_subject_alt_othername_oid
                                                        (gnutls_x509_crt_t cert,
                                                         unsigned int seq,
                                                         void *ret,
                                                         size_t *ret_size);
```

This function will extract the type OID of an otherName Subject Alternative Name, contained in the given certificate, and return the type as an enumerated element.

This function is only useful if gnutls_x509_crt_get_subject_alt_name() returned GNUTLS_SAN_OTHERNAME.

cert : should contain a gnutls_x509_crt_t structure

seq : specifies the sequence number of the alt name (0 for the first one, 1 for the second etc.)

ret : is the place where the otherName OID will be copied to

ret_size : holds the size of ret.

Returns : the alternative subject name type on success, one of the enumerated gnutls_x509_subject_alt_name_t. For supported OIDs, it will return one of the virtual (GNUTLS_SAN_OTHERNAME_*) types, e.g. GNUTLS_SAN_OTHERNAME_XMPP, and GNUTLS_SAN_OTHERNAME for unknown OIDs. It will return GNUTLS_E_SHORT_MEMORY_BUFFER if *ret_size* is not large enough to hold the value. In that case *ret_size* will be updated with the required size. If the certificate does not have an Alternative name with the specified sequence number and with the otherName type then GNUTLS_E_REQUESTED_DATA_NOT_AVAILABLE is returned.

gnutls_x509_crt_get_issuer_alt_name ()

```
int                 gnutls_x509_crt_get_issuer_alt_name (gnutls_x509_crt_t cert,
                                                         unsigned int seq,
                                                         void *ret,
                                                         size_t *ret_size,
                                                         unsigned int *critical);
```

This function retrieves the Issuer Alternative Name (2.5.29.18), contained in the given certificate in the X509v3 Certificate Extensions.

When the SAN type is otherName, it will extract the data in the otherName's value field, and GNUTLS_SAN_OTHERNAME is returned. You may use gnutls_x509_crt_get_subject_alt_othername_oid() to get the corresponding OID and the "virtual" SAN types (e.g., GNUTLS_SAN_OTHERNAME_XMPP).

If an otherName OID is known, the data will be decoded. Otherwise the returned data will be DER encoded, and you will have to decode it yourself. Currently, only the RFC 3920 id-on-xmppAddr Issuer AltName is recognized.

cert **:** should contain a gnutls_x509_crt_t structure

seq **:** specifies the sequence number of the alt name (0 for the first one, 1 for the second etc.)

ret **:** is the place where the alternative name will be copied to

ret_size **:** holds the size of ret.

critical **:** will be non zero if the extension is marked as critical (may be null)

Returns **:** the alternative issuer name type on success, one of the enumerated gnutls_x509_subject_alt_name_t. It will return GNUTLS_E_SHORT_MEMORY_BUFFER if *ret_size* is not large enough to hold the value. In that case *ret_size* will be updated with the required size. If the certificate does not have an Alternative name with the specified sequence number then GNUTLS_E_REQUESTED_DATA_NOT_AVAILABLE is returned.

Since 2.10.0

gnutls_x509_crt_get_issuer_alt_name2 ()

```
int                 gnutls_x509_crt_get_issuer_alt_name2
                                                        (gnutls_x509_crt_t cert,
                                                         unsigned int seq,
                                                         void *ret,
                                                         size_t *ret_size,
                                                         unsigned int *ret_type,
                                                         unsigned int *critical);
```

This function will return the alternative names, contained in the given certificate. It is the same as gnutls_x509_crt_get_issuer_alt_name() except for the fact that it will return the type of the alternative name in *ret_type* even if the function fails for some reason (i.e. the buffer provided is not enough).

cert **:** should contain a gnutls_x509_crt_t structure

seq **:** specifies the sequence number of the alt name (0 for the first one, 1 for the second etc.)

ret **:** is the place where the alternative name will be copied to

ret_size **:** holds the size of ret.

ret_type **:** holds the type of the alternative name (one of gnutls_x509_subject_alt_name_t).

critical **:** will be non zero if the extension is marked as critical (may be null)

Returns **:** the alternative issuer name type on success, one of the enumerated gnutls_x509_subject_alt_name_t. It will return GNUTLS_E_SHORT_MEMORY_BUFFER if *ret_size* is not large enough to hold the value. In that case *ret_size* will be updated with the required size. If the certificate does not have an Alternative name with the specified sequence number then GNUTLS_E_REQUESTED_DATA_NOT_AVAILABLE is returned.

Since 2.10.0

gnutls_x509_crt_get_issuer_alt_othername_oid ()

```
int                 gnutls_x509_crt_get_issuer_alt_othername_oid
                                                    (gnutls_x509_crt_t cert,
                                                     unsigned int seq,
                                                     void *ret,
                                                     size_t *ret_size);
```

This function will extract the type OID of an otherName Subject Alternative Name, contained in the given certificate, and return the type as an enumerated element.

This function is only useful if gnutls_x509_crt_get_issuer_alt_name() returned GNUTLS_SAN_OTHERNAME.

cert **:** should contain a gnutls_x509_crt_t structure

seq **:** specifies the sequence number of the alt name (0 for the first one, 1 for the second etc.)

ret **:** is the place where the otherName OID will be copied to

ret_size **:** holds the size of ret.

Returns **:** the alternative issuer name type on success, one of the enumerated gnutls_x509_subject_alt_name_t. For supported OIDs, it will return one of the virtual (GNUTLS_SAN_OTHERNAME_*) types, e.g. GNUTLS_SAN_OTHERNAME_XMPP, and GNUTLS_SAN_OTHERNAME for unknown OIDs. It will return GNUTLS_E_SHORT_MEMORY_BUFFER if *ret_size* is not large enough to hold the value. In that case *ret_size* will be updated with the required size. If the certificate does not have an Alternative name with the specified sequence number and with the otherName type then GNUTLS_E_REQUESTED_DATA_NOT_AVAILABLE is returned.

Since 2.10.0

gnutls_x509_crt_get_ca_status ()

```
int                 gnutls_x509_crt_get_ca_status       (gnutls_x509_crt_t cert,
                                                         unsigned int *critical);
```

This function will return certificates CA status, by reading the basicConstraints X.509 extension (2.5.29.19). If the certificate is a CA a positive value will be returned, or zero if the certificate does not have CA flag set.

Use gnutls_x509_crt_get_basic_constraints() if you want to read the pathLenConstraint field too.

cert **:** should contain a gnutls_x509_crt_t structure

critical **:** will be non zero if the extension is marked as critical

Returns **:** A negative value may be returned in case of parsing error. If the certificate does not contain the basicConstraints extension GNUTLS_E_REQUESTED_DATA_NOT_AVAILABLE will be returned.

gnutls_x509_crt_get_basic_constraints ()

```
int                 gnutls_x509_crt_get_basic_constraints
                                                    (gnutls_x509_crt_t cert,
                                                     unsigned int *critical,
                                                     int *ca,
                                                     int *pathlen);
```

This function will read the certificate's basic constraints, and return the certificates CA status. It reads the basicConstraints X.509 extension (2.5.29.19).

cert : should contain a gnutls_x509_crt_t structure

critical : will be non zero if the extension is marked as critical

ca : pointer to output integer indicating CA status, may be NULL, value is 1 if the certificate CA flag is set, 0 otherwise.

pathlen : pointer to output integer indicating path length (may be NULL), non-negative values indicate a present pathLenConstraint field and the actual value, -1 indicate that the field is absent.

Returns : If the certificate is a CA a positive value will be returned, or zero if the certificate does not have CA flag set. A negative value may be returned in case of errors. If the certificate does not contain the basicConstraints extension GNUTLS_E_REQUESTED_DATA_NOT_AVAILABLE will be returned.

gnutls_x509_crt_get_key_usage ()

```
int                 gnutls_x509_crt_get_key_usage       (gnutls_x509_crt_t cert,
                                                         unsigned int *key_usage,
                                                         unsigned int *critical);
```

This function will return certificate's key usage, by reading the keyUsage X.509 extension (2.5.29.15). The key usage value will ORed values of the: GNUTLS_KEY_DIGITAL_SIGNATURE, GNUTLS_KEY_NON_REPUDIATION, GNUTLS_KEY_KEY_ENCI GNUTLS_KEY_DATA_ENCIPHERMENT, GNUTLS_KEY_KEY_AGREEMENT, GNUTLS_KEY_KEY_CERT_SIGN, GNUTLS_ GNUTLS_KEY_ENCIPHER_ONLY, GNUTLS_KEY_DECIPHER_ONLY.

cert : should contain a gnutls_x509_crt_t structure

key_usage : where the key usage bits will be stored

critical : will be non zero if the extension is marked as critical

Returns : the certificate key usage, or a negative value in case of parsing error. If the certificate does not contain the keyUsage extension GNUTLS_E_REQUESTED_DATA_NOT_AVAILABLE will be returned.

gnutls_x509_crt_set_key_usage ()

```
int                 gnutls_x509_crt_set_key_usage       (gnutls_x509_crt_t crt,
                                                         unsigned int usage);
```

This function will set the keyUsage certificate extension.

crt : a certificate of type gnutls_x509_crt_t

usage : an ORed sequence of the GNUTLS_KEY_* elements.

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_x509_crt_get_proxy ()

```
int                 gnutls_x509_crt_get_proxy           (gnutls_x509_crt_t cert,
                                                         unsigned int *critical,
                                                         int *pathlen,
                                                         char **policyLanguage,
                                                         char **policy,
                                                         size_t *sizeof_policy);
```

This function will get information from a proxy certificate. It reads the ProxyCertInfo X.509 extension (1.3.6.1.5.5.7.1.14).

cert : should contain a gnutls_x509_crt_t structure

critical : will be non zero if the extension is marked as critical

pathlen : pointer to output integer indicating path length (may be NULL), non-negative values indicate a present pCPathLenConstraint field and the actual value, -1 indicate that the field is absent.

policyLanguage : output variable with OID of policy language

policy : output variable with policy data

sizeof_policy : output variable size of policy data

Returns : On success, GNUTLS_E_SUCCESS (zero) is returned, otherwise an error code is returned.

gnutls_x509_dn_oid_known ()

```
int                 gnutls_x509_dn_oid_known            (const char *oid);
```

This function will inform about known DN OIDs. This is useful since functions like gnutls_x509_crt_set_dn_by_oid() use the information on known OIDs to properly encode their input. Object Identifiers that are not known are not encoded by these functions, and their input is stored directly into the ASN.1 structure. In that case of unknown OIDs, you have the responsibility of DER encoding your data.

oid : holds an Object Identifier in a null terminated string

Returns : 1 on known OIDs and 0 otherwise.

gnutls_x509_crt_get_extension_oid ()

```
int                 gnutls_x509_crt_get_extension_oid   (gnutls_x509_crt_t cert,
                                                         int indx,
                                                         void *oid,
                                                         size_t *sizeof_oid);
```

This function will return the requested extension OID in the certificate. The extension OID will be stored as a string in the provided buffer.

cert : should contain a gnutls_x509_crt_t structure

indx : Specifies which extension OID to send. Use zero to get the first one.

oid : a pointer to a structure to hold the OID (may be null)

sizeof_oid : initially holds the size of *oid*

Returns : On success, GNUTLS_E_SUCCESS (zero) is returned, otherwise an error code is returned. If you have reached the last extension available GNUTLS_E_REQUESTED_DATA_NOT_AVAILABLE will be returned.

gnutls_x509_crt_get_extension_by_oid ()

```
int                 gnutls_x509_crt_get_extension_by_oid
                                                    (gnutls_x509_crt_t cert,
                                                     const char *oid,
                                                     int indx,
                                                     void *buf,
                                                     size_t *sizeof_buf,
                                                     unsigned int *critical);
```

This function will return the extension specified by the OID in the certificate. The extensions will be returned as binary data DER encoded, in the provided buffer.

cert : should contain a gnutls_x509_crt_t structure

oid : holds an Object Identified in null terminated string

indx : In case multiple same OIDs exist in the extensions, this specifies which to send. Use zero to get the first one.

buf : a pointer to a structure to hold the name (may be null)

sizeof_buf : initially holds the size of *buf*

critical : will be non zero if the extension is marked as critical

Returns : On success, GNUTLS_E_SUCCESS (zero) is returned, otherwise an error code is returned. If the certificate does not contain the specified extension GNUTLS_E_REQUESTED_DATA_NOT_AVAILABLE will be returned.

gnutls_x509_crt_get_extension_info ()

```
int                 gnutls_x509_crt_get_extension_info  (gnutls_x509_crt_t cert,
                                                         int indx,
                                                         void *oid,
                                                         size_t *sizeof_oid,
                                                         int *critical);
```

This function will return the requested extension OID in the certificate, and the critical flag for it. The extension OID will be stored as a string in the provided buffer. Use gnutls_x509_crt_get_extension_data() to extract the data.

If the buffer provided is not long enough to hold the output, then **sizeof_oid* is updated and GNUTLS_E_SHORT_MEMORY_BUFF will be returned.

cert : should contain a gnutls_x509_crt_t structure

indx : Specifies which extension OID to send. Use zero to get the first one.

oid : a pointer to a structure to hold the OID

sizeof_oid : initially holds the maximum size of *oid*, on return holds actual size of *oid*.

critical : output variable with critical flag, may be NULL.

Returns : On success, GNUTLS_E_SUCCESS (zero) is returned, otherwise an error code is returned. If you have reached the last extension available GNUTLS_E_REQUESTED_DATA_NOT_AVAILABLE will be returned.

gnutls_x509_crt_get_extension_data ()

```
int                 gnutls_x509_crt_get_extension_data  (gnutls_x509_crt_t cert,
                                                         int indx,
                                                         void *data,
                                                         size_t *sizeof_data);
```

This function will return the requested extension data in the certificate. The extension data will be stored as a string in the provided buffer.

Use gnutls_x509_crt_get_extension_info() to extract the OID and critical flag. Use gnutls_x509_crt_get_extension_by_oid() instead, if you want to get data indexed by the extension OID rather than sequence.

cert : should contain a gnutls_x509_crt_t structure

indx : Specifies which extension OID to send. Use zero to get the first one.

data : a pointer to a structure to hold the data (may be null)

sizeof_data : initially holds the size of *oid*

Returns : On success, GNUTLS_E_SUCCESS (zero) is returned, otherwise an error code is returned. If you have reached the last extension available GNUTLS_E_REQUESTED_DATA_NOT_AVAILABLE will be returned.

gnutls_x509_crt_set_extension_by_oid ()

```
int                 gnutls_x509_crt_set_extension_by_oid
                                                        (gnutls_x509_crt_t crt,
                                                         const char *oid,
                                                         const void *buf,
                                                         size_t sizeof_buf,
                                                         unsigned int critical);
```

This function will set an the extension, by the specified OID, in the certificate. The extension data should be binary data DER encoded.

crt : a certificate of type gnutls_x509_crt_t

oid : holds an Object Identified in null terminated string

buf : a pointer to a DER encoded data

sizeof_buf : holds the size of *buf*

critical : should be non zero if the extension is to be marked as critical

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_x509_crt_set_dn_by_oid ()

```
int                 gnutls_x509_crt_set_dn_by_oid       (gnutls_x509_crt_t crt,
                                                         const char *oid,
                                                         unsigned int raw_flag,
                                                         const void *name,
                                                         unsigned int sizeof_name);
```

This function will set the part of the name of the Certificate subject, specified by the given OID. The input string should be ASCII or UTF-8 encoded.

Some helper macros with popular OIDs can be found in gnutls/x509.h With this function you can only set the known OIDs. You can test for known OIDs using gnutls_x509_dn_oid_known(). For OIDs that are not known (by gnutls) you should properly DER encode your data, and call this function with *raw_flag* set.

crt : a certificate of type gnutls_x509_crt_t

oid : holds an Object Identifier in a null terminated string

raw_flag : must be 0, or 1 if the data are DER encoded

name : a pointer to the name

sizeof_name : holds the size of *name*

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_x509_crt_set_issuer_dn_by_oid ()

```
int                 gnutls_x509_crt_set_issuer_dn_by_oid
                                                    (gnutls_x509_crt_t crt,
                                                     const char *oid,
                                                     unsigned int raw_flag,
                                                     const void *name,
                                                     unsigned int sizeof_name);
```

This function will set the part of the name of the Certificate issuer, specified by the given OID. The input string should be ASCII or UTF-8 encoded.

Some helper macros with popular OIDs can be found in gnutls/x509.h With this function you can only set the known OIDs. You can test for known OIDs using gnutls_x509_dn_oid_known(). For OIDs that are not known (by gnutls) you should properly DER encode your data, and call this function with *raw_flag* set.

Normally you do not need to call this function, since the signing operation will copy the signer's name as the issuer of the certificate.

crt : a certificate of type gnutls_x509_crt_t

oid : holds an Object Identifier in a null terminated string

raw_flag : must be 0, or 1 if the data are DER encoded

name : a pointer to the name

sizeof_name : holds the size of *name*

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_x509_crt_set_version ()

```
int                 gnutls_x509_crt_set_version         (gnutls_x509_crt_t crt,
                                                         unsigned int version);
```

This function will set the version of the certificate. This must be one for X.509 version 1, and so on. Plain certificates without extensions must have version set to one.

To create well-formed certificates, you must specify version 3 if you use any certificate extensions. Extensions are created by functions such as gnutls_x509_crt_set_subject_alt_name() or gnutls_x509_crt_set_key_usage().

crt : a certificate of type gnutls_x509_crt_t

version : holds the version number. For X.509v1 certificates must be 1.

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_x509_crt_set_key ()

```
int                 gnutls_x509_crt_set_key             (gnutls_x509_crt_t crt,
                                                         gnutls_x509_privkey_t key);
```

This function will set the public parameters from the given private key to the certificate. Only RSA keys are currently supported.

crt : a certificate of type gnutls_x509_crt_t

key : holds a private key

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_x509_crt_set_ca_status ()

```
int                 gnutls_x509_crt_set_ca_status       (gnutls_x509_crt_t crt,
                                                         unsigned int ca);
```

This function will set the basicConstraints certificate extension. Use gnutls_x509_crt_set_basic_constraints() if you want to control the pathLenConstraint field too.

crt : a certificate of type gnutls_x509_crt_t

ca : true(1) or false(0). Depending on the Certificate authority status.

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_x509_crt_set_basic_constraints ()

```
int                 gnutls_x509_crt_set_basic_constraints
                                                        (gnutls_x509_crt_t crt,
                                                         unsigned int ca,
                                                         int pathLenConstraint);
```

This function will set the basicConstraints certificate extension.

crt : a certificate of type gnutls_x509_crt_t

ca : true(1) or false(0). Depending on the Certificate authority status.

pathLenConstraint : non-negative values indicate maximum length of path, and negative values indicate that the pathLenConstraints field should not be present.

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_x509_crt_set_subject_alternative_name ()

```
int                 gnutls_x509_crt_set_subject_alternative_name
                                                        (gnutls_x509_crt_t crt,
                                                         gnutls_x509_subject_alt_name_t ↩
                                                             type,
                                                         const char *data_string);
```

This function will set the subject alternative name certificate extension. This function assumes that data can be expressed as a null terminated string.

The name of the function is unfortunate since it is incosistent with gnutls_x509_crt_get_subject_alt_name().

crt **:** a certificate of type gnutls_x509_crt_t

type **:** is one of the gnutls_x509_subject_alt_name_t enumerations

data_string **:** The data to be set, a zero terminated string

Returns **:** On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_x509_crt_set_subject_alt_name ()

```
int                 gnutls_x509_crt_set_subject_alt_name
                                                    (gnutls_x509_crt_t crt,
                                                     gnutls_x509_subject_alt_name_t  ↩
                                                         type,
                                                     const void *data,
                                                     unsigned int data_size,
                                                     unsigned int flags);
```

This function will set the subject alternative name certificate extension. It can set the following types:

&GNUTLS_SAN_DNSNAME: as a text string

&GNUTLS_SAN_RFC822NAME: as a text string

&GNUTLS_SAN_URI: as a text string

&GNUTLS_SAN_IPADDRESS: as a binary IP address (4 or 16 bytes)

Other values can be set as binary values with the proper DER encoding.

crt **:** a certificate of type gnutls_x509_crt_t

type **:** is one of the gnutls_x509_subject_alt_name_t enumerations

data **:** The data to be set

data_size **:** The size of data to be set

flags **:** GNUTLS_FSAN_SET to clear previous data or GNUTLS_FSAN_APPEND to append.

Returns **:** On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

Since 2.6.0

gnutls_x509_crt_sign ()

```
int                 gnutls_x509_crt_sign                (gnutls_x509_crt_t crt,
                                                         gnutls_x509_crt_t issuer,
                                                         gnutls_x509_privkey_t issuer_key);
```

This function is the same a gnutls_x509_crt_sign2() with no flags, and SHA1 as the hash algorithm.

crt **:** a certificate of type gnutls_x509_crt_t

issuer **:** is the certificate of the certificate issuer

issuer_key **:** holds the issuer's private key

Returns **:** On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_x509_crt_sign2 ()

```
int                 gnutls_x509_crt_sign2               (gnutls_x509_crt_t crt,
                                                         gnutls_x509_crt_t issuer,
                                                         gnutls_x509_privkey_t issuer_key,
                                                         gnutls_digest_algorithm_t dig,
                                                         unsigned int flags);
```

This function will sign the certificate with the issuer's private key, and will copy the issuer's information into the certificate.

This must be the last step in a certificate generation since all the previously set parameters are now signed.

crt : a certificate of type gnutls_x509_crt_t

issuer : is the certificate of the certificate issuer

issuer_key : holds the issuer's private key

dig : The message digest to use, GNUTLS_DIG_SHA1 is a safe choice

flags : must be 0

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_x509_crt_set_activation_time ()

```
int                 gnutls_x509_crt_set_activation_time (gnutls_x509_crt_t cert,
                                                         time_t act_time);
```

This function will set the time this Certificate was or will be activated.

cert : a certificate of type gnutls_x509_crt_t

act_time : The actual time

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_x509_crt_set_expiration_time ()

```
int                 gnutls_x509_crt_set_expiration_time (gnutls_x509_crt_t cert,
                                                         time_t exp_time);
```

This function will set the time this Certificate will expire.

cert : a certificate of type gnutls_x509_crt_t

exp_time : The actual time

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_x509_crt_set_serial ()

```
int                 gnutls_x509_crt_set_serial          (gnutls_x509_crt_t cert,
                                                         const void *serial,
                                                         size_t serial_size);
```

This function will set the X.509 certificate's serial number. Serial is not always a 32 or 64bit number. Some CAs use large serial numbers, thus it may be wise to handle it as something opaque.

cert : a certificate of type gnutls_x509_crt_t

serial : The serial number

serial_size : Holds the size of the serial field.

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_x509_crt_set_subject_key_id ()

```
int                 gnutls_x509_crt_set_subject_key_id  (gnutls_x509_crt_t cert,
                                                         const void *id,
                                                         size_t id_size);
```

This function will set the X.509 certificate's subject key ID extension.

cert : a certificate of type gnutls_x509_crt_t

id : The key ID

id_size : Holds the size of the serial field.

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_x509_crt_set_proxy_dn ()

```
int                 gnutls_x509_crt_set_proxy_dn        (gnutls_x509_crt_t crt,
                                                         gnutls_x509_crt_t eecrt,
                                                         unsigned int raw_flag,
                                                         const void *name,
                                                         unsigned int sizeof_name);
```

This function will set the subject in *crt* to the end entity's *eecrt* subject name, and add a single Common Name component *name* of size *sizeof_name*. This corresponds to the required proxy certificate naming style. Note that if *name* is NULL, you MUST set it later by using gnutls_x509_crt_set_dn_by_oid() or similar.

crt : a gnutls_x509_crt_t structure with the new proxy cert

eecrt : the end entity certificate that will be issuing the proxy

raw_flag : must be 0, or 1 if the CN is DER encoded

name : a pointer to the CN name, may be NULL (but MUST then be added later)

sizeof_name : holds the size of *name*

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_x509_crt_set_proxy ()

```
int                 gnutls_x509_crt_set_proxy           (gnutls_x509_crt_t crt,
                                                         int pathLenConstraint,
                                                         const char *policyLanguage,
                                                         const char *policy,
                                                         size_t sizeof_policy);
```

This function will set the proxyCertInfo extension.

crt : a certificate of type gnutls_x509_crt_t

pathLenConstraint : non-negative values indicate maximum length of path, and negative values indicate that the pathLenConstraints field should not be present.

policyLanguage : OID describing the language of *policy*.

policy : opaque byte array with policy language, can be NULL

sizeof_policy : size of *policy*.

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_x509_crt_print ()

```
int                 gnutls_x509_crt_print               (gnutls_x509_crt_t cert,
                                                         gnutls_certificate_print_formats_t ↩
                                                              format,
                                                         gnutls_datum_t *out);
```

This function will pretty print a X.509 certificate, suitable for display to a human.

If the format is GNUTLS_CRT_PRINT_FULL then all fields of the certificate will be output, on multiple lines. The GNUTLS_CRT_PRI format will generate one line with some selected fields, which is useful for logging purposes.

The output *out* needs to be deallocate using gnutls_free().

cert : The structure to be printed

format : Indicate the format to use

out : Newly allocated datum with zero terminated string.

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_x509_crl_print ()

```
int                 gnutls_x509_crl_print               (gnutls_x509_crl_t crl,
                                                         gnutls_certificate_print_formats_t ↩
                                                              format,
                                                         gnutls_datum_t *out);
```

This function will pretty print a X.509 certificate revocation list, suitable for display to a human.

The output *out* needs to be deallocate using gnutls_free().

crl : The structure to be printed

format : Indicate the format to use

out : Newly allocated datum with zero terminated string.

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_x509_crt_get_raw_issuer_dn ()

```
int                 gnutls_x509_crt_get_raw_issuer_dn   (gnutls_x509_crt_t cert,
                                                         gnutls_datum_t *start);
```

This function will return a pointer to the DER encoded DN structure and the length.

cert : should contain a gnutls_x509_crt_t structure

start : will hold the starting point of the DN

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.or a negative value on error.

gnutls_x509_crt_get_raw_dn ()

```
int                 gnutls_x509_crt_get_raw_dn          (gnutls_x509_crt_t cert,
                                                         gnutls_datum_t *start);
```

This function will return a pointer to the DER encoded DN structure and the length.

cert : should contain a gnutls_x509_crt_t structure

start : will hold the starting point of the DN

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value. or a negative value on error.

gnutls_x509_rdn_get ()

```
int                 gnutls_x509_rdn_get                 (const gnutls_datum_t *idn,
                                                         char *buf,
                                                         size_t *sizeof_buf);
```

This function will return the name of the given RDN sequence. The name will be in the form "C=xxxx,O=yyyy,CN=zzzz" as described in RFC2253.

idn : should contain a DER encoded RDN sequence

buf : a pointer to a structure to hold the peer's name

sizeof_buf : holds the size of *buf*

Returns : On success, GNUTLS_E_SUCCESS is returned, or GNUTLS_E_SHORT_MEMORY_BUFFER is returned and **s-izeof_buf* is updated if the provided buffer is not long enough, otherwise a negative error value.

gnutls_x509_rdn_get_oid ()

```
int                 gnutls_x509_rdn_get_oid             (const gnutls_datum_t *idn,
                                                         int indx,
                                                         void *buf,
                                                         size_t *sizeof_buf);
```

This function will return the specified Object identifier, of the RDN sequence.

idn : should contain a DER encoded RDN sequence

indx : Indicates which OID to return. Use 0 for the first one.

***buf*:** a pointer to a structure to hold the peer's name OID

***sizeof_buf*:** holds the size of *buf*

***Returns*:** On success, GNUTLS_E_SUCCESS is returned, or GNUTLS_E_SHORT_MEMORY_BUFFER is returned and **sizeof_buf* is updated if the provided buffer is not long enough, otherwise a negative error value.

Since 2.4.0

gnutls_x509_rdn_get_by_oid ()

```
int                 gnutls_x509_rdn_get_by_oid          (const gnutls_datum_t *idn,
                                                         const char *oid,
                                                         int indx,
                                                         unsigned int raw_flag,
                                                         void *buf,
                                                         size_t *sizeof_buf);
```

This function will return the name of the given Object identifier, of the RDN sequence. The name will be encoded using the rules from RFC2253.

***idn*:** should contain a DER encoded RDN sequence

***oid*:** an Object Identifier

***indx*:** In case multiple same OIDs exist in the RDN indicates which to send. Use 0 for the first one.

***raw_flag*:** If non zero then the raw DER data are returned.

***buf*:** a pointer to a structure to hold the peer's name

***sizeof_buf*:** holds the size of *buf*

***Returns*:** On success, GNUTLS_E_SUCCESS is returned, or GNUTLS_E_SHORT_MEMORY_BUFFER is returned and **sizeof_buf* is updated if the provided buffer is not long enough, otherwise a negative error value.

gnutls_x509_dn_t

```
typedef void *gnutls_x509_dn_t;
```

gnutls_x509_crt_get_subject ()

```
int                 gnutls_x509_crt_get_subject         (gnutls_x509_crt_t cert,
                                                         gnutls_x509_dn_t *dn);
```

Return the Certificate's Subject DN as an opaque data type. You may use gnutls_x509_dn_get_rdn_ava() to decode the DN.

Note that *dn* should be treated as constant. Because points into the *cert* object, you may not deallocate *cert* and continue to access *dn*.

***cert*:** should contain a gnutls_x509_crt_t structure

***dn*:** output variable with pointer to opaque DN.

***Returns*:** Returns 0 on success, or an error code.

gnutls_x509_crt_get_issuer ()

```
int                 gnutls_x509_crt_get_issuer          (gnutls_x509_crt_t cert,
                                                         gnutls_x509_dn_t *dn);
```

Return the Certificate's Issuer DN as an opaque data type. You may use gnutls_x509_dn_get_rdn_ava() to decode the DN.

Note that *dn* should be treated as constant. Because points into the *cert* object, you may not deallocate *cert* and continue to access *dn*.

cert **:** should contain a gnutls_x509_crt_t structure

dn **:** output variable with pointer to opaque DN

Returns **:** Returns 0 on success, or an error code.

gnutls_x509_dn_get_rdn_ava ()

```
int                 gnutls_x509_dn_get_rdn_ava          (gnutls_x509_dn_t dn,
                                                         int irdn,
                                                         int iava,
                                                         gnutls_x509_ava_st *ava);
```

Get pointers to data within the DN.

Note that *ava* will contain pointers into the *dn* structure, so you should not modify any data or deallocate it. Note also that the DN in turn points into the original certificate structure, and thus you may not deallocate the certificate and continue to access *dn*.

dn **:** input variable with opaque DN pointer

irdn **:** index of RDN

iava **:** index of AVA.

ava **:** Pointer to structure which will hold output information.

Returns **:** Returns 0 on success, or an error code.

gnutls_x509_dn_init ()

```
int                 gnutls_x509_dn_init                 (gnutls_x509_dn_t *dn);
```

This function initializes a gnutls_x509_dn_t structure.

The object returned must be deallocated using gnutls_x509_dn_deinit().

dn **:** the object to be initialized

Returns **:** On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

Since 2.4.0

gnutls_x509_dn_import ()

```
int                 gnutls_x509_dn_import               (gnutls_x509_dn_t dn,
                                                         const gnutls_datum_t *data);
```

This function parses an RDN sequence and stores the result to a gnutls_x509_dn_t structure. The structure must have been initialized with gnutls_x509_dn_init(). You may use gnutls_x509_dn_get_rdn_ava() to decode the DN.

dn : the structure that will hold the imported DN

data : should contain a DER encoded RDN sequence

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

Since 2.4.0

gnutls_x509_dn_export ()

```
int                 gnutls_x509_dn_export               (gnutls_x509_dn_t dn,
                                                         gnutls_x509_crt_fmt_t format,
                                                         void *output_data,
                                                         size_t *output_data_size);
```

This function will export the DN to DER or PEM format.

If the buffer provided is not long enough to hold the output, then *`output_data_size` is updated and GNUTLS_E_SHORT_MEMORY will be returned.

If the structure is PEM encoded, it will have a header of "BEGIN NAME".

dn : Holds the opaque DN object

format : the format of output params. One of PEM or DER.

output_data : will contain a DN PEM or DER encoded

output_data_size : holds the size of output_data (and will be replaced by the actual size of parameters)

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_x509_dn_deinit ()

```
void                gnutls_x509_dn_deinit               (gnutls_x509_dn_t dn);
```

This function deallocates the DN object as returned by gnutls_x509_dn_import().

dn : a DN opaque object pointer.

Since 2.4.0

gnutls_x509_crl_init ()

```
int                 gnutls_x509_crl_init                (gnutls_x509_crl_t *crl);
```

This function will initialize a CRL structure. CRL stands for Certificate Revocation List. A revocation list usually contains lists of certificate serial numbers that have been revoked by an Authority. The revocation lists are always signed with the authority's private key.

crl : The structure to be initialized

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_x509_crl_deinit ()

```
void                gnutls_x509_crl_deinit              (gnutls_x509_crl_t crl);
```

This function will deinitialize a CRL structure.

crl : The structure to be initialized

gnutls_x509_crl_import ()

```
int                 gnutls_x509_crl_import              (gnutls_x509_crl_t crl,
                                                         const gnutls_datum_t *data,
                                                         gnutls_x509_crt_fmt_t format);
```

This function will convert the given DER or PEM encoded CRL to the native gnutls_x509_crl_t format. The output will be stored in 'crl'.

If the CRL is PEM encoded it should have a header of "X509 CRL".

crl : The structure to store the parsed CRL.

data : The DER or PEM encoded CRL.

format : One of DER or PEM

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_x509_crl_export ()

```
int                 gnutls_x509_crl_export              (gnutls_x509_crl_t crl,
                                                         gnutls_x509_crt_fmt_t format,
                                                         void *output_data,
                                                         size_t *output_data_size);
```

This function will export the revocation list to DER or PEM format.

If the buffer provided is not long enough to hold the output, then GNUTLS_E_SHORT_MEMORY_BUFFER will be returned.

If the structure is PEM encoded, it will have a header of "BEGIN X509 CRL".

crl : Holds the revocation list

format : the format of output params. One of PEM or DER.

output_data : will contain a private key PEM or DER encoded

output_data_size : holds the size of output_data (and will be replaced by the actual size of parameters)

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value. and a negative value on failure.

gnutls_x509_crl_get_issuer_dn ()

```
int                 gnutls_x509_crl_get_issuer_dn       (const gnutls_x509_crl_t crl,
                                                         char *buf,
                                                         size_t *sizeof_buf);
```

This function will copy the name of the CRL issuer in the provided buffer. The name will be in the form "C=xxxx,O=yyyy,CN=zzzz" as described in RFC2253. The output string will be ASCII or UTF-8 encoded, depending on the certificate data.

If buf is NULL then only the size will be filled.

crl : should contain a gnutls_x509_crl_t structure

buf : a pointer to a structure to hold the peer's name (may be null)

sizeof_buf : initially holds the size of *buf*

Returns : GNUTLS_E_SHORT_MEMORY_BUFFER if the provided buffer is not long enough, and in that case the sizeof_buf will be updated with the required size, and 0 on success.

gnutls_x509_crl_get_issuer_dn_by_oid ()

```
int                 gnutls_x509_crl_get_issuer_dn_by_oid
                                                    (gnutls_x509_crl_t crl,
                                                     const char *oid,
                                                     int indx,
                                                     unsigned int raw_flag,
                                                     void *buf,
                                                     size_t *sizeof_buf);
```

This function will extract the part of the name of the CRL issuer specified by the given OID. The output will be encoded as described in RFC2253. The output string will be ASCII or UTF-8 encoded, depending on the certificate data.

Some helper macros with popular OIDs can be found in gnutls/x509.h If raw flag is zero, this function will only return known OIDs as text. Other OIDs will be DER encoded, as described in RFC2253 -- in hex format with a '#' prefix. You can check about known OIDs using gnutls_x509_dn_oid_known().

If buf is null then only the size will be filled.

crl : should contain a gnutls_x509_crl_t structure

oid : holds an Object Identified in null terminated string

indx : In case multiple same OIDs exist in the RDN, this specifies which to send. Use zero to get the first one.

raw_flag : If non zero returns the raw DER data of the DN part.

buf : a pointer to a structure to hold the peer's name (may be null)

sizeof_buf : initially holds the size of *buf*

Returns : GNUTLS_E_SHORT_MEMORY_BUFFER if the provided buffer is not long enough, and in that case the sizeof_buf will be updated with the required size, and 0 on success.

gnutls_x509_crl_get_dn_oid ()

```
int                 gnutls_x509_crl_get_dn_oid          (gnutls_x509_crl_t crl,
                                                     int indx,
                                                     void *oid,
                                                     size_t *sizeof_oid);
```

This function will extract the requested OID of the name of the CRL issuer, specified by the given index.

If oid is null then only the size will be filled.

crl : should contain a gnutls_x509_crl_t structure

indx : Specifies which DN OID to send. Use zero to get the first one.

oid : a pointer to a structure to hold the name (may be null)

sizeof_oid : initially holds the size of 'oid'

Returns : GNUTLS_E_SHORT_MEMORY_BUFFER if the provided buffer is not long enough, and in that case the sizeof_oid will be updated with the required size. On success 0 is returned.

gnutls_x509_crl_get_signature_algorithm ()

```
int                 gnutls_x509_crl_get_signature_algorithm
                                                    (gnutls_x509_crl_t crl);
```

This function will return a value of the gnutls_sign_algorithm_t enumeration that is the signature algorithm.

crl : should contain a gnutls_x509_crl_t structure

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_x509_crl_get_signature ()

```
int                 gnutls_x509_crl_get_signature       (gnutls_x509_crl_t crl,
                                                         char *sig,
                                                         size_t *sizeof_sig);
```

This function will extract the signature field of a CRL.

crl : should contain a gnutls_x509_crl_t structure

sig : a pointer where the signature part will be copied (may be null).

sizeof_sig : initially holds the size of *sig*

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value. and a negative value on error.

gnutls_x509_crl_get_version ()

```
int                 gnutls_x509_crl_get_version         (gnutls_x509_crl_t crl);
```

This function will return the version of the specified CRL.

crl : should contain a gnutls_x509_crl_t structure

Returns : The version number, or a negative value on error.

gnutls_x509_crl_get_this_update ()

```
time_t              gnutls_x509_crl_get_this_update     (gnutls_x509_crl_t crl);
```

This function will return the time this CRL was issued.

crl : should contain a gnutls_x509_crl_t structure

Returns : when the CRL was issued, or (time_t)-1 on error.

gnutls_x509_crl_get_next_update ()

```
time_t              gnutls_x509_crl_get_next_update     (gnutls_x509_crl_t crl);
```

This function will return the time the next CRL will be issued. This field is optional in a CRL so it might be normal to get an error instead.

crl : should contain a gnutls_x509_crl_t structure

Returns : when the next CRL will be issued, or (time_t)-1 on error.

gnutls_x509_crl_get_crt_count ()

```
int                 gnutls_x509_crl_get_crt_count       (gnutls_x509_crl_t crl);
```

This function will return the number of revoked certificates in the given CRL.

crl : should contain a gnutls_x509_crl_t structure

Returns : number of certificates, a negative value on failure.

gnutls_x509_crl_get_crt_serial ()

```
int                 gnutls_x509_crl_get_crt_serial      (gnutls_x509_crl_t crl,
                                                         int indx,
                                                         unsigned char *serial,
                                                         size_t *serial_size,
                                                         time_t *t);
```

This function will retrieve the serial number of the specified, by the index, revoked certificate.

crl : should contain a gnutls_x509_crl_t structure

indx : the index of the certificate to extract (starting from 0)

serial : where the serial number will be copied

serial_size : initially holds the size of serial

t : if non null, will hold the time this certificate was revoked

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value. and a negative value on error.

gnutls_x509_crl_get_certificate_count

```
#define gnutls_x509_crl_get_certificate_count gnutls_x509_crl_get_crt_count
```

gnutls_x509_crl_get_certificate

```
#define gnutls_x509_crl_get_certificate gnutls_x509_crl_get_crt_serial
```

gnutls_x509_crl_check_issuer ()

```
int                 gnutls_x509_crl_check_issuer        (gnutls_x509_crl_t crl,
                                                         gnutls_x509_crt_t issuer);
```

This function will check if the given CRL was issued by the given issuer certificate. It will return true (1) if the given CRL was issued by the given issuer, and false (0) if not.

crl : is the CRL to be checked

issuer : is the certificate of a possible issuer

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_x509_crl_set_version ()

```
int                 gnutls_x509_crl_set_version         (gnutls_x509_crl_t crl,
                                                         unsigned int version);
```

This function will set the version of the CRL. This must be one for CRL version 1, and so on. The CRLs generated by gnutls should have a version number of 2.

crl : should contain a gnutls_x509_crl_t structure

version : holds the version number. For CRLv1 crls must be 1.

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_x509_crl_sign ()

```
int                 gnutls_x509_crl_sign                (gnutls_x509_crl_t crl,
                                                         gnutls_x509_crt_t issuer,
                                                         gnutls_x509_privkey_t issuer_key);
```

This function is the same a gnutls_x509_crl_sign2() with no flags, and SHA1 as the hash algorithm.

crl : should contain a gnutls_x509_crl_t structure

issuer : is the certificate of the certificate issuer

issuer_key : holds the issuer's private key

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_x509_crl_sign2 ()

```
int                 gnutls_x509_crl_sign2               (gnutls_x509_crl_t crl,
                                                         gnutls_x509_crt_t issuer,
                                                         gnutls_x509_privkey_t issuer_key,
                                                         gnutls_digest_algorithm_t dig,
                                                         unsigned int flags);
```

This function will sign the CRL with the issuer's private key, and will copy the issuer's information into the CRL.

This must be the last step in a certificate CRL since all the previously set parameters are now signed.

crl : should contain a gnutls_x509_crl_t structure

issuer : is the certificate of the certificate issuer

issuer_key : holds the issuer's private key

dig : The message digest to use. GNUTLS_DIG_SHA1 is the safe choice unless you know what you're doing.

flags : must be 0

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_x509_crl_set_this_update ()

```
int                 gnutls_x509_crl_set_this_update     (gnutls_x509_crl_t crl,
                                                         time_t act_time);
```

This function will set the time this CRL was issued.

crl : should contain a gnutls_x509_crl_t structure

act_time : The actual time

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_x509_crl_set_next_update ()

```
int                 gnutls_x509_crl_set_next_update     (gnutls_x509_crl_t crl,
                                                         time_t exp_time);
```

This function will set the time this CRL will be updated.

crl : should contain a gnutls_x509_crl_t structure

exp_time : The actual time

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_x509_crl_set_crt_serial ()

```
int                 gnutls_x509_crl_set_crt_serial      (gnutls_x509_crl_t crl,
                                                         const void *serial,
                                                         size_t serial_size,
                                                         time_t revocation_time);
```

This function will set a revoked certificate's serial number to the CRL.

crl : should contain a gnutls_x509_crl_t structure

serial : The revoked certificate's serial number

serial_size : Holds the size of the serial field.

revocation_time : The time this certificate was revoked

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_x509_crl_set_crt ()

```
int                 gnutls_x509_crl_set_crt             (gnutls_x509_crl_t crl,
                                                         gnutls_x509_crt_t crt,
                                                         time_t revocation_time);
```

This function will set a revoked certificate's serial number to the CRL.

crl : should contain a gnutls_x509_crl_t structure

crt : a certificate of type gnutls_x509_crt_t with the revoked certificate

revocation_time : The time this certificate was revoked

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_x509_crl_get_authority_key_id ()

```
int                 gnutls_x509_crl_get_authority_key_id
                                                    (gnutls_x509_crl_t crl,
                                                     void *ret,
                                                     size_t *ret_size,
                                                     unsigned int *critical);
```

This function will return the CRL authority's key identifier. This is obtained by the X.509 Authority Key identifier extension field (2.5.29.35). Note that this function only returns the keyIdentifier field of the extension.

crl : should contain a gnutls_x509_crl_t structure

ret : The place where the identifier will be copied

ret_size : Holds the size of the result field.

critical : will be non zero if the extension is marked as critical (may be null)

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative value in case of an error.

Since 2.8.0

gnutls_x509_crl_get_number ()

```
int                 gnutls_x509_crl_get_number          (gnutls_x509_crl_t crl,
                                                     void *ret,
                                                     size_t *ret_size,
                                                     unsigned int *critical);
```

This function will return the CRL number extension. This is obtained by the CRL Number extension field (2.5.29.20).

crl : should contain a gnutls_x509_crl_t structure

ret : The place where the number will be copied

ret_size : Holds the size of the result field.

critical : will be non zero if the extension is marked as critical (may be null)

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative value in case of an error.

Since 2.8.0

gnutls_x509_crl_get_extension_oid ()

```
int                 gnutls_x509_crl_get_extension_oid   (gnutls_x509_crl_t crl,
                                                     int indx,
                                                     void *oid,
                                                     size_t *sizeof_oid);
```

This function will return the requested extension OID in the CRL. The extension OID will be stored as a string in the provided buffer.

crl : should contain a gnutls_x509_crl_t structure

indx : Specifies which extension OID to send, use zero to get the first one.

oid : a pointer to a structure to hold the OID (may be null)

sizeof_oid : initially holds the size of *oid*

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative value in case of an error. If your have reached the last extension available GNUTLS_E_REQUESTED_DATA_NOT_AVAILABLE will be returned.

Since 2.8.0

gnutls_x509_crl_get_extension_info ()

```
int                 gnutls_x509_crl_get_extension_info  (gnutls_x509_crl_t crl,
                                                         int indx,
                                                         void *oid,
                                                         size_t *sizeof_oid,
                                                         int *critical);
```

This function will return the requested extension OID in the CRL, and the critical flag for it. The extension OID will be stored as a string in the provided buffer. Use gnutls_x509_crl_get_extension_data() to extract the data.

If the buffer provided is not long enough to hold the output, then **sizeof_oid* is updated and GNUTLS_E_SHORT_MEMORY_BUFF will be returned.

crl : should contain a gnutls_x509_crl_t structure

indx : Specifies which extension OID to send, use zero to get the first one.

oid : a pointer to a structure to hold the OID

sizeof_oid : initially holds the maximum size of *oid*, on return holds actual size of *oid*.

critical : output variable with critical flag, may be NULL.

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative value in case of an error. If your have reached the last extension available GNUTLS_E_REQUESTED_DATA_NOT_AVAILABLE will be returned.

Since 2.8.0

gnutls_x509_crl_get_extension_data ()

```
int                 gnutls_x509_crl_get_extension_data  (gnutls_x509_crl_t crl,
                                                         int indx,
                                                         void *data,
                                                         size_t *sizeof_data);
```

This function will return the requested extension data in the CRL. The extension data will be stored as a string in the provided buffer.

Use gnutls_x509_crl_get_extension_info() to extract the OID and critical flag. Use gnutls_x509_crl_get_extension_info() instead, if you want to get data indexed by the extension OID rather than sequence.

crl : should contain a gnutls_x509_crl_t structure

indx : Specifies which extension OID to send. Use zero to get the first one.

data : a pointer to a structure to hold the data (may be null)

sizeof_data : initially holds the size of *oid*

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative value in case of an error. If your have reached the last extension available GNUTLS_E_REQUESTED_DATA_NOT_AVAILABLE will be returned.

Since 2.8.0

gnutls_x509_crl_set_authority_key_id ()

```
int                 gnutls_x509_crl_set_authority_key_id
                                                    (gnutls_x509_crl_t crl,
                                                     const void *id,
                                                     size_t id_size);
```

This function will set the CRL's authority key ID extension. Only the keyIdentifier field can be set with this function.

crl : a CRL of type gnutls_x509_crl_t

id : The key ID

id_size : Holds the size of the serial field.

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

Since 2.8.0

gnutls_x509_crl_set_number ()

```
int                 gnutls_x509_crl_set_number          (gnutls_x509_crl_t crl,
                                                         const void *nr,
                                                         size_t nr_size);
```

This function will set the CRL's number extension.

crl : a CRL of type gnutls_x509_crl_t

nr : The CRL number

nr_size : Holds the size of the nr field.

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

Since 2.8.0

struct gnutls_pkcs7_int

```
struct gnutls_pkcs7_int;
```

gnutls_pkcs7_t

```
  typedef struct gnutls_pkcs7_int *gnutls_pkcs7_t;
```

gnutls_pkcs7_init ()

```
int                 gnutls_pkcs7_init                   (gnutls_pkcs7_t *pkcs7);
```

This function will initialize a PKCS7 structure. PKCS7 structures usually contain lists of X.509 Certificates and X.509 Certificate revocation lists.

pkcs7 : The structure to be initialized

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_pkcs7_deinit ()

```
void                gnutls_pkcs7_deinit             (gnutls_pkcs7_t pkcs7);
```

This function will deinitialize a PKCS7 structure.

pkcs7 : The structure to be initialized

gnutls_pkcs7_import ()

```
int                 gnutls_pkcs7_import             (gnutls_pkcs7_t pkcs7,
                                                     const gnutls_datum_t *data,
                                                     gnutls_x509_crt_fmt_t format);
```

This function will convert the given DER or PEM encoded PKCS7 to the native gnutls_pkcs7_t format. The output will be stored in *pkcs7*.

If the PKCS7 is PEM encoded it should have a header of "PKCS7".

pkcs7 : The structure to store the parsed PKCS7.

data : The DER or PEM encoded PKCS7.

format : One of DER or PEM

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_pkcs7_export ()

```
int                 gnutls_pkcs7_export             (gnutls_pkcs7_t pkcs7,
                                                     gnutls_x509_crt_fmt_t format,
                                                     void *output_data,
                                                     size_t *output_data_size);
```

This function will export the pkcs7 structure to DER or PEM format.

If the buffer provided is not long enough to hold the output, then **output_data_size* is updated and GNUTLS_E_SHORT_MEMORY will be returned.

If the structure is PEM encoded, it will have a header of "BEGIN PKCS7".

pkcs7 : Holds the pkcs7 structure

format : the format of output params. One of PEM or DER.

output_data : will contain a structure PEM or DER encoded

output_data_size : holds the size of output_data (and will be replaced by the actual size of parameters)

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_pkcs7_get_crt_count ()

```
int                 gnutls_pkcs7_get_crt_count      (gnutls_pkcs7_t pkcs7);
```

This function will return the number of certifcates in the PKCS7 or RFC2630 certificate set.

pkcs7 : should contain a gnutls_pkcs7_t structure

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_pkcs7_get_crt_raw ()

```
int                 gnutls_pkcs7_get_crt_raw            (gnutls_pkcs7_t pkcs7,
                                                         int indx,
                                                         void *certificate,
                                                         size_t *certificate_size);
```

This function will return a certificate of the PKCS7 or RFC2630 certificate set.

After the last certificate has been read GNUTLS_E_REQUESTED_DATA_NOT_AVAILABLE will be returned.

pkcs7 : should contain a gnutls_pkcs7_t structure

indx : contains the index of the certificate to extract

certificate : the contents of the certificate will be copied there (may be null)

certificate_size : should hold the size of the certificate

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value. If the provided buffer is not long enough, then *certificate_size* is updated and GNUTLS_E_SHORT_MEMORY_BUFFER is returned.

gnutls_pkcs7_set_crt_raw ()

```
int                 gnutls_pkcs7_set_crt_raw            (gnutls_pkcs7_t pkcs7,
                                                         const gnutls_datum_t *crt);
```

This function will add a certificate to the PKCS7 or RFC2630 certificate set.

pkcs7 : should contain a gnutls_pkcs7_t structure

crt : the DER encoded certificate to be added

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_pkcs7_set_crt ()

```
int                 gnutls_pkcs7_set_crt                (gnutls_pkcs7_t pkcs7,
                                                         gnutls_x509_crt_t crt);
```

This function will add a parsed certificate to the PKCS7 or RFC2630 certificate set. This is a wrapper function over gnutls_pkcs7_set_crt_
.

pkcs7 : should contain a gnutls_pkcs7_t structure

crt : the certificate to be copied.

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_pkcs7_delete_crt ()

```
int                 gnutls_pkcs7_delete_crt             (gnutls_pkcs7_t pkcs7,
                                                         int indx);
```

This function will delete a certificate from a PKCS7 or RFC2630 certificate set. Index starts from 0. Returns 0 on success.

pkcs7 : should contain a gnutls_pkcs7_t structure

indx : the index of the certificate to delete

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_pkcs7_get_crl_raw ()

```
int                 gnutls_pkcs7_get_crl_raw            (gnutls_pkcs7_t pkcs7,
                                                         int indx,
                                                         void *crl,
                                                         size_t *crl_size);
```

This function will return a crl of the PKCS7 or RFC2630 crl set.

pkcs7 : should contain a gnutls_pkcs7_t structure

indx : contains the index of the crl to extract

crl : the contents of the crl will be copied there (may be null)

crl_size : should hold the size of the crl

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value. If the provided buffer is not long enough, then *crl_size* is updated and GNUTLS_E_SHORT_MEMORY_BUFFER is returned. After the last crl has been read GNUTLS_E_REQUESTED_DATA_NOT_AVAILABLE will be returned.

gnutls_pkcs7_get_crl_count ()

```
int                 gnutls_pkcs7_get_crl_count          (gnutls_pkcs7_t pkcs7);
```

This function will return the number of certifcates in the PKCS7 or RFC2630 crl set.

pkcs7 : should contain a gnutls_pkcs7_t structure

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_pkcs7_set_crl_raw ()

```
int                 gnutls_pkcs7_set_crl_raw            (gnutls_pkcs7_t pkcs7,
                                                         const gnutls_datum_t *crl);
```

This function will add a crl to the PKCS7 or RFC2630 crl set.

pkcs7 : should contain a gnutls_pkcs7_t structure

crl : the DER encoded crl to be added

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_pkcs7_set_crl ()

```
int                 gnutls_pkcs7_set_crl                (gnutls_pkcs7_t pkcs7,
                                                         gnutls_x509_crl_t crl);
```

This function will add a parsed CRL to the PKCS7 or RFC2630 crl set.

pkcs7 : should contain a gnutls_pkcs7_t structure

crl : the DER encoded crl to be added

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_pkcs7_delete_crl ()

```
int                 gnutls_pkcs7_delete_crl             (gnutls_pkcs7_t pkcs7,
                                                         int indx);
```

This function will delete a crl from a PKCS7 or RFC2630 crl set. Index starts from 0. Returns 0 on success.

pkcs7 : should contain a gnutls_pkcs7_t structure

indx : the index of the crl to delete

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

enum gnutls_certificate_verify_flags

```
typedef enum gnutls_certificate_verify_flags
{
  GNUTLS_VERIFY_DISABLE_CA_SIGN = 1,
  GNUTLS_VERIFY_ALLOW_X509_V1_CA_CRT = 2,
  GNUTLS_VERIFY_DO_NOT_ALLOW_SAME = 4,
  GNUTLS_VERIFY_ALLOW_ANY_X509_V1_CA_CRT = 8,
  GNUTLS_VERIFY_ALLOW_SIGN_RSA_MD2 = 16,
  GNUTLS_VERIFY_ALLOW_SIGN_RSA_MD5 = 32,
  GNUTLS_VERIFY_DISABLE_TIME_CHECKS = 64,
  GNUTLS_VERIFY_DISABLE_TRUSTED_TIME_CHECKS = 128,
} gnutls_certificate_verify_flags;
```

Enumeration of different certificate verify flags.

GNUTLS_VERIFY_DISABLE_CA_SIGN If set a signer does not have to be a certificate authority. This flag should normaly be disabled, unless you know what this means.

GNUTLS_VERIFY_ALLOW_X509_V1_CA_CRT Allow only trusted CA certificates that have version 1. This is safer than GNUTLS_VERIFY_ALLOW_ANY_X509_V1_CA_CRT, and should be used instead. That way only signers in your trusted list will be allowed to have certificates of version 1.

GNUTLS_VERIFY_DO_NOT_ALLOW_SAME If a certificate is not signed by anyone trusted but exists in the trusted CA list do not treat it as trusted.

GNUTLS_VERIFY_ALLOW_ANY_X509_V1_CA_CRT Allow CA certificates that have version 1 (both root and intermediate). This might be dangerous since those haven't the basicConstraints extension. Must be used in combination with GNUTLS_VERIFY_ALLOW_X509_V1_CA_CRT.

GNUTLS_VERIFY_ALLOW_SIGN_RSA_MD2 Allow certificates to be signed using the broken MD2 algorithm.

GNUTLS_VERIFY_ALLOW_SIGN_RSA_MD5 Allow certificates to be signed using the broken MD5 algorithm.

GNUTLS_VERIFY_DISABLE_TIME_CHECKS Disable checking of activation and expiration validity periods of certificate chains. Don't set this unless you understand the security implications.

GNUTLS_VERIFY_DISABLE_TRUSTED_TIME_CHECKS If set a signer in the trusted list is never checked for expiration or activation.

gnutls_x509_crt_check_issuer ()

```
int                 gnutls_x509_crt_check_issuer        (gnutls_x509_crt_t cert,
                                                         gnutls_x509_crt_t issuer);
```

This function will check if the given certificate was issued by the given issuer.

cert **:** is the certificate to be checked

issuer **:** is the certificate of a possible issuer

Returns **:** It will return true (1) if the given certificate is issued by the given issuer, and false (0) if not. A negative value is returned in case of an error.

gnutls_x509_crt_list_verify ()

```
int                 gnutls_x509_crt_list_verify         (const gnutls_x509_crt_t *cert_list ↩
    ,
                                                         int cert_list_length,
                                                         const gnutls_x509_crt_t *CA_list,
                                                         int CA_list_length,
                                                         const gnutls_x509_crl_t *CRL_list,
                                                         int CRL_list_length,
                                                         unsigned int flags,
                                                         unsigned int *verify);
```

This function will try to verify the given certificate list and return its status. If no flags are specified (0), this function will use the basicConstraints (2.5.29.19) PKIX extension. This means that only a certificate authority is allowed to sign a certificate.

You must also check the peer's name in order to check if the verified certificate belongs to the actual peer.

The certificate verification output will be put in *verify* and will be one or more of the gnutls_certificate_status_t enumerated elements bitwise or'd. For a more detailed verification status use gnutls_x509_crt_verify() per list element.

GNUTLS_CERT_INVALID: the certificate chain is not valid.

GNUTLS_CERT_REVOKED: a certificate in the chain has been revoked.

cert_list **:** is the certificate list to be verified

cert_list_length **:** holds the number of certificate in cert_list

CA_list **:** is the CA list which will be used in verification

CA_list_length **:** holds the number of CA certificate in CA_list

CRL_list **:** holds a list of CRLs.

CRL_list_length **:** the length of CRL list.

flags **:** Flags that may be used to change the verification algorithm. Use OR of the gnutls_certificate_verify_flags enumerations.

verify **:** will hold the certificate verification output.

Returns **:** On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_x509_crt_verify ()

```
int                 gnutls_x509_crt_verify              (gnutls_x509_crt_t cert,
                                                         const gnutls_x509_crt_t *CA_list,
                                                         int CA_list_length,
                                                         unsigned int flags,
                                                         unsigned int *verify);
```

This function will try to verify the given certificate and return its status.

cert **:** is the certificate to be verified

CA_list **:** is one certificate that is considered to be trusted one

CA_list_length : holds the number of CA certificate in CA_list

flags : Flags that may be used to change the verification algorithm. Use OR of the gnutls_certificate_verify_flags enumerations.

verify : will hold the certificate verification output.

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_x509_crl_verify ()

```
int                 gnutls_x509_crl_verify              (gnutls_x509_crl_t crl,
                                                         const gnutls_x509_crt_t *CA_list,
                                                         int CA_list_length,
                                                         unsigned int flags,
                                                         unsigned int *verify);
```

This function will try to verify the given crl and return its status. See gnutls_x509_crt_list_verify() for a detailed description of return values.

crl : is the crl to be verified

CA_list : is a certificate list that is considered to be trusted one

CA_list_length : holds the number of CA certificates in CA_list

flags : Flags that may be used to change the verification algorithm. Use OR of the gnutls_certificate_verify_flags enumerations.

verify : will hold the crl verification output.

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_x509_crt_check_revocation ()

```
int                 gnutls_x509_crt_check_revocation    (gnutls_x509_crt_t cert,
                                                         const gnutls_x509_crl_t *crl_list,
                                                         int crl_list_length);
```

This function will return check if the given certificate is revoked. It is assumed that the CRLs have been verified before.

cert : should contain a gnutls_x509_crt_t structure

crl_list : should contain a list of gnutls_x509_crl_t structures

crl_list_length : the length of the crl_list

Returns : 0 if the certificate is NOT revoked, and 1 if it is. A negative value is returned on error.

gnutls_x509_crt_get_fingerprint ()

```
int                 gnutls_x509_crt_get_fingerprint     (gnutls_x509_crt_t cert,
                                                         gnutls_digest_algorithm_t algo,
                                                         void *buf,
                                                         size_t *sizeof_buf);
```

This function will calculate and copy the certificate's fingerprint in the provided buffer.

If the buffer is null then only the size will be filled.

cert : should contain a gnutls_x509_crt_t structure

algo : is a digest algorithm

buf : a pointer to a structure to hold the fingerprint (may be null)

sizeof_buf : initially holds the size of *buf*

Returns : GNUTLS_E_SHORT_MEMORY_BUFFER if the provided buffer is not long enough, and in that case the *sizeof_buf will be updated with the required size. On success 0 is returned.

gnutls_x509_crt_get_key_purpose_oid ()

```
int                 gnutls_x509_crt_get_key_purpose_oid (gnutls_x509_crt_t cert,
                                                         int indx,
                                                         void *oid,
                                                         size_t *sizeof_oid,
                                                         unsigned int *critical);
```

This function will extract the key purpose OIDs of the Certificate specified by the given index. These are stored in the Extended Key Usage extension (2.5.29.37) See the GNUTLS_KP_* definitions for human readable names.

If *oid* is null then only the size will be filled.

cert : should contain a gnutls_x509_crt_t structure

indx : This specifies which OID to return. Use zero to get the first one.

oid : a pointer to a buffer to hold the OID (may be null)

sizeof_oid : initially holds the size of *oid*

critical : output flag to indicate criticality of extension

Returns : GNUTLS_E_SHORT_MEMORY_BUFFER if the provided buffer is not long enough, and in that case the *sizeof_oid will be updated with the required size. On success 0 is returned.

gnutls_x509_crt_set_key_purpose_oid ()

```
int                 gnutls_x509_crt_set_key_purpose_oid (gnutls_x509_crt_t cert,
                                                         const void *oid,
                                                         unsigned int critical);
```

This function will set the key purpose OIDs of the Certificate. These are stored in the Extended Key Usage extension (2.5.29.37) See the GNUTLS_KP_* definitions for human readable names.

Subsequent calls to this function will append OIDs to the OID list.

cert : a certificate of type gnutls_x509_crt_t

oid : a pointer to a null terminated string that holds the OID

critical : Whether this extension will be critical or not

Returns : On success, GNUTLS_E_SUCCESS (zero) is returned, otherwise an error code is returned.

enum gnutls_pkcs_encrypt_flags_t

```
typedef enum gnutls_pkcs_encrypt_flags_t
{
  GNUTLS_PKCS_PLAIN = 1,
  GNUTLS_PKCS8_PLAIN = GNUTLS_PKCS_PLAIN,
  GNUTLS_PKCS_USE_PKCS12_3DES = 2,
  GNUTLS_PKCS8_USE_PKCS12_3DES = GNUTLS_PKCS_USE_PKCS12_3DES,
  GNUTLS_PKCS_USE_PKCS12_ARCFOUR = 4,
  GNUTLS_PKCS8_USE_PKCS12_ARCFOUR = GNUTLS_PKCS_USE_PKCS12_ARCFOUR,
  GNUTLS_PKCS_USE_PKCS12_RC2_40 = 8,
  GNUTLS_PKCS8_USE_PKCS12_RC2_40 = GNUTLS_PKCS_USE_PKCS12_RC2_40,
  GNUTLS_PKCS_USE_PBES2_3DES = 16,
  GNUTLS_PKCS_USE_PBES2_AES_128 = 32,
  GNUTLS_PKCS_USE_PBES2_AES_192 = 64,
  GNUTLS_PKCS_USE_PBES2_AES_256 = 128,
} gnutls_pkcs_encrypt_flags_t;
```

Enumeration of different PKCS encryption flags.

GNUTLS_PKCS_PLAIN Unencrypted private key.

GNUTLS_PKCS8_PLAIN Same as GNUTLS_PKCS_PLAIN.

GNUTLS_PKCS_USE_PKCS12_3DES PKCS-12 3DES.

GNUTLS_PKCS8_USE_PKCS12_3DES Same as GNUTLS_PKCS_USE_PKCS12_3DES.

GNUTLS_PKCS_USE_PKCS12_ARCFOUR PKCS-12 ARCFOUR.

GNUTLS_PKCS8_USE_PKCS12_ARCFOUR Same as GNUTLS_PKCS_USE_PKCS12_ARCFOUR.

GNUTLS_PKCS_USE_PKCS12_RC2_40 PKCS-12 RC2-40.

GNUTLS_PKCS8_USE_PKCS12_RC2_40 Same as GNUTLS_PKCS_USE_PKCS12_RC2_40.

GNUTLS_PKCS_USE_PBES2_3DES PBES2 3DES.

GNUTLS_PKCS_USE_PBES2_AES_128 PBES2 AES-128.

GNUTLS_PKCS_USE_PBES2_AES_192 PBES2 AES-192.

GNUTLS_PKCS_USE_PBES2_AES_256 PBES2 AES-256.

gnutls_x509_privkey_init ()

```
int                 gnutls_x509_privkey_init            (gnutls_x509_privkey_t *key);
```

This function will initialize an private key structure.

key : The structure to be initialized

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_x509_privkey_deinit ()

```
void                gnutls_x509_privkey_deinit          (gnutls_x509_privkey_t key);
```

This function will deinitialize a private key structure.

key : The structure to be initialized

gnutls_x509_privkey_cpy ()

```
int                 gnutls_x509_privkey_cpy             (gnutls_x509_privkey_t dst,
                                                         gnutls_x509_privkey_t src);
```

This function will copy a private key from source to destination key.

dst : The destination key, which should be initialized.

src : The source key

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_x509_privkey_import ()

```
int                 gnutls_x509_privkey_import          (gnutls_x509_privkey_t key,
                                                         const gnutls_datum_t *data,
                                                         gnutls_x509_crt_fmt_t format);
```

This function will convert the given DER or PEM encoded key to the native gnutls_x509_privkey_t format. The output will be stored in *key* .

If the key is PEM encoded it should have a header of "RSA PRIVATE KEY", or "DSA PRIVATE KEY".

key : The structure to store the parsed key

data : The DER or PEM encoded certificate.

format : One of DER or PEM

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_x509_privkey_import_pkcs8 ()

```
int                 gnutls_x509_privkey_import_pkcs8    (gnutls_x509_privkey_t key,
                                                         const gnutls_datum_t *data,
                                                         gnutls_x509_crt_fmt_t format,
                                                         const char *password,
                                                         unsigned int flags);
```

This function will convert the given DER or PEM encoded PKCS8 2.0 encrypted key to the native gnutls_x509_privkey_t format. The output will be stored in *key*. Both RSA and DSA keys can be imported, and flags can only be used to indicate an unencrypted key.

The *password* can be either ASCII or UTF-8 in the default PBES2 encryption schemas, or ASCII for the PKCS12 schemas.

If the Certificate is PEM encoded it should have a header of "ENCRYPTED PRIVATE KEY", or "PRIVATE KEY". You only need to specify the flags if the key is DER encoded, since in that case the encryption status cannot be auto-detected.

key : The structure to store the parsed key

data : The DER or PEM encoded key.

format : One of DER or PEM

password : the password to decrypt the key (if it is encrypted).

flags : 0 if encrypted or GNUTLS_PKCS_PLAIN if not encrypted.

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_x509_privkey_import_rsa_raw ()

```
int                 gnutls_x509_privkey_import_rsa_raw  (gnutls_x509_privkey_t key,
                                                         const gnutls_datum_t *m,
                                                         const gnutls_datum_t *e,
                                                         const gnutls_datum_t *d,
                                                         const gnutls_datum_t *p,
                                                         const gnutls_datum_t *q,
                                                         const gnutls_datum_t *u);
```

This function will convert the given RSA raw parameters to the native gnutls_x509_privkey_t format. The output will be stored in *key*.

key : The structure to store the parsed key

m : holds the modulus

e : holds the public exponent

d : holds the private exponent

p : holds the first prime (p)

q : holds the second prime (q)

u : holds the coefficient

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_x509_privkey_fix ()

```
int                 gnutls_x509_privkey_fix             (gnutls_x509_privkey_t key);
```

This function will recalculate the secondary parameters in a key. In RSA keys, this can be the coefficient and exponent1,2.

key : Holds the key

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_x509_privkey_export_dsa_raw ()

```
int                 gnutls_x509_privkey_export_dsa_raw  (gnutls_x509_privkey_t key,
                                                         gnutls_datum_t *p,
                                                         gnutls_datum_t *q,
                                                         gnutls_datum_t *g,
                                                         gnutls_datum_t *y,
                                                         gnutls_datum_t *x);
```

This function will export the DSA private key's parameters found in the given structure. The new parameters will be allocated using gnutls_malloc() and will be stored in the appropriate datum.

key : a structure that holds the DSA parameters

p : will hold the p

q : will hold the q

g : will hold the g

y : will hold the y

x : will hold the x

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_x509_privkey_import_dsa_raw ()

```
int                 gnutls_x509_privkey_import_dsa_raw  (gnutls_x509_privkey_t key,
                                                         const gnutls_datum_t *p,
                                                         const gnutls_datum_t *q,
                                                         const gnutls_datum_t *g,
                                                         const gnutls_datum_t *y,
                                                         const gnutls_datum_t *x);
```

This function will convert the given DSA raw parameters to the native gnutls_x509_privkey_t format. The output will be stored in *key*.

key : The structure to store the parsed key

p : holds the p

q : holds the q

g : holds the g

y : holds the y

x : holds the x

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_x509_privkey_get_pk_algorithm ()

```
int                 gnutls_x509_privkey_get_pk_algorithm
                                                        (gnutls_x509_privkey_t key);
```

This function will return the public key algorithm of a private key.

key : should contain a gnutls_x509_privkey_t structure

Returns : a member of the gnutls_pk_algorithm_t enumeration on success, or a negative value on error.

gnutls_x509_privkey_get_key_id ()

```
int                 gnutls_x509_privkey_get_key_id      (gnutls_x509_privkey_t key,
                                                         unsigned int flags,
                                                         unsigned char *output_data,
                                                         size_t *output_data_size);
```

This function will return a unique ID the depends on the public key parameters. This ID can be used in checking whether a certificate corresponds to the given key.

If the buffer provided is not long enough to hold the output, then **output_data_size* is updated and GNUTLS_E_SHORT_MEMORY will be returned. The output will normally be a SHA-1 hash output, which is 20 bytes.

key : Holds the key

flags : should be 0 for now

output_data : will contain the key ID

output_data_size : holds the size of output_data (and will be replaced by the actual size of parameters)

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_x509_privkey_generate ()

```
int                 gnutls_x509_privkey_generate        (gnutls_x509_privkey_t key,
                                                         gnutls_pk_algorithm_t algo,
                                                         unsigned int bits,
                                                         unsigned int flags);
```

This function will generate a random private key. Note that this function must be called on an empty private key.

key : should contain a gnutls_x509_privkey_t structure

algo : is one of RSA or DSA.

bits : the size of the modulus

flags : unused for now. Must be 0.

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_x509_privkey_export ()

```
int                 gnutls_x509_privkey_export          (gnutls_x509_privkey_t key,
                                                         gnutls_x509_crt_fmt_t format,
                                                         void *output_data,
                                                         size_t *output_data_size);
```

This function will export the private key to a PKCS1 structure for RSA keys, or an integer sequence for DSA keys. The DSA keys are in the same format with the parameters used by openssl.

If the buffer provided is not long enough to hold the output, then *`output_data_size` is updated and GNUTLS_E_SHORT_MEMORY will be returned.

If the structure is PEM encoded, it will have a header of "BEGIN RSA PRIVATE KEY".

key : Holds the key

format : the format of output params. One of PEM or DER.

output_data : will contain a private key PEM or DER encoded

output_data_size : holds the size of output_data (and will be replaced by the actual size of parameters)

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_x509_privkey_export_pkcs8 ()

```
int                 gnutls_x509_privkey_export_pkcs8    (gnutls_x509_privkey_t key,
                                                         gnutls_x509_crt_fmt_t format,
                                                         const char *password,
                                                         unsigned int flags,
                                                         void *output_data,
                                                         size_t *output_data_size);
```

This function will export the private key to a PKCS8 structure. Both RSA and DSA keys can be exported. For DSA keys we use PKCS 11 definitions. If the flags do not specify the encryption cipher, then the default 3DES (PBES2) will be used.

The `password` can be either ASCII or UTF-8 in the default PBES2 encryption schemas, or ASCII for the PKCS12 schemas.

If the buffer provided is not long enough to hold the output, then *output_data_size is updated and GNUTLS_E_SHORT_MEMORY_BU will be returned.

If the structure is PEM encoded, it will have a header of "BEGIN ENCRYPTED PRIVATE KEY" or "BEGIN PRIVATE KEY" if encryption is not used.

key : Holds the key

format : the format of output params. One of PEM or DER.

password : the password that will be used to encrypt the key.

flags : an ORed sequence of gnutls_pkcs_encrypt_flags_t

output_data : will contain a private key PEM or DER encoded

output_data_size : holds the size of output_data (and will be replaced by the actual size of parameters)

Returns : In case of failure a negative value will be returned, and 0 on success.

gnutls_x509_privkey_export_rsa_raw ()

```
int                 gnutls_x509_privkey_export_rsa_raw  (gnutls_x509_privkey_t key,
                                                         gnutls_datum_t *m,
                                                         gnutls_datum_t *e,
                                                         gnutls_datum_t *d,
                                                         gnutls_datum_t *p,
                                                         gnutls_datum_t *q,
                                                         gnutls_datum_t *u);
```

This function will export the RSA private key's parameters found in the given structure. The new parameters will be allocated using gnutls_malloc() and will be stored in the appropriate datum.

key : a structure that holds the rsa parameters

m : will hold the modulus

e : will hold the public exponent

d : will hold the private exponent

p : will hold the first prime (p)

q : will hold the second prime (q)

u : will hold the coefficient

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_x509_privkey_sign_data ()

```
int                 gnutls_x509_privkey_sign_data       (gnutls_x509_privkey_t key,
                                                         gnutls_digest_algorithm_t digest,
                                                         unsigned int flags,
                                                         const gnutls_datum_t *data,
                                                         void *signature,
                                                         size_t *signature_size);
```

This function will sign the given data using a signature algorithm supported by the private key. Signature algorithms are always used together with a hash functions. Different hash functions may be used for the RSA algorithm, but only SHA-1 for the DSA keys.

If the buffer provided is not long enough to hold the output, then *`signature_size` is updated and GNUTLS_E_SHORT_MEMORY_B will be returned.

key : Holds the key

digest : should be MD5 or SHAx. May be ignored.

flags : should be 0 for now

data : holds the data to be signed

signature : will contain the signature

signature_size : holds the size of signature (and will be replaced by the new size)

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_x509_privkey_verify_data ()

```
int                 gnutls_x509_privkey_verify_data     (gnutls_x509_privkey_t key,
                                                         unsigned int flags,
                                                         const gnutls_datum_t *data,
                                                         const gnutls_datum_t *signature);
```

This function will verify the given signed data, using the parameters in the private key.

key : Holds the key

flags : should be 0 for now

data : holds the data to be signed

signature : contains the signature

Returns : In case of a verification failure 0 is returned, and 1 on success.

gnutls_x509_crt_verify_data ()

```
int                 gnutls_x509_crt_verify_data         (gnutls_x509_crt_t crt,
                                                         unsigned int flags,
                                                         const gnutls_datum_t *data,
                                                         const gnutls_datum_t *signature);
```

This function will verify the given signed data, using the parameters from the certificate.

crt : Holds the certificate

flags : should be 0 for now

data : holds the data to be signed

signature : contains the signature

Returns : In case of a verification failure 0 is returned, and 1 on success.

gnutls_x509_crt_verify_hash ()

```
int                 gnutls_x509_crt_verify_hash         (gnutls_x509_crt_t crt,
                                                         unsigned int flags,
                                                         const gnutls_datum_t *hash,
                                                         const gnutls_datum_t *signature);
```

This function will verify the given signed digest, using the parameters from the certificate.

crt : Holds the certificate

flags : should be 0 for now

hash : holds the hash digest to be verified

signature : contains the signature

Returns : In case of a verification failure 0 is returned, and 1 on success.

gnutls_x509_crt_get_verify_algorithm ()

```
int                 gnutls_x509_crt_get_verify_algorithm
                                                    (gnutls_x509_crt_t crt,
                                                     const gnutls_datum_t *signature,
                                                     gnutls_digest_algorithm_t *hash);
```

This function will read the certifcate and the signed data to determine the hash algorithm used to generate the signature.

crt : Holds the certificate

signature : contains the signature

hash : The result of the call with the hash algorithm used for signature

Returns : the 0 if the hash algorithm is found. A negative value is returned on error.

Since 2.8.0

gnutls_x509_privkey_sign_hash ()

```
int                 gnutls_x509_privkey_sign_hash       (gnutls_x509_privkey_t key,
                                                         const gnutls_datum_t *hash,
                                                         gnutls_datum_t *signature);
```

This function will sign the given hash using the private key. Do not use this function directly unless you know what it is. Typical signing requires the data to be hashed and stored in special formats (e.g. BER Digest-Info for RSA).

key : Holds the key

hash : holds the data to be signed

signature : will contain newly allocated signature

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

struct gnutls_x509_crq_int

```
struct gnutls_x509_crq_int;
```

gnutls_x509_crq_t

```
  typedef struct gnutls_x509_crq_int *gnutls_x509_crq_t;
```

gnutls_x509_crq_print ()

```
int                 gnutls_x509_crq_print               (gnutls_x509_crq_t crq,
                                                         gnutls_certificate_print_formats_t ↩
                                                              format,
                                                         gnutls_datum_t *out);
```

This function will pretty print a certificate request, suitable for display to a human.

The output *out* needs to be deallocate using gnutls_free().

crq : The structure to be printed

format : Indicate the format to use

out : Newly allocated datum with zero terminated string.

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

Since 2.8.0

gnutls_x509_crq_init ()

```
int                 gnutls_x509_crq_init                (gnutls_x509_crq_t *crq);
```

This function will initialize a PKCS10 certificate request structure.

crq : The structure to be initialized

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_x509_crq_deinit ()

```
void                gnutls_x509_crq_deinit              (gnutls_x509_crq_t crq);
```

This function will deinitialize a PKCS10 certificate request structure.

crq : The structure to be initialized

gnutls_x509_crq_import ()

```
int                 gnutls_x509_crq_import              (gnutls_x509_crq_t crq,
                                                         const gnutls_datum_t *data,
                                                         gnutls_x509_crt_fmt_t format);
```

This function will convert the given DER or PEM encoded certificate request to a gnutls_x509_crq_t structure. The output will be stored in *crq*.

If the Certificate is PEM encoded it should have a header of "NEW CERTIFICATE REQUEST".

crq : The structure to store the parsed certificate request.

data : The DER or PEM encoded certificate.

format : One of DER or PEM

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_x509_crq_get_dn ()

```
int                 gnutls_x509_crq_get_dn              (gnutls_x509_crq_t crq,
                                                         char *buf,
                                                         size_t *sizeof_buf);
```

This function will copy the name of the Certificate request subject to the provided buffer. The name will be in the form "C=xxxx,O=yyyy,CN=zzzz" as described in RFC 2253. The output string *buf* will be ASCII or UTF-8 encoded, depending on the certificate data.

crq : should contain a gnutls_x509_crq_t structure

buf : a pointer to a structure to hold the name (may be NULL)

sizeof_buf : initially holds the size of *buf*

Returns : GNUTLS_E_SHORT_MEMORY_BUFFER if the provided buffer is not long enough, and in that case the *sizeof_buf will be updated with the required size. On success 0 is returned.

gnutls_x509_crq_get_dn_oid ()

```
int                 gnutls_x509_crq_get_dn_oid          (gnutls_x509_crq_t crq,
                                                         int indx,
                                                         void *oid,
                                                         size_t *sizeof_oid);
```

This function will extract the requested OID of the name of the certificate request subject, specified by the given index.

crq : should contain a gnutls_x509_crq_t structure

indx : Specifies which DN OID to send. Use zero to get the first one.

oid : a pointer to a structure to hold the name (may be NULL)

sizeof_oid : initially holds the size of *oid*

Returns : GNUTLS_E_SHORT_MEMORY_BUFFER if the provided buffer is not long enough, and in that case the *sizeof_oid will be updated with the required size. On success 0 is returned.

gnutls_x509_crq_get_dn_by_oid ()

```
int                 gnutls_x509_crq_get_dn_by_oid       (gnutls_x509_crq_t crq,
                                                         const char *oid,
                                                         int indx,
                                                         unsigned int raw_flag,
                                                         void *buf,
                                                         size_t *sizeof_buf);
```

This function will extract the part of the name of the Certificate request subject, specified by the given OID. The output will be encoded as described in RFC2253. The output string will be ASCII or UTF-8 encoded, depending on the certificate data.

Some helper macros with popular OIDs can be found in gnutls/x509.h If raw flag is zero, this function will only return known OIDs as text. Other OIDs will be DER encoded, as described in RFC2253 -- in hex format with a '#' prefix. You can check about known OIDs using gnutls_x509_dn_oid_known().

crq : should contain a gnutls_x509_crq_t structure

oid : holds an Object Identified in null terminated string

***indx*:** In case multiple same OIDs exist in the RDN, this specifies which to send. Use zero to get the first one.

***raw_flag*:** If non zero returns the raw DER data of the DN part.

***buf*:** a pointer to a structure to hold the name (may be NULL)

***sizeof_buf*:** initially holds the size of *buf*

***Returns*:** GNUTLS_E_SHORT_MEMORY_BUFFER if the provided buffer is not long enough, and in that case the **sizeof_buf* will be updated with the required size. On success 0 is returned.

gnutls_x509_crq_set_dn_by_oid ()

```
int                 gnutls_x509_crq_set_dn_by_oid       (gnutls_x509_crq_t crq,
                                                         const char *oid,
                                                         unsigned int raw_flag,
                                                         const void *data,
                                                         unsigned int sizeof_data);
```

This function will set the part of the name of the Certificate request subject, specified by the given OID. The input string should be ASCII or UTF-8 encoded.

Some helper macros with popular OIDs can be found in gnutls/x509.h With this function you can only set the known OIDs. You can test for known OIDs using gnutls_x509_dn_oid_known(). For OIDs that are not known (by gnutls) you should properly DER encode your data, and call this function with raw_flag set.

***crq*:** should contain a gnutls_x509_crq_t structure

***oid*:** holds an Object Identifier in a zero-terminated string

***raw_flag*:** must be 0, or 1 if the data are DER encoded

***data*:** a pointer to the input data

***sizeof_data*:** holds the size of *data*

***Returns*:** On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_x509_crq_set_version ()

```
int                 gnutls_x509_crq_set_version         (gnutls_x509_crq_t crq,
                                                         unsigned int version);
```

This function will set the version of the certificate request. For version 1 requests this must be one.

***crq*:** should contain a gnutls_x509_crq_t structure

***version*:** holds the version number, for v1 Requests must be 1

***Returns*:** On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_x509_crq_get_version ()

```
int                 gnutls_x509_crq_get_version         (gnutls_x509_crq_t crq);
```

This function will return the version of the specified Certificate request.

***crq*:** should contain a gnutls_x509_crq_t structure

***Returns*:** version of certificate request, or a negative value on error.

gnutls_x509_crq_set_key ()

```
int                 gnutls_x509_crq_set_key             (gnutls_x509_crq_t crq,
                                                         gnutls_x509_privkey_t key);
```

This function will set the public parameters from the given private key to the request. Only RSA keys are currently supported.

crq : should contain a gnutls_x509_crq_t structure

key : holds a private key

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_x509_crq_sign2 ()

```
int                 gnutls_x509_crq_sign2               (gnutls_x509_crq_t crq,
                                                         gnutls_x509_privkey_t key,
                                                         gnutls_digest_algorithm_t dig,
                                                         unsigned int flags);
```

This function will sign the certificate request with a private key. This must be the same key as the one used in gnutls_x509_crt_set_key() since a certificate request is self signed.

This must be the last step in a certificate request generation since all the previously set parameters are now signed.

crq : should contain a gnutls_x509_crq_t structure

key : holds a private key

dig : The message digest to use, i.e., GNUTLS_DIG_SHA1

flags : must be 0

Returns : GNUTLS_E_SUCCESS on success, otherwise an error. GNUTLS_E_ASN1_VALUE_NOT_FOUND is returned if you didn't set all information in the certificate request (e.g., the version using gnutls_x509_crq_set_version()).

gnutls_x509_crq_sign ()

```
int                 gnutls_x509_crq_sign                (gnutls_x509_crq_t crq,
                                                         gnutls_x509_privkey_t key);
```

This function is the same a gnutls_x509_crq_sign2() with no flags, and SHA1 as the hash algorithm.

crq : should contain a gnutls_x509_crq_t structure

key : holds a private key

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_x509_crq_set_challenge_password ()

```
int                 gnutls_x509_crq_set_challenge_password
                                                        (gnutls_x509_crq_t crq,
                                                         const char *pass);
```

This function will set a challenge password to be used when revoking the request.

crq : should contain a gnutls_x509_crq_t structure

pass : holds a zero-terminated password

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_x509_crq_get_challenge_password ()

```
int                 gnutls_x509_crq_get_challenge_password
                                                    (gnutls_x509_crq_t crq,
                                                     char *pass,
                                                     size_t *sizeof_pass);
```

This function will return the challenge password in the request. The challenge password is intended to be used for requesting a revocation of the certificate.

crq : should contain a gnutls_x509_crq_t structure

pass : will hold a zero-terminated password string

sizeof_pass : Initially holds the size of *pass*.

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_x509_crq_set_attribute_by_oid ()

```
int                 gnutls_x509_crq_set_attribute_by_oid
                                                    (gnutls_x509_crq_t crq,
                                                     const char *oid,
                                                     void *buf,
                                                     size_t sizeof_buf);
```

This function will set the attribute in the certificate request specified by the given Object ID. The attribute must be be DER encoded.

crq : should contain a gnutls_x509_crq_t structure

oid : holds an Object Identified in zero-terminated string

buf : a pointer to a structure that holds the attribute data

sizeof_buf : holds the size of *buf*

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_x509_crq_get_attribute_by_oid ()

```
int                 gnutls_x509_crq_get_attribute_by_oid
                                                    (gnutls_x509_crq_t crq,
                                                     const char *oid,
                                                     int indx,
                                                     void *buf,
                                                     size_t *sizeof_buf);
```

This function will return the attribute in the certificate request specified by the given Object ID. The attribute will be DER encoded.

crq : should contain a gnutls_x509_crq_t structure

oid : holds an Object Identified in zero-terminated string

indx : In case multiple same OIDs exist in the attribute list, this specifies which to send, use zero to get the first one

buf : a pointer to a structure to hold the attribute data (may be NULL)

sizeof_buf : initially holds the size of *buf*

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_x509_crq_export ()

```
int                 gnutls_x509_crq_export              (gnutls_x509_crq_t crq,
                                                         gnutls_x509_crt_fmt_t format,
                                                         void *output_data,
                                                         size_t *output_data_size);
```

This function will export the certificate request to a PEM or DER encoded PKCS10 structure.

If the buffer provided is not long enough to hold the output, then GNUTLS_E_SHORT_MEMORY_BUFFER will be returned and *`output_data_size` will be updated.

If the structure is PEM encoded, it will have a header of "BEGIN NEW CERTIFICATE REQUEST".

`crq` **:** should contain a gnutls_x509_crq_t structure

`format` **:** the format of output params. One of PEM or DER.

`output_data` **:** will contain a certificate request PEM or DER encoded

`output_data_size` **:** holds the size of output_data (and will be replaced by the actual size of parameters)

Returns **:** In case of failure a negative value will be returned, and 0 on success.

gnutls_x509_crt_set_crq ()

```
int                 gnutls_x509_crt_set_crq             (gnutls_x509_crt_t crt,
                                                         gnutls_x509_crq_t crq);
```

This function will set the name and public parameters as well as the extensions from the given certificate request to the certificate. Only RSA keys are currently supported.

`crt` **:** a certificate of type gnutls_x509_crt_t

`crq` **:** holds a certificate request

Returns **:** On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_x509_crt_set_crq_extensions ()

```
int                 gnutls_x509_crt_set_crq_extensions  (gnutls_x509_crt_t crt,
                                                         gnutls_x509_crq_t crq);
```

This function will set extensions from the given request to the certificate.

`crt` **:** a certificate of type gnutls_x509_crt_t

`crq` **:** holds a certificate request

Returns **:** On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

Since 2.8.0

gnutls_x509_crq_set_key_rsa_raw ()

```
int                 gnutls_x509_crq_set_key_rsa_raw     (gnutls_x509_crq_t crq,
                                                         const gnutls_datum_t *m,
                                                         const gnutls_datum_t *e);
```

This function will set the public parameters from the given private key to the request. Only RSA keys are currently supported.

crq : should contain a gnutls_x509_crq_t structure

m : holds the modulus

e : holds the public exponent

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

Since 2.6.0

gnutls_x509_crq_set_subject_alt_name ()

```
int                 gnutls_x509_crq_set_subject_alt_name
                                                        (gnutls_x509_crq_t crq,
                                                         gnutls_x509_subject_alt_name_t nt,
                                                         const void *data,
                                                         unsigned int data_size,
                                                         unsigned int flags);
```

This function will set the subject alternative name certificate extension. It can set the following types:
&GNUTLS_SAN_DNSNAME: as a text string
&GNUTLS_SAN_RFC822NAME: as a text string
&GNUTLS_SAN_URI: as a text string
&GNUTLS_SAN_IPADDRESS: as a binary IP address (4 or 16 bytes)
Other values can be set as binary values with the proper DER encoding.

crq : a certificate request of type gnutls_x509_crq_t

nt : is one of the gnutls_x509_subject_alt_name_t enumerations

data : The data to be set

data_size : The size of data to be set

flags : GNUTLS_FSAN_SET to clear previous data or GNUTLS_FSAN_APPEND to append.

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

Since 2.8.0

gnutls_x509_crq_set_key_usage ()

```
int                 gnutls_x509_crq_set_key_usage       (gnutls_x509_crq_t crq,
                                                         unsigned int usage);
```

This function will set the keyUsage certificate extension.

crq : a certificate request of type gnutls_x509_crq_t

usage : an ORed sequence of the GNUTLS_KEY_* elements.

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

Since 2.8.0

gnutls_x509_crq_set_basic_constraints ()

```
int                 gnutls_x509_crq_set_basic_constraints
                                                    (gnutls_x509_crq_t crq,
                                                     unsigned int ca,
                                                     int pathLenConstraint);
```

This function will set the basicConstraints certificate extension.

crq : a certificate request of type gnutls_x509_crq_t

ca : true(1) or false(0) depending on the Certificate authority status.

pathLenConstraint : non-negative values indicate maximum length of path, and negative values indicate that the pathLenConstraints field should not be present.

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

Since 2.8.0

gnutls_x509_crq_set_key_purpose_oid ()

```
int                 gnutls_x509_crq_set_key_purpose_oid (gnutls_x509_crq_t crq,
                                                     const void *oid,
                                                     unsigned int critical);
```

This function will set the key purpose OIDs of the Certificate. These are stored in the Extended Key Usage extension (2.5.29.37) See the GNUTLS_KP_* definitions for human readable names.

Subsequent calls to this function will append OIDs to the OID list.

crq : a certificate of type gnutls_x509_crq_t

oid : a pointer to a zero-terminated string that holds the OID

critical : Whether this extension will be critical or not

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

Since 2.8.0

gnutls_x509_crq_get_key_purpose_oid ()

```
int                 gnutls_x509_crq_get_key_purpose_oid (gnutls_x509_crq_t crq,
                                                     int indx,
                                                     void *oid,
                                                     size_t *sizeof_oid,
                                                     unsigned int *critical);
```

This function will extract the key purpose OIDs of the Certificate specified by the given index. These are stored in the Extended Key Usage extension (2.5.29.37). See the GNUTLS_KP_* definitions for human readable names.

crq : should contain a gnutls_x509_crq_t structure

indx : This specifies which OID to return, use zero to get the first one

oid : a pointer to a buffer to hold the OID (may be NULL)

sizeof_oid : initially holds the size of *oid*

***critical*:** output variable with critical flag, may be NULL.

***Returns*:** GNUTLS_E_SHORT_MEMORY_BUFFER if the provided buffer is not long enough, and in that case the *sizeof_oid will be updated with the required size. On success 0 is returned.

Since 2.8.0

gnutls_x509_crq_get_extension_data ()

```
int                 gnutls_x509_crq_get_extension_data  (gnutls_x509_crq_t crq,
                                                         int indx,
                                                         void *data,
                                                         size_t *sizeof_data);
```

This function will return the requested extension data in the certificate. The extension data will be stored as a string in the provided buffer.

Use gnutls_x509_crq_get_extension_info() to extract the OID and critical flag. Use gnutls_x509_crq_get_extension_by_oid() instead, if you want to get data indexed by the extension OID rather than sequence.

***crq*:** should contain a gnutls_x509_crq_t structure

***indx*:** Specifies which extension OID to send. Use zero to get the first one.

***data*:** a pointer to a structure to hold the data (may be null)

***sizeof_data*:** initially holds the size of *oid*

***Returns*:** On success, GNUTLS_E_SUCCESS is returned, otherwise a negative value in case of an error. If your have reached the last extension available GNUTLS_E_REQUESTED_DATA_NOT_AVAILABLE will be returned.

Since 2.8.0

gnutls_x509_crq_get_extension_info ()

```
int                 gnutls_x509_crq_get_extension_info  (gnutls_x509_crq_t crq,
                                                         int indx,
                                                         void *oid,
                                                         size_t *sizeof_oid,
                                                         int *critical);
```

This function will return the requested extension OID in the certificate, and the critical flag for it. The extension OID will be stored as a string in the provided buffer. Use gnutls_x509_crq_get_extension_data() to extract the data.

If the buffer provided is not long enough to hold the output, then *sizeof_oid is updated and GNUTLS_E_SHORT_MEMORY_BUFF will be returned.

***crq*:** should contain a gnutls_x509_crq_t structure

***indx*:** Specifies which extension OID to send. Use zero to get the first one.

***oid*:** a pointer to a structure to hold the OID

***sizeof_oid*:** initially holds the maximum size of *oid*, on return holds actual size of *oid*.

***critical*:** output variable with critical flag, may be NULL.

***Returns*:** On success, GNUTLS_E_SUCCESS is returned, otherwise a negative value in case of an error. If your have reached the last extension available GNUTLS_E_REQUESTED_DATA_NOT_AVAILABLE will be returned.

Since 2.8.0

gnutls_x509_crq_get_attribute_data ()

```
int                 gnutls_x509_crq_get_attribute_data  (gnutls_x509_crq_t crq,
                                                         int indx,
                                                         void *data,
                                                         size_t *sizeof_data);
```

This function will return the requested attribute data in the certificate request. The attribute data will be stored as a string in the provided buffer.

Use gnutls_x509_crq_get_attribute_info() to extract the OID. Use gnutls_x509_crq_get_attribute_by_oid() instead, if you want to get data indexed by the attribute OID rather than sequence.

crq : should contain a gnutls_x509_crq_t structure

indx : Specifies which attribute OID to send. Use zero to get the first one.

data : a pointer to a structure to hold the data (may be null)

sizeof_data : initially holds the size of *oid*

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative value in case of an error. If your have reached the last extension available GNUTLS_E_REQUESTED_DATA_NOT_AVAILABLE will be returned.

Since 2.8.0

gnutls_x509_crq_get_attribute_info ()

```
int                 gnutls_x509_crq_get_attribute_info  (gnutls_x509_crq_t crq,
                                                         int indx,
                                                         void *oid,
                                                         size_t *sizeof_oid);
```

This function will return the requested attribute OID in the certificate, and the critical flag for it. The attribute OID will be stored as a string in the provided buffer. Use gnutls_x509_crq_get_attribute_data() to extract the data.

If the buffer provided is not long enough to hold the output, then **sizeof_oid* is updated and GNUTLS_E_SHORT_MEMORY_BUFF will be returned.

crq : should contain a gnutls_x509_crq_t structure

indx : Specifies which attribute OID to send. Use zero to get the first one.

oid : a pointer to a structure to hold the OID

sizeof_oid : initially holds the maximum size of *oid*, on return holds actual size of *oid*.

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative value in case of an error. If your have reached the last extension available GNUTLS_E_REQUESTED_DATA_NOT_AVAILABLE will be returned.

Since 2.8.0

gnutls_x509_crq_get_pk_algorithm ()

```
int                 gnutls_x509_crq_get_pk_algorithm    (gnutls_x509_crq_t crq,
                                                         unsigned int *bits);
```

This function will return the public key algorithm of a PKCS10 certificate request.

If bits is non-NULL, it should have enough size to hold the parameters size in bits. For RSA the bits returned is the modulus. For DSA the bits returned are of the public exponent.

crq **:** should contain a gnutls_x509_crq_t structure

bits **:** if bits is non-NULL it will hold the size of the parameters' in bits

Returns **:** a member of the gnutls_pk_algorithm_t enumeration on success, or a negative value on error.

gnutls_x509_crq_get_key_id ()

```
int                 gnutls_x509_crq_get_key_id          (gnutls_x509_crq_t crq,
                                                         unsigned int flags,
                                                         unsigned char *output_data,
                                                         size_t *output_data_size);
```

This function will return a unique ID the depends on the public key parameters. This ID can be used in checking whether a certificate corresponds to the given private key.

If the buffer provided is not long enough to hold the output, then *`output_data_size` is updated and GNUTLS_E_SHORT_MEMORY will be returned. The output will normally be a SHA-1 hash output, which is 20 bytes.

crq **:** a certificate of type gnutls_x509_crq_t

flags **:** should be 0 for now

output_data **:** will contain the key ID

output_data_size **:** holds the size of output_data (and will be replaced by the actual size of parameters)

Returns **:** In case of failure a negative value will be returned, and 0 on success.

Since 2.8.0

gnutls_x509_crq_get_key_rsa_raw ()

```
int                 gnutls_x509_crq_get_key_rsa_raw     (gnutls_x509_crq_t crq,
                                                         gnutls_datum_t *m,
                                                         gnutls_datum_t *e);
```

This function will export the RSA public key's parameters found in the given structure. The new parameters will be allocated using gnutls_malloc() and will be stored in the appropriate datum.

crq **:** Holds the certificate

m **:** will hold the modulus

e **:** will hold the public exponent

Returns **:** On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

Since 2.8.0

gnutls_x509_crq_get_key_usage ()

```
int                 gnutls_x509_crq_get_key_usage       (gnutls_x509_crq_t crq,
                                                         unsigned int *key_usage,
                                                         unsigned int *critical);
```

This function will return certificate's key usage, by reading the keyUsage X.509 extension (2.5.29.15). The key usage value will ORed values of the: GNUTLS_KEY_DIGITAL_SIGNATURE, GNUTLS_KEY_NON_REPUDIATION, GNUTLS_KEY_KEY_ENCI GNUTLS_KEY_DATA_ENCIPHERMENT, GNUTLS_KEY_KEY_AGREEMENT, GNUTLS_KEY_KEY_CERT_SIGN, GNUTLS_ GNUTLS_KEY_ENCIPHER_ONLY, GNUTLS_KEY_DECIPHER_ONLY.

crq : should contain a gnutls_x509_crq_t structure

key_usage : where the key usage bits will be stored

critical : will be non zero if the extension is marked as critical

Returns : the certificate key usage, or a negative value in case of parsing error. If the certificate does not contain the keyUsage extension GNUTLS_E_REQUESTED_DATA_NOT_AVAILABLE will be returned.

Since 2.8.0

gnutls_x509_crq_get_basic_constraints ()

```
int                 gnutls_x509_crq_get_basic_constraints
                                                    (gnutls_x509_crq_t crq,
                                                     unsigned int *critical,
                                                     int *ca,
                                                     int *pathlen);
```

This function will read the certificate's basic constraints, and return the certificates CA status. It reads the basicConstraints X.509 extension (2.5.29.19).

crq : should contain a gnutls_x509_crq_t structure

critical : will be non zero if the extension is marked as critical

ca : pointer to output integer indicating CA status, may be NULL, value is 1 if the certificate CA flag is set, 0 otherwise.

pathlen : pointer to output integer indicating path length (may be NULL), non-negative values indicate a present pathLenConstraint field and the actual value, -1 indicate that the field is absent.

Returns : If the certificate is a CA a positive value will be returned, or zero if the certificate does not have CA flag set. A negative value may be returned in case of errors. If the certificate does not contain the basicConstraints extension GNUTLS_E_REQUESTED_DATA_NOT_AVAILABLE will be returned.

Since 2.8.0

gnutls_x509_crq_get_subject_alt_name ()

```
int                 gnutls_x509_crq_get_subject_alt_name
                                                    (gnutls_x509_crq_t crq,
                                                     unsigned int seq,
                                                     void *ret,
                                                     size_t *ret_size,
                                                     unsigned int *ret_type,
                                                     unsigned int *critical);
```

This function will return the alternative names, contained in the given certificate. It is the same as gnutls_x509_crq_get_subject_alt_name except for the fact that it will return the type of the alternative name in *ret_type* even if the function fails for some reason (i.e. the buffer provided is not enough).

crq : should contain a gnutls_x509_crq_t structure

seq : specifies the sequence number of the alt name, 0 for the first one, 1 for the second etc.

ret : is the place where the alternative name will be copied to

ret_size : holds the size of ret.

ret_type : holds the gnutls_x509_subject_alt_name_t name type

critical : will be non zero if the extension is marked as critical (may be null)

Returns : the alternative subject name type on success, one of the enumerated gnutls_x509_subject_alt_name_t. It will return GNUTLS_E_SHORT_MEMORY_BUFFER if *ret_size* is not large enough to hold the value. In that case *ret_size* will be updated with the required size. If the certificate request does not have an Alternative name with the specified sequence number then GNUTLS_E_REQUESTED_DATA_NOT_AVAILABLE is returned.

Since 2.8.0

gnutls_x509_crq_get_subject_alt_othername_oid ()

```
int                 gnutls_x509_crq_get_subject_alt_othername_oid
                                                    (gnutls_x509_crq_t crq,
                                                     unsigned int seq,
                                                     void *ret,
                                                     size_t *ret_size);
```

This function will extract the type OID of an otherName Subject Alternative Name, contained in the given certificate, and return the type as an enumerated element.

This function is only useful if gnutls_x509_crq_get_subject_alt_name() returned GNUTLS_SAN_OTHERNAME.

crq : should contain a gnutls_x509_crq_t structure

seq : specifies the sequence number of the alt name (0 for the first one, 1 for the second etc.)

ret : is the place where the otherName OID will be copied to

ret_size : holds the size of ret.

Returns : the alternative subject name type on success, one of the enumerated gnutls_x509_subject_alt_name_t. For supported OIDs, it will return one of the virtual (GNUTLS_SAN_OTHERNAME_*) types, e.g. GNUTLS_SAN_OTHERNAME_XMPP, and GNUTLS_SAN_OTHERNAME for unknown OIDs. It will return GNUTLS_E_SHORT_MEMORY_BUFFER if *ret_size* is not large enough to hold the value. In that case *ret_size* will be updated with the required size. If the certificate does not have an Alternative name with the specified sequence number and with the otherName type then GNUTLS_E_REQUESTED_DATA_NOT_AVAILABLE is returned.

Since 2.8.0

gnutls_x509_crq_get_extension_by_oid ()

```
int                 gnutls_x509_crq_get_extension_by_oid
                                                    (gnutls_x509_crq_t crq,
                                                     const char *oid,
                                                     int indx,
                                                     void *buf,
                                                     size_t *sizeof_buf,
                                                     unsigned int *critical);
```

This function will return the extension specified by the OID in the certificate. The extensions will be returned as binary data DER encoded, in the provided buffer.

crq : should contain a gnutls_x509_crq_t structure

oid : holds an Object Identified in null terminated string

indx : In case multiple same OIDs exist in the extensions, this specifies which to send. Use zero to get the first one.

buf : a pointer to a structure to hold the name (may be null)

sizeof_buf : initially holds the size of *buf*

critical : will be non zero if the extension is marked as critical

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative value in case of an error. If the certificate does not contain the specified extension GNUTLS_E_REQUESTED_DATA_NOT_AVAILABLE will be returned.

Since 2.8.0

1.4 pkcs12

pkcs12 —

Synopsis

```
struct                  gnutls_pkcs12_int;
typedef                 gnutls_pkcs12_t;
struct                  gnutls_pkcs12_bag_int;
typedef                 gnutls_pkcs12_bag_t;
int                     gnutls_pkcs12_init                  (gnutls_pkcs12_t *pkcs12);
void                    gnutls_pkcs12_deinit                (gnutls_pkcs12_t pkcs12);
int                     gnutls_pkcs12_import                (gnutls_pkcs12_t pkcs12,
                                                             const gnutls_datum_t *data,
                                                             gnutls_x509_crt_fmt_t format,
                                                             unsigned int flags);
int                     gnutls_pkcs12_export                (gnutls_pkcs12_t pkcs12,
                                                             gnutls_x509_crt_fmt_t format,
                                                             void *output_data,
                                                             size_t *output_data_size);
int                     gnutls_pkcs12_get_bag               (gnutls_pkcs12_t pkcs12,
                                                             int indx,
                                                             gnutls_pkcs12_bag_t bag);
int                     gnutls_pkcs12_set_bag               (gnutls_pkcs12_t pkcs12,
                                                             gnutls_pkcs12_bag_t bag);
int                     gnutls_pkcs12_generate_mac          (gnutls_pkcs12_t pkcs12,
                                                             const char *pass);
int                     gnutls_pkcs12_verify_mac            (gnutls_pkcs12_t pkcs12,
                                                             const char *pass);
int                     gnutls_pkcs12_bag_decrypt           (gnutls_pkcs12_bag_t bag,
                                                             const char *pass);
int                     gnutls_pkcs12_bag_encrypt           (gnutls_pkcs12_bag_t bag,
                                                             const char *pass,
                                                             unsigned int flags);
enum                    gnutls_pkcs12_bag_type_t;
gnutls_pkcs12_bag_type_t  gnutls_pkcs12_bag_get_type        (gnutls_pkcs12_bag_t bag,
                                                             int indx);
int                     gnutls_pkcs12_bag_get_data          (gnutls_pkcs12_bag_t bag,
                                                             int indx,
```

```
                                                     gnutls_datum_t *data);
int                 gnutls_pkcs12_bag_set_data      (gnutls_pkcs12_bag_t bag,
                                                     gnutls_pkcs12_bag_type_t type,
                                                     const gnutls_datum_t *data);
int                 gnutls_pkcs12_bag_set_crl       (gnutls_pkcs12_bag_t bag,
                                                     gnutls_x509_crl_t crl);
int                 gnutls_pkcs12_bag_set_crt       (gnutls_pkcs12_bag_t bag,
                                                     gnutls_x509_crt_t crt);
int                 gnutls_pkcs12_bag_init          (gnutls_pkcs12_bag_t *bag);
void                gnutls_pkcs12_bag_deinit        (gnutls_pkcs12_bag_t bag);
int                 gnutls_pkcs12_bag_get_count     (gnutls_pkcs12_bag_t bag);
int                 gnutls_pkcs12_bag_get_key_id    (gnutls_pkcs12_bag_t bag,
                                                     int indx,
                                                     gnutls_datum_t *id);
int                 gnutls_pkcs12_bag_set_key_id    (gnutls_pkcs12_bag_t bag,
                                                     int indx,
                                                     const gnutls_datum_t *id);
int                 gnutls_pkcs12_bag_get_friendly_name
                                                    (gnutls_pkcs12_bag_t bag,
                                                     int indx,
                                                     char **name);
int                 gnutls_pkcs12_bag_set_friendly_name
                                                    (gnutls_pkcs12_bag_t bag,
                                                     int indx,
                                                     const char *name);
```

Description

Details

struct gnutls_pkcs12_int

```
struct gnutls_pkcs12_int;
```

gnutls_pkcs12_t

```
  typedef struct gnutls_pkcs12_int *gnutls_pkcs12_t;
```

struct gnutls_pkcs12_bag_int

```
struct gnutls_pkcs12_bag_int;
```

gnutls_pkcs12_bag_t

```
  typedef struct gnutls_pkcs12_bag_int *gnutls_pkcs12_bag_t;
```

gnutls_pkcs12_init ()

```
int                 gnutls_pkcs12_init                  (gnutls_pkcs12_t *pkcs12);
```

This function will initialize a PKCS12 structure. PKCS12 structures usually contain lists of X.509 Certificates and X.509 Certificate revocation lists.

pkcs12 : The structure to be initialized

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_pkcs12_deinit ()

```
void                gnutls_pkcs12_deinit            (gnutls_pkcs12_t pkcs12);
```

This function will deinitialize a PKCS12 structure.

pkcs12 : The structure to be initialized

gnutls_pkcs12_import ()

```
int                 gnutls_pkcs12_import            (gnutls_pkcs12_t pkcs12,
                                                     const gnutls_datum_t *data,
                                                     gnutls_x509_crt_fmt_t format,
                                                     unsigned int flags);
```

This function will convert the given DER or PEM encoded PKCS12 to the native gnutls_pkcs12_t format. The output will be stored in 'pkcs12'.

If the PKCS12 is PEM encoded it should have a header of "PKCS12".

pkcs12 : The structure to store the parsed PKCS12.

data : The DER or PEM encoded PKCS12.

format : One of DER or PEM

flags : an ORed sequence of gnutls_privkey_pkcs8_flags

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_pkcs12_export ()

```
int                 gnutls_pkcs12_export            (gnutls_pkcs12_t pkcs12,
                                                     gnutls_x509_crt_fmt_t format,
                                                     void *output_data,
                                                     size_t *output_data_size);
```

This function will export the pkcs12 structure to DER or PEM format.

If the buffer provided is not long enough to hold the output, then *output_data_size will be updated and GNUTLS_E_SHORT_MEMORY will be returned.

If the structure is PEM encoded, it will have a header of "BEGIN PKCS12".

pkcs12 : Holds the pkcs12 structure

format : the format of output params. One of PEM or DER.

output_data : will contain a structure PEM or DER encoded

output_data_size : holds the size of output_data (and will be replaced by the actual size of parameters)

Returns : In case of failure a negative value will be returned, and 0 on success.

gnutls_pkcs12_get_bag ()

```
int                 gnutls_pkcs12_get_bag               (gnutls_pkcs12_t pkcs12,
                                                         int indx,
                                                         gnutls_pkcs12_bag_t bag);
```

This function will return a Bag from the PKCS12 structure.

After the last Bag has been read GNUTLS_E_REQUESTED_DATA_NOT_AVAILABLE will be returned.

pkcs12 : should contain a gnutls_pkcs12_t structure

indx : contains the index of the bag to extract

bag : An initialized bag, where the contents of the bag will be copied

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_pkcs12_set_bag ()

```
int                 gnutls_pkcs12_set_bag               (gnutls_pkcs12_t pkcs12,
                                                         gnutls_pkcs12_bag_t bag);
```

This function will insert a Bag into the PKCS12 structure.

pkcs12 : should contain a gnutls_pkcs12_t structure

bag : An initialized bag

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_pkcs12_generate_mac ()

```
int                 gnutls_pkcs12_generate_mac          (gnutls_pkcs12_t pkcs12,
                                                         const char *pass);
```

This function will generate a MAC for the PKCS12 structure.

pkcs12 : should contain a gnutls_pkcs12_t structure

pass : The password for the MAC

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_pkcs12_verify_mac ()

```
int                 gnutls_pkcs12_verify_mac            (gnutls_pkcs12_t pkcs12,
                                                         const char *pass);
```

This function will verify the MAC for the PKCS12 structure.

pkcs12 : should contain a gnutls_pkcs12_t structure

pass : The password for the MAC

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_pkcs12_bag_decrypt ()

```
int                 gnutls_pkcs12_bag_decrypt           (gnutls_pkcs12_bag_t bag,
                                                         const char *pass);
```

This function will decrypt the given encrypted bag and return 0 on success.

bag : The bag

pass : The password used for encryption, must be ASCII.

Returns : On success, GNUTLS_E_SUCCESS (zero) is returned, otherwise an error code is returned.

gnutls_pkcs12_bag_encrypt ()

```
int                 gnutls_pkcs12_bag_encrypt           (gnutls_pkcs12_bag_t bag,
                                                         const char *pass,
                                                         unsigned int flags);
```

This function will encrypt the given bag.

bag : The bag

pass : The password used for encryption, must be ASCII

flags : should be one of gnutls_pkcs_encrypt_flags_t elements bitwise or'd

Returns : On success, GNUTLS_E_SUCCESS (zero) is returned, otherwise an error code is returned.

enum gnutls_pkcs12_bag_type_t

```
typedef enum gnutls_pkcs12_bag_type_t
{
  GNUTLS_BAG_EMPTY = 0,
  GNUTLS_BAG_PKCS8_ENCRYPTED_KEY = 1,
  GNUTLS_BAG_PKCS8_KEY = 2,
  GNUTLS_BAG_CERTIFICATE = 3,
  GNUTLS_BAG_CRL = 4,
  GNUTLS_BAG_SECRET = 5,~/* Secret data. Underspecified in pkcs-12,
       * gnutls extension. We use the PKCS-9
       * random nonce ID 1.2.840.113549.1.9.25.3
       * to store randomly generated keys.
       */
  GNUTLS_BAG_ENCRYPTED = 10,
  GNUTLS_BAG_UNKNOWN = 20
} gnutls_pkcs12_bag_type_t;
```

Enumeration of different PKCS 12 bag types.

GNUTLS_BAG_EMPTY Empty PKCS-12 bag.

GNUTLS_BAG_PKCS8_ENCRYPTED_KEY PKCS-12 bag with PKCS-8 encrypted key.

GNUTLS_BAG_PKCS8_KEY PKCS-12 bag with PKCS-8 key.

GNUTLS_BAG_CERTIFICATE PKCS-12 bag with certificate.

GNUTLS_BAG_CRL PKCS-12 bag with CRL.

GNUTLS_BAG_SECRET PKCS-12 bag with secret PKCS-9 keys.

GNUTLS_BAG_ENCRYPTED Encrypted PKCS-12 bag.

GNUTLS_BAG_UNKNOWN Unknown PKCS-12 bag.

gnutls_pkcs12_bag_get_type ()

```
gnutls_pkcs12_bag_type_t  gnutls_pkcs12_bag_get_type    (gnutls_pkcs12_bag_t bag,
                                                         int indx);
```

This function will return the bag's type.

bag : The bag

indx : The element of the bag to get the type

Returns : One of the gnutls_pkcs12_bag_type_t enumerations.

gnutls_pkcs12_bag_get_data ()

```
int                 gnutls_pkcs12_bag_get_data          (gnutls_pkcs12_bag_t bag,
                                                         int indx,
                                                         gnutls_datum_t *data);
```

This function will return the bag's data. The data is a constant that is stored into the bag. Should not be accessed after the bag is deleted.

bag : The bag

indx : The element of the bag to get the data from

data : where the bag's data will be. Should be treated as constant.

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_pkcs12_bag_set_data ()

```
int                 gnutls_pkcs12_bag_set_data          (gnutls_pkcs12_bag_t bag,
                                                         gnutls_pkcs12_bag_type_t type,
                                                         const gnutls_datum_t *data);
```

This function will insert the given data of the given type into the bag.

bag : The bag

type : The data's type

data : the data to be copied.

Returns : the index of the added bag on success, or a negative value on error.

gnutls_pkcs12_bag_set_crl ()

```
int                 gnutls_pkcs12_bag_set_crl           (gnutls_pkcs12_bag_t bag,
                                                         gnutls_x509_crl_t crl);
```

This function will insert the given CRL into the bag. This is just a wrapper over gnutls_pkcs12_bag_set_data().

bag : The bag

crl : the CRL to be copied.

Returns : the index of the added bag on success, or a negative value on failure.

gnutls_pkcs12_bag_set_crt ()

```
int                 gnutls_pkcs12_bag_set_crt           (gnutls_pkcs12_bag_t bag,
                                                         gnutls_x509_crt_t crt);
```

This function will insert the given certificate into the bag. This is just a wrapper over gnutls_pkcs12_bag_set_data().

bag : The bag

crt : the certificate to be copied.

Returns : the index of the added bag on success, or a negative value on failure.

gnutls_pkcs12_bag_init ()

```
int                 gnutls_pkcs12_bag_init              (gnutls_pkcs12_bag_t *bag);
```

This function will initialize a PKCS12 bag structure. PKCS12 Bags usually contain private keys, lists of X.509 Certificates and X.509 Certificate revocation lists.

bag : The structure to be initialized

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_pkcs12_bag_deinit ()

```
void                gnutls_pkcs12_bag_deinit            (gnutls_pkcs12_bag_t bag);
```

This function will deinitialize a PKCS12 Bag structure.

bag : The structure to be initialized

gnutls_pkcs12_bag_get_count ()

```
int                 gnutls_pkcs12_bag_get_count         (gnutls_pkcs12_bag_t bag);
```

This function will return the number of the elements withing the bag.

bag : The bag

Returns : Number of elements in bag, or an negative error code on error.

gnutls_pkcs12_bag_get_key_id ()

```
int                 gnutls_pkcs12_bag_get_key_id        (gnutls_pkcs12_bag_t bag,
                                                         int indx,
                                                         gnutls_datum_t *id);
```

This function will return the key ID, of the specified bag element. The key ID is usually used to distinguish the local private key and the certificate pair.

bag : The bag

indx : The bag's element to add the id

id : where the ID will be copied (to be treated as const)

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value. or a negative value on error.

gnutls_pkcs12_bag_set_key_id ()

```
int                 gnutls_pkcs12_bag_set_key_id        (gnutls_pkcs12_bag_t bag,
                                                         int indx,
                                                         const gnutls_datum_t *id);
```

This function will add the given key ID, to the specified, by the index, bag element. The key ID will be encoded as a 'Local key identifier' bag attribute, which is usually used to distinguish the local private key and the certificate pair.

bag : The bag

indx : The bag's element to add the id

id : the ID

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value. or a negative value on error.

gnutls_pkcs12_bag_get_friendly_name ()

```
int                 gnutls_pkcs12_bag_get_friendly_name (gnutls_pkcs12_bag_t bag,
                                                         int indx,
                                                         char **name);
```

This function will return the friendly name, of the specified bag element. The key ID is usually used to distinguish the local private key and the certificate pair.

bag : The bag

indx : The bag's element to add the id

name : will hold a pointer to the name (to be treated as const)

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value. or a negative value on error.

gnutls_pkcs12_bag_set_friendly_name ()

```
int                 gnutls_pkcs12_bag_set_friendly_name (gnutls_pkcs12_bag_t bag,
                                                         int indx,
                                                         const char *name);
```

This function will add the given key friendly name, to the specified, by the index, bag element. The name will be encoded as a 'Friendly name' bag attribute, which is usually used to set a user name to the local private key and the certificate pair.

bag : The bag

indx : The bag's element to add the id

name : the name

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value. or a negative value on error.

1.5 openpgp

openpgp —

Synopsis

```
enum                gnutls_openpgp_crt_fmt_t;
typedef             gnutls_openpgp_keyid_t;
int                 gnutls_openpgp_crt_init             (gnutls_openpgp_crt_t *key);
void                gnutls_openpgp_crt_deinit           (gnutls_openpgp_crt_t key);
int                 gnutls_openpgp_crt_import           (gnutls_openpgp_crt_t key,
                                                         const gnutls_datum_t *data,
                                                         gnutls_openpgp_crt_fmt_t format);
int                 gnutls_openpgp_crt_export           (gnutls_openpgp_crt_t key,
                                                         gnutls_openpgp_crt_fmt_t format,
                                                         void *output_data,
                                                         size_t *output_data_size);
int                 gnutls_openpgp_crt_print            (gnutls_openpgp_crt_t cert,
                                                         gnutls_certificate_print_formats_
                                                         gnutls_datum_t *out);
int                 gnutls_openpgp_crt_get_key_usage    (gnutls_openpgp_crt_t key,
                                                         unsigned int *key_usage);
int                 gnutls_openpgp_crt_get_fingerprint  (gnutls_openpgp_crt_t key,
                                                         void *fpr,
                                                         size_t *fprlen);
int                 gnutls_openpgp_crt_get_subkey_fingerprint
                                                        (gnutls_openpgp_crt_t key,
                                                         unsigned int idx,
                                                         void *fpr,
                                                         size_t *fprlen);
int                 gnutls_openpgp_crt_get_name         (gnutls_openpgp_crt_t key,
                                                         int idx,
                                                         char *buf,
                                                         size_t *sizeof_buf);
gnutls_pk_algorithm_t  gnutls_openpgp_crt_get_pk_algorithm
                                                        (gnutls_openpgp_crt_t key,
                                                         unsigned int *bits);
int                 gnutls_openpgp_crt_get_version      (gnutls_openpgp_crt_t key);
time_t              gnutls_openpgp_crt_get_creation_time
                                                        (gnutls_openpgp_crt_t key);
time_t              gnutls_openpgp_crt_get_expiration_time
                                                        (gnutls_openpgp_crt_t key);
int                 gnutls_openpgp_crt_get_key_id       (gnutls_openpgp_crt_t key,
                                                         gnutls_openpgp_keyid_t keyid);
int                 gnutls_openpgp_crt_check_hostname   (gnutls_openpgp_crt_t key,
                                                         const char *hostname);
int                 gnutls_openpgp_crt_get_revoked_status
                                                        (gnutls_openpgp_crt_t key);
int                 gnutls_openpgp_crt_get_subkey_count (gnutls_openpgp_crt_t key);
int                 gnutls_openpgp_crt_get_subkey_idx   (gnutls_openpgp_crt_t key,
                                                         const gnutls_openpgp_keyid_t keyi
int                 gnutls_openpgp_crt_get_subkey_revoked_status
                                                        (gnutls_openpgp_crt_t key,
                                                         unsigned int idx);
gnutls_pk_algorithm_t  gnutls_openpgp_crt_get_subkey_pk_algorithm
                                                        (gnutls_openpgp_crt_t key,
                                                         unsigned int idx,
                                                         unsigned int *bits);
time_t              gnutls_openpgp_crt_get_subkey_creation_time
                                                        (gnutls_openpgp_crt_t key,
                                                         unsigned int idx);
```

```
time_t               gnutls_openpgp_crt_get_subkey_expiration_time
                                                    (gnutls_openpgp_crt_t key,
                                                     unsigned int idx);
int                  gnutls_openpgp_crt_get_subkey_id    (gnutls_openpgp_crt_t key,
                                                     unsigned int idx,
                                                     gnutls_openpgp_keyid_t keyid);
int                  gnutls_openpgp_crt_get_subkey_usage (gnutls_openpgp_crt_t key,
                                                     unsigned int idx,
                                                     unsigned int *key_usage);
int                  gnutls_openpgp_crt_get_subkey_pk_dsa_raw
                                                    (gnutls_openpgp_crt_t crt,
                                                     unsigned int idx,
                                                     gnutls_datum_t *p,
                                                     gnutls_datum_t *q,
                                                     gnutls_datum_t *g,
                                                     gnutls_datum_t *y);
int                  gnutls_openpgp_crt_get_subkey_pk_rsa_raw
                                                    (gnutls_openpgp_crt_t crt,
                                                     unsigned int idx,
                                                     gnutls_datum_t *m,
                                                     gnutls_datum_t *e);
int                  gnutls_openpgp_crt_get_pk_dsa_raw   (gnutls_openpgp_crt_t crt,
                                                     gnutls_datum_t *p,
                                                     gnutls_datum_t *q,
                                                     gnutls_datum_t *g,
                                                     gnutls_datum_t *y);
int                  gnutls_openpgp_crt_get_pk_rsa_raw   (gnutls_openpgp_crt_t crt,
                                                     gnutls_datum_t *m,
                                                     gnutls_datum_t *e);
int                  gnutls_openpgp_crt_get_preferred_key_id
                                                    (gnutls_openpgp_crt_t key,
                                                     gnutls_openpgp_keyid_t keyid);
int                  gnutls_openpgp_crt_set_preferred_key_id
                                                    (gnutls_openpgp_crt_t key,
                                                     const gnutls_openpgp_keyid_t keyi
int                  gnutls_openpgp_privkey_init         (gnutls_openpgp_privkey_t *key);
void                 gnutls_openpgp_privkey_deinit       (gnutls_openpgp_privkey_t key);
gnutls_pk_algorithm_t  gnutls_openpgp_privkey_get_pk_algorithm
                                                    (gnutls_openpgp_privkey_t key,
                                                     unsigned int *bits);
int                  gnutls_openpgp_privkey_import       (gnutls_openpgp_privkey_t key,
                                                     const gnutls_datum_t *data,
                                                     gnutls_openpgp_crt_fmt_t format,
                                                     const char *password,
                                                     unsigned int flags);
int                  gnutls_openpgp_privkey_sign_hash    (gnutls_openpgp_privkey_t key,
                                                     const gnutls_datum_t *hash,
                                                     gnutls_datum_t *signature);
int                  gnutls_openpgp_privkey_get_fingerprint
                                                    (gnutls_openpgp_privkey_t key,
                                                     void *fpr,
                                                     size_t *fprlen);
int                  gnutls_openpgp_privkey_get_subkey_fingerprint
                                                    (gnutls_openpgp_privkey_t key,
                                                     unsigned int idx,
                                                     void *fpr,
                                                     size_t *fprlen);
```

```
int                 gnutls_openpgp_privkey_get_key_id   (gnutls_openpgp_privkey_t key,
                                                         gnutls_openpgp_keyid_t keyid);
int                 gnutls_openpgp_privkey_get_subkey_count
                                                        (gnutls_openpgp_privkey_t key);
int                 gnutls_openpgp_privkey_get_subkey_idx
                                                        (gnutls_openpgp_privkey_t key,
                                                         const gnutls_openpgp_keyid_t keyi
int                 gnutls_openpgp_privkey_get_subkey_revoked_status
                                                        (gnutls_openpgp_privkey_t key,
                                                         unsigned int idx);
int                 gnutls_openpgp_privkey_get_revoked_status
                                                        (gnutls_openpgp_privkey_t key);
gnutls_pk_algorithm_t  gnutls_openpgp_privkey_get_subkey_pk_algorithm
                                                        (gnutls_openpgp_privkey_t key,
                                                         unsigned int idx,
                                                         unsigned int *bits);
time_t              gnutls_openpgp_privkey_get_subkey_expiration_time
                                                        (gnutls_openpgp_privkey_t key,
                                                         unsigned int idx);
int                 gnutls_openpgp_privkey_get_subkey_id
                                                        (gnutls_openpgp_privkey_t key,
                                                         unsigned int idx,
                                                         gnutls_openpgp_keyid_t keyid);
time_t              gnutls_openpgp_privkey_get_subkey_creation_time
                                                        (gnutls_openpgp_privkey_t key,
                                                         unsigned int idx);
int                 gnutls_openpgp_privkey_export_subkey_dsa_raw
                                                        (gnutls_openpgp_privkey_t pkey,
                                                         unsigned int idx,
                                                         gnutls_datum_t *p,
                                                         gnutls_datum_t *q,
                                                         gnutls_datum_t *g,
                                                         gnutls_datum_t *y,
                                                         gnutls_datum_t *x);
int                 gnutls_openpgp_privkey_export_subkey_rsa_raw
                                                        (gnutls_openpgp_privkey_t pkey,
                                                         unsigned int idx,
                                                         gnutls_datum_t *m,
                                                         gnutls_datum_t *e,
                                                         gnutls_datum_t *d,
                                                         gnutls_datum_t *p,
                                                         gnutls_datum_t *q,
                                                         gnutls_datum_t *u);
int                 gnutls_openpgp_privkey_export_dsa_raw
                                                        (gnutls_openpgp_privkey_t pkey,
                                                         gnutls_datum_t *p,
                                                         gnutls_datum_t *q,
                                                         gnutls_datum_t *g,
                                                         gnutls_datum_t *y,
                                                         gnutls_datum_t *x);
int                 gnutls_openpgp_privkey_export_rsa_raw
                                                        (gnutls_openpgp_privkey_t pkey,
                                                         gnutls_datum_t *m,
                                                         gnutls_datum_t *e,
                                                         gnutls_datum_t *d,
                                                         gnutls_datum_t *p,
                                                         gnutls_datum_t *q,
```

```
                                                gnutls_datum_t *u);
int                 gnutls_openpgp_privkey_export       (gnutls_openpgp_privkey_t key,
                                                gnutls_openpgp_crt_fmt_t format,
                                                const char *password,
                                                unsigned int flags,
                                                void *output_data,
                                                size_t *output_data_size);
int                 gnutls_openpgp_privkey_set_preferred_key_id
                                               (gnutls_openpgp_privkey_t key,
                                                const gnutls_openpgp_keyid_t keyi
int                 gnutls_openpgp_privkey_get_preferred_key_id
                                               (gnutls_openpgp_privkey_t key,
                                                gnutls_openpgp_keyid_t keyid);
int                 gnutls_openpgp_crt_get_auth_subkey  (gnutls_openpgp_crt_t crt,
                                                gnutls_openpgp_keyid_t keyid,
                                                unsigned int flag);
int                 gnutls_openpgp_keyring_init         (gnutls_openpgp_keyring_t *keyring
void                gnutls_openpgp_keyring_deinit       (gnutls_openpgp_keyring_t keyring)
int                 gnutls_openpgp_keyring_import       (gnutls_openpgp_keyring_t keyring,
                                                const gnutls_datum_t *data,
                                                gnutls_openpgp_crt_fmt_t format);
int                 gnutls_openpgp_keyring_check_id     (gnutls_openpgp_keyring_t ring,
                                                const gnutls_openpgp_keyid_t keyi
                                                unsigned int flags);
int                 gnutls_openpgp_crt_verify_ring      (gnutls_openpgp_crt_t key,
                                                gnutls_openpgp_keyring_t keyring,
                                                unsigned int flags,
                                                unsigned int *verify);
int                 gnutls_openpgp_crt_verify_self      (gnutls_openpgp_crt_t key,
                                                unsigned int flags,
                                                unsigned int *verify);
int                 gnutls_openpgp_keyring_get_crt      (gnutls_openpgp_keyring_t ring,
                                                unsigned int idx,
                                                gnutls_openpgp_crt_t *cert);
int                 gnutls_openpgp_keyring_get_crt_count
                                               (gnutls_openpgp_keyring_t ring);
int                 (*gnutls_openpgp_recv_key_func)     (gnutls_session_t session,
                                                unsigned char *keyfpr,
                                                unsigned int keyfpr_length,
                                                gnutls_datum_t *key);
void                gnutls_openpgp_set_recv_key_function
                                               (gnutls_session_t session,
                                                gnutls_openpgp_recv_key_func func
int                 gnutls_certificate_set_openpgp_key  (gnutls_certificate_credentials_t
                                                gnutls_openpgp_crt_t key,
                                                gnutls_openpgp_privkey_t pkey);
int                 gnutls_certificate_set_openpgp_key_file
                                               (gnutls_certificate_credentials_t
                                                const char *certfile,
                                                const char *keyfile,
                                                gnutls_openpgp_crt_fmt_t format);
int                 gnutls_certificate_set_openpgp_key_mem
                                               (gnutls_certificate_credentials_t
                                                const gnutls_datum_t *cert,
                                                const gnutls_datum_t *key,
                                                gnutls_openpgp_crt_fmt_t format);
int                 gnutls_certificate_set_openpgp_key_file2
```

```
                                                    (gnutls_certificate_credentials_t
                                                     const char *certfile,
                                                     const char *keyfile,
                                                     const char *subkey_id,
                                                     gnutls_openpgp_crt_fmt_t format);
int                 gnutls_certificate_set_openpgp_key_mem2
                                                    (gnutls_certificate_credentials_t
                                                     const gnutls_datum_t *cert,
                                                     const gnutls_datum_t *key,
                                                     const char *subkey_id,
                                                     gnutls_openpgp_crt_fmt_t format);
int                 gnutls_certificate_set_openpgp_keyring_mem
                                                    (gnutls_certificate_credentials_t
                                                     unsigned char *data,
                                                     size_t dlen,
                                                     gnutls_openpgp_crt_fmt_t format);
int                 gnutls_certificate_set_openpgp_keyring_file
                                                    (gnutls_certificate_credentials_t
                                                     const char *file,
                                                     gnutls_openpgp_crt_fmt_t format);
```

Description

Details

enum gnutls_openpgp_crt_fmt_t

```
typedef enum gnutls_openpgp_crt_fmt
{
  GNUTLS_OPENPGP_FMT_RAW,
  GNUTLS_OPENPGP_FMT_BASE64
} gnutls_openpgp_crt_fmt_t;
```

Enumeration of different OpenPGP key formats.

GNUTLS_OPENPGP_FMT_RAW OpenPGP certificate in raw format.

GNUTLS_OPENPGP_FMT_BASE64 OpenPGP certificate in base64 format.

gnutls_openpgp_keyid_t

```
typedef unsigned char gnutls_openpgp_keyid_t[8];
```

gnutls_openpgp_crt_init ()

```
int                 gnutls_openpgp_crt_init             (gnutls_openpgp_crt_t *key);
```

This function will initialize an OpenPGP key structure.

key : The structure to be initialized

Returns : GNUTLS_E_SUCCESS on success, or an error code.

gnutls_openpgp_crt_deinit ()

```
void                gnutls_openpgp_crt_deinit           (gnutls_openpgp_crt_t key);
```

This function will deinitialize a key structure.

key : The structure to be initialized

gnutls_openpgp_crt_import ()

```
int                 gnutls_openpgp_crt_import           (gnutls_openpgp_crt_t key,
                                                         const gnutls_datum_t *data,
                                                         gnutls_openpgp_crt_fmt_t format);
```

This function will convert the given RAW or Base64 encoded key to the native gnutls_openpgp_crt_t format. The output will be stored in 'key'.

key : The structure to store the parsed key.

data : The RAW or BASE64 encoded key.

format : One of gnutls_openpgp_crt_fmt_t elements.

Returns : GNUTLS_E_SUCCESS on success, or an error code.

gnutls_openpgp_crt_export ()

```
int                 gnutls_openpgp_crt_export           (gnutls_openpgp_crt_t key,
                                                         gnutls_openpgp_crt_fmt_t format,
                                                         void *output_data,
                                                         size_t *output_data_size);
```

This function will convert the given key to RAW or Base64 format. If the buffer provided is not long enough to hold the output, then GNUTLS_E_SHORT_MEMORY_BUFFER will be returned.

key : Holds the key.

format : One of gnutls_openpgp_crt_fmt_t elements.

output_data : will contain the key base64 encoded or raw

output_data_size : holds the size of output_data (and will be replaced by the actual size of parameters)

Returns : GNUTLS_E_SUCCESS on success, or an error code.

gnutls_openpgp_crt_print ()

```
int                 gnutls_openpgp_crt_print            (gnutls_openpgp_crt_t cert,
                                                         gnutls_certificate_print_formats_t ↩
                                                              format,
                                                         gnutls_datum_t *out);
```

This function will pretty print an OpenPGP certificate, suitable for display to a human.

The format should be zero for future compatibility.

The output *out* needs to be deallocate using gnutls_free().

cert : The structure to be printed

format : Indicate the format to use

out : Newly allocated datum with zero terminated string.

Returns : GNUTLS_E_SUCCESS on success, or an error code.

gnutls_openpgp_crt_get_key_usage ()

```
int                 gnutls_openpgp_crt_get_key_usage    (gnutls_openpgp_crt_t key,
                                                         unsigned int *key_usage);
```

This function will return certificate's key usage, by checking the key algorithm. The key usage value will ORed values of the: GNUTLS_KEY_DIGITAL_SIGNATURE, GNUTLS_KEY_KEY_ENCIPHERMENT.

key : should contain a gnutls_openpgp_crt_t structure

key_usage : where the key usage bits will be stored

Returns : GNUTLS_E_SUCCESS on success, or an error code.

gnutls_openpgp_crt_get_fingerprint ()

```
int                 gnutls_openpgp_crt_get_fingerprint  (gnutls_openpgp_crt_t key,
                                                         void *fpr,
                                                         size_t *fprlen);
```

Get key fingerprint. Depending on the algorithm, the fingerprint can be 16 or 20 bytes.

key : the raw data that contains the OpenPGP public key.

fpr : the buffer to save the fingerprint, must hold at least 20 bytes.

fprlen : the integer to save the length of the fingerprint.

Returns : On success, 0 is returned. Otherwise, an error code.

gnutls_openpgp_crt_get_subkey_fingerprint ()

```
int                 gnutls_openpgp_crt_get_subkey_fingerprint
                                                        (gnutls_openpgp_crt_t key,
                                                         unsigned int idx,
                                                         void *fpr,
                                                         size_t *fprlen);
```

Get key fingerprint of a subkey. Depending on the algorithm, the fingerprint can be 16 or 20 bytes.

key : the raw data that contains the OpenPGP public key.

idx : the subkey index

fpr : the buffer to save the fingerprint, must hold at least 20 bytes.

fprlen : the integer to save the length of the fingerprint.

Returns : On success, 0 is returned. Otherwise, an error code.

Since 2.4.0

gnutls_openpgp_crt_get_name ()

```
int                 gnutls_openpgp_crt_get_name         (gnutls_openpgp_crt_t key,
                                                         int idx,
                                                         char *buf,
                                                         size_t *sizeof_buf);
```

Extracts the userID from the parsed OpenPGP key.

key : the structure that contains the OpenPGP public key.

idx : the index of the ID to extract

buf : a pointer to a structure to hold the name, may be NULL to only get the *sizeof_buf*.

sizeof_buf : holds the maximum size of *buf*, on return hold the actual/required size of *buf*.

Returns : GNUTLS_E_SUCCESS on success, and if the index of the ID does not exist GNUTLS_E_REQUESTED_DATA_NOT_AVA
or an error code.

gnutls_openpgp_crt_get_pk_algorithm ()

```
gnutls_pk_algorithm_t  gnutls_openpgp_crt_get_pk_algorithm
                                                        (gnutls_openpgp_crt_t key,
                                                         unsigned int *bits);
```

This function will return the public key algorithm of an OpenPGP certificate.

If bits is non null, it should have enough size to hold the parameters size in bits. For RSA the bits returned is the modulus. For DSA the bits returned are of the public exponent.

key : is an OpenPGP key

bits : if bits is non null it will hold the size of the parameters' in bits

Returns : a member of the gnutls_pk_algorithm_t enumeration on success, or a negative value on error.

gnutls_openpgp_crt_get_version ()

```
int                 gnutls_openpgp_crt_get_version      (gnutls_openpgp_crt_t key);
```

Extract the version of the OpenPGP key.

key : the structure that contains the OpenPGP public key.

Returns : the version number is returned, or a negative value on errors.

gnutls_openpgp_crt_get_creation_time ()

```
time_t              gnutls_openpgp_crt_get_creation_time
                                                        (gnutls_openpgp_crt_t key);
```

Get key creation time.

key : the structure that contains the OpenPGP public key.

Returns : the timestamp when the OpenPGP key was created.

gnutls_openpgp_crt_get_expiration_time ()

```
time_t              gnutls_openpgp_crt_get_expiration_time
                                                    (gnutls_openpgp_crt_t key);
```

Get key expiration time. A value of '0' means that the key doesn't expire at all.

key : the structure that contains the OpenPGP public key.

Returns : the time when the OpenPGP key expires.

gnutls_openpgp_crt_get_key_id ()

```
int                 gnutls_openpgp_crt_get_key_id       (gnutls_openpgp_crt_t key,
                                                         gnutls_openpgp_keyid_t keyid);
```

Get key id string.

key : the structure that contains the OpenPGP public key.

keyid : the buffer to save the keyid.

Returns : the 64-bit keyID of the OpenPGP key.

Since 2.4.0

gnutls_openpgp_crt_check_hostname ()

```
int                 gnutls_openpgp_crt_check_hostname   (gnutls_openpgp_crt_t key,
                                                         const char *hostname);
```

This function will check if the given key's owner matches the given hostname. This is a basic implementation of the matching described in RFC2818 (HTTPS), which takes into account wildcards.

key : should contain a gnutls_openpgp_crt_t structure

hostname : A null terminated string that contains a DNS name

Returns : GNUTLS_E_SUCCESS on success, or an error code.

gnutls_openpgp_crt_get_revoked_status ()

```
int                 gnutls_openpgp_crt_get_revoked_status
                                                    (gnutls_openpgp_crt_t key);
```

Get revocation status of key.

key : the structure that contains the OpenPGP public key.

Returns : true (1) if the key has been revoked, or false (0) if it has not.

Since 2.4.0

gnutls_openpgp_crt_get_subkey_count ()

```
int                 gnutls_openpgp_crt_get_subkey_count (gnutls_openpgp_crt_t key);
```

This function will return the number of subkeys present in the given OpenPGP certificate.

key **:** is an OpenPGP key

Returns **:** the number of subkeys, or a negative value on error.

Since 2.4.0

gnutls_openpgp_crt_get_subkey_idx ()

```
int                 gnutls_openpgp_crt_get_subkey_idx   (gnutls_openpgp_crt_t key,
                                                         const gnutls_openpgp_keyid_t keyid ↩
                                                             );
```

Get subkey's index.

key **:** the structure that contains the OpenPGP public key.

keyid **:** the keyid.

Returns **:** the index of the subkey or a negative error value.

Since 2.4.0

gnutls_openpgp_crt_get_subkey_revoked_status ()

```
int                 gnutls_openpgp_crt_get_subkey_revoked_status
                                                        (gnutls_openpgp_crt_t key,
                                                         unsigned int idx);
```

Get subkey revocation status. A negative value indicates an error.

key **:** the structure that contains the OpenPGP public key.

idx **:** is the subkey index

Returns **:** true (1) if the key has been revoked, or false (0) if it has not.

Since 2.4.0

gnutls_openpgp_crt_get_subkey_pk_algorithm ()

```
gnutls_pk_algorithm_t  gnutls_openpgp_crt_get_subkey_pk_algorithm
                                                        (gnutls_openpgp_crt_t key,
                                                         unsigned int idx,
                                                         unsigned int *bits);
```

This function will return the public key algorithm of a subkey of an OpenPGP certificate.

If bits is non null, it should have enough size to hold the parameters size in bits. For RSA the bits returned is the modulus. For DSA the bits returned are of the public exponent.

key : is an OpenPGP key

idx : is the subkey index

bits : if bits is non null it will hold the size of the parameters' in bits

Returns : a member of the gnutls_pk_algorithm_t enumeration on success, or a negative value on error.

Since 2.4.0

gnutls_openpgp_crt_get_subkey_creation_time ()

```
time_t              gnutls_openpgp_crt_get_subkey_creation_time
                                                    (gnutls_openpgp_crt_t key,
                                                     unsigned int idx);
```

Get subkey creation time.

key : the structure that contains the OpenPGP public key.

idx : the subkey index

Returns : the timestamp when the OpenPGP sub-key was created.

Since 2.4.0

gnutls_openpgp_crt_get_subkey_expiration_time ()

```
time_t              gnutls_openpgp_crt_get_subkey_expiration_time
                                                    (gnutls_openpgp_crt_t key,
                                                     unsigned int idx);
```

Get subkey expiration time. A value of '0' means that the key doesn't expire at all.

key : the structure that contains the OpenPGP public key.

idx : the subkey index

Returns : the time when the OpenPGP key expires.

Since 2.4.0

gnutls_openpgp_crt_get_subkey_id ()

```
int                 gnutls_openpgp_crt_get_subkey_id    (gnutls_openpgp_crt_t key,
                                                         unsigned int idx,
                                                         gnutls_openpgp_keyid_t keyid);
```

Get the subkey's key-id.

key : the structure that contains the OpenPGP public key.

idx : the subkey index

keyid : the buffer to save the keyid.

Returns : the 64-bit keyID of the OpenPGP key.

gnutls_openpgp_crt_get_subkey_usage ()

```
int                 gnutls_openpgp_crt_get_subkey_usage (gnutls_openpgp_crt_t key,
                                                         unsigned int idx,
                                                         unsigned int *key_usage);
```

This function will return certificate's key usage, by checking the key algorithm. The key usage value will ORed values of GNUTLS_KEY_DIGITAL_SIGNATURE or GNUTLS_KEY_KEY_ENCIPHERMENT.

A negative value may be returned in case of parsing error.

key : should contain a gnutls_openpgp_crt_t structure

idx : the subkey index

key_usage : where the key usage bits will be stored

Returns : key usage value.

Since 2.4.0

gnutls_openpgp_crt_get_subkey_pk_dsa_raw ()

```
int                 gnutls_openpgp_crt_get_subkey_pk_dsa_raw
                                                        (gnutls_openpgp_crt_t crt,
                                                         unsigned int idx,
                                                         gnutls_datum_t *p,
                                                         gnutls_datum_t *q,
                                                         gnutls_datum_t *g,
                                                         gnutls_datum_t *y);
```

This function will export the DSA public key's parameters found in the given certificate. The new parameters will be allocated using gnutls_malloc() and will be stored in the appropriate datum.

crt : Holds the certificate

idx : Is the subkey index

p : will hold the p

q : will hold the q

g : will hold the g

y : will hold the y

Returns : GNUTLS_E_SUCCESS on success, otherwise an error.

Since 2.4.0

gnutls_openpgp_crt_get_subkey_pk_rsa_raw ()

```
int                 gnutls_openpgp_crt_get_subkey_pk_rsa_raw
                                                        (gnutls_openpgp_crt_t crt,
                                                         unsigned int idx,
                                                         gnutls_datum_t *m,
                                                         gnutls_datum_t *e);
```

This function will export the RSA public key's parameters found in the given structure. The new parameters will be allocated using gnutls_malloc() and will be stored in the appropriate datum.

crt : Holds the certificate

idx : Is the subkey index

m : will hold the modulus

e : will hold the public exponent

Returns : GNUTLS_E_SUCCESS on success, otherwise an error.

Since 2.4.0

gnutls_openpgp_crt_get_pk_dsa_raw ()

```
int                 gnutls_openpgp_crt_get_pk_dsa_raw   (gnutls_openpgp_crt_t crt,
                                                         gnutls_datum_t *p,
                                                         gnutls_datum_t *q,
                                                         gnutls_datum_t *g,
                                                         gnutls_datum_t *y);
```

This function will export the DSA public key's parameters found in the given certificate. The new parameters will be allocated using gnutls_malloc() and will be stored in the appropriate datum.

crt : Holds the certificate

p : will hold the p

q : will hold the q

g : will hold the g

y : will hold the y

Returns : GNUTLS_E_SUCCESS on success, otherwise an error.

Since 2.4.0

gnutls_openpgp_crt_get_pk_rsa_raw ()

```
int                 gnutls_openpgp_crt_get_pk_rsa_raw   (gnutls_openpgp_crt_t crt,
                                                         gnutls_datum_t *m,
                                                         gnutls_datum_t *e);
```

This function will export the RSA public key's parameters found in the given structure. The new parameters will be allocated using gnutls_malloc() and will be stored in the appropriate datum.

crt : Holds the certificate

m : will hold the modulus

e : will hold the public exponent

Returns : GNUTLS_E_SUCCESS on success, otherwise an error.

Since 2.4.0

gnutls_openpgp_crt_get_preferred_key_id ()

```
int                 gnutls_openpgp_crt_get_preferred_key_id
                                                    (gnutls_openpgp_crt_t key,
                                                     gnutls_openpgp_keyid_t keyid);
```

Get preferred key id. If it hasn't been set it returns GNUTLS_E_INVALID_REQUEST.

key : the structure that contains the OpenPGP public key.

keyid : the struct to save the keyid.

Returns : the 64-bit preferred keyID of the OpenPGP key.

gnutls_openpgp_crt_set_preferred_key_id ()

```
int                 gnutls_openpgp_crt_set_preferred_key_id
                                                    (gnutls_openpgp_crt_t key,
                                                     const gnutls_openpgp_keyid_t keyid ↩
                                                         );
```

This allows setting a preferred key id for the given certificate. This key will be used by functions that involve key handling.

key : the structure that contains the OpenPGP public key.

keyid : the selected keyid

Returns : On success, GNUTLS_E_SUCCESS (zero) is returned, otherwise an error code is returned.

gnutls_openpgp_privkey_init ()

```
int                 gnutls_openpgp_privkey_init         (gnutls_openpgp_privkey_t *key);
```

This function will initialize an OpenPGP key structure.

key : The structure to be initialized

Returns : GNUTLS_E_SUCCESS on success, or an error code.

gnutls_openpgp_privkey_deinit ()

```
void                gnutls_openpgp_privkey_deinit       (gnutls_openpgp_privkey_t key);
```

This function will deinitialize a key structure.

key : The structure to be initialized

gnutls_openpgp_privkey_get_pk_algorithm ()

```
gnutls_pk_algorithm_t  gnutls_openpgp_privkey_get_pk_algorithm
                                                    (gnutls_openpgp_privkey_t key,
                                                     unsigned int *bits);
```

This function will return the public key algorithm of an OpenPGP certificate.

If bits is non null, it should have enough size to hold the parameters size in bits. For RSA the bits returned is the modulus. For DSA the bits returned are of the public exponent.

key : is an OpenPGP key

bits : if bits is non null it will hold the size of the parameters' in bits

Returns : a member of the gnutls_pk_algorithm_t enumeration on success, or a negative value on error.

Since 2.4.0

gnutls_openpgp_privkey_import ()

```
int                 gnutls_openpgp_privkey_import       (gnutls_openpgp_privkey_t key,
                                                         const gnutls_datum_t *data,
                                                         gnutls_openpgp_crt_fmt_t format,
                                                         const char *password,
                                                         unsigned int flags);
```

This function will convert the given RAW or Base64 encoded key to the native gnutls_openpgp_privkey_t format. The output will be stored in 'key'.

key : The structure to store the parsed key.

data : The RAW or BASE64 encoded key.

format : One of gnutls_openpgp_crt_fmt_t elements.

password : not used for now

flags : should be zero

Returns : GNUTLS_E_SUCCESS on success, or an error code.

gnutls_openpgp_privkey_sign_hash ()

```
int                 gnutls_openpgp_privkey_sign_hash    (gnutls_openpgp_privkey_t key,
                                                         const gnutls_datum_t *hash,
                                                         gnutls_datum_t *signature);
```

This function will sign the given hash using the private key. You should use gnutls_openpgp_privkey_set_preferred_key_id() before calling this function to set the subkey to use.

key : Holds the key

hash : holds the data to be signed

signature : will contain newly allocated signature

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_openpgp_privkey_get_fingerprint ()

```
int                 gnutls_openpgp_privkey_get_fingerprint
                                                    (gnutls_openpgp_privkey_t key,
                                                     void *fpr,
                                                     size_t *fprlen);
```

Get the fingerprint of the OpenPGP key. Depends on the algorithm, the fingerprint can be 16 or 20 bytes.

key : the raw data that contains the OpenPGP secret key.

fpr : the buffer to save the fingerprint, must hold at least 20 bytes.

fprlen : the integer to save the length of the fingerprint.

Returns : On success, 0 is returned, or an error code.

Since 2.4.0

gnutls_openpgp_privkey_get_subkey_fingerprint ()

```
int                 gnutls_openpgp_privkey_get_subkey_fingerprint
                                                    (gnutls_openpgp_privkey_t key,
                                                     unsigned int idx,
                                                     void *fpr,
                                                     size_t *fprlen);
```

Get the fingerprint of an OpenPGP subkey. Depends on the algorithm, the fingerprint can be 16 or 20 bytes.

key : the raw data that contains the OpenPGP secret key.

idx : the subkey index

fpr : the buffer to save the fingerprint, must hold at least 20 bytes.

fprlen : the integer to save the length of the fingerprint.

Returns : On success, 0 is returned, or an error code.

Since 2.4.0

gnutls_openpgp_privkey_get_key_id ()

```
int                 gnutls_openpgp_privkey_get_key_id   (gnutls_openpgp_privkey_t key,
                                                         gnutls_openpgp_keyid_t keyid);
```

Get key-id.

key : the structure that contains the OpenPGP secret key.

keyid : the buffer to save the keyid.

Returns : the 64-bit keyID of the OpenPGP key.

Since 2.4.0

gnutls_openpgp_privkey_get_subkey_count ()

```
int                 gnutls_openpgp_privkey_get_subkey_count
                                                    (gnutls_openpgp_privkey_t key);
```

This function will return the number of subkeys present in the given OpenPGP certificate.

key **:** is an OpenPGP key

Returns **:** the number of subkeys, or a negative value on error.

Since 2.4.0

gnutls_openpgp_privkey_get_subkey_idx ()

```
int                 gnutls_openpgp_privkey_get_subkey_idx
                                                    (gnutls_openpgp_privkey_t key,
                                                     const gnutls_openpgp_keyid_t keyid ↩
                                                         );
```

Get index of subkey.

key **:** the structure that contains the OpenPGP private key.

keyid **:** the keyid.

Returns **:** the index of the subkey or a negative error value.

Since 2.4.0

gnutls_openpgp_privkey_get_subkey_revoked_status ()

```
int                 gnutls_openpgp_privkey_get_subkey_revoked_status
                                                    (gnutls_openpgp_privkey_t key,
                                                     unsigned int idx);
```

Get revocation status of key.

key **:** the structure that contains the OpenPGP private key.

idx **:** is the subkey index

Returns **:** true (1) if the key has been revoked, or false (0) if it has not, or a negative value indicates an error.

Since 2.4.0

gnutls_openpgp_privkey_get_revoked_status ()

```
int                 gnutls_openpgp_privkey_get_revoked_status
                                                    (gnutls_openpgp_privkey_t key);
```

Get revocation status of key.

key **:** the structure that contains the OpenPGP private key.

Returns **:** true (1) if the key has been revoked, or false (0) if it has not, or a negative value indicates an error.

Since 2.4.0

gnutls_openpgp_privkey_get_subkey_pk_algorithm ()

```
gnutls_pk_algorithm_t  gnutls_openpgp_privkey_get_subkey_pk_algorithm
                                                        (gnutls_openpgp_privkey_t key,
                                                         unsigned int idx,
                                                         unsigned int *bits);
```

This function will return the public key algorithm of a subkey of an OpenPGP certificate.

If bits is non null, it should have enough size to hold the parameters size in bits. For RSA the bits returned is the modulus. For DSA the bits returned are of the public exponent.

key : is an OpenPGP key

idx : is the subkey index

bits : if bits is non null it will hold the size of the parameters' in bits

Returns : a member of the gnutls_pk_algorithm_t enumeration on success, or a negative value on error.

Since 2.4.0

gnutls_openpgp_privkey_get_subkey_expiration_time ()

```
time_t              gnutls_openpgp_privkey_get_subkey_expiration_time
                                                        (gnutls_openpgp_privkey_t key,
                                                         unsigned int idx);
```

Get subkey expiration time. A value of '0' means that the key doesn't expire at all.

key : the structure that contains the OpenPGP private key.

idx : the subkey index

Returns : the time when the OpenPGP key expires.

Since 2.4.0

gnutls_openpgp_privkey_get_subkey_id ()

```
int                 gnutls_openpgp_privkey_get_subkey_id
                                                        (gnutls_openpgp_privkey_t key,
                                                         unsigned int idx,
                                                         gnutls_openpgp_keyid_t keyid);
```

Get the key-id for the subkey.

key : the structure that contains the OpenPGP secret key.

idx : the subkey index

keyid : the buffer to save the keyid.

Returns : the 64-bit keyID of the OpenPGP key.

Since 2.4.0

gnutls_openpgp_privkey_get_subkey_creation_time ()

```
time_t              gnutls_openpgp_privkey_get_subkey_creation_time
                                                    (gnutls_openpgp_privkey_t key,
                                                     unsigned int idx);
```

Get subkey creation time.

key : the structure that contains the OpenPGP private key.

idx : the subkey index

Returns : the timestamp when the OpenPGP key was created.

Since 2.4.0

gnutls_openpgp_privkey_export_subkey_dsa_raw ()

```
int                 gnutls_openpgp_privkey_export_subkey_dsa_raw
                                                    (gnutls_openpgp_privkey_t pkey,
                                                     unsigned int idx,
                                                     gnutls_datum_t *p,
                                                     gnutls_datum_t *q,
                                                     gnutls_datum_t *g,
                                                     gnutls_datum_t *y,
                                                     gnutls_datum_t *x);
```

This function will export the DSA private key's parameters found in the given certificate. The new parameters will be allocated using gnutls_malloc() and will be stored in the appropriate datum.

pkey : Holds the certificate

idx : Is the subkey index

p : will hold the p

q : will hold the q

g : will hold the g

y : will hold the y

x : will hold the x

Returns : GNUTLS_E_SUCCESS on success, otherwise an error.

Since 2.4.0

gnutls_openpgp_privkey_export_subkey_rsa_raw ()

```
int                 gnutls_openpgp_privkey_export_subkey_rsa_raw
                                                    (gnutls_openpgp_privkey_t pkey,
                                                     unsigned int idx,
                                                     gnutls_datum_t *m,
                                                     gnutls_datum_t *e,
                                                     gnutls_datum_t *d,
                                                     gnutls_datum_t *p,
                                                     gnutls_datum_t *q,
                                                     gnutls_datum_t *u);
```

This function will export the RSA private key's parameters found in the given structure. The new parameters will be allocated using gnutls_malloc() and will be stored in the appropriate datum.

pkey **:** Holds the certificate

idx **:** Is the subkey index

m **:** will hold the modulus

e **:** will hold the public exponent

d **:** will hold the private exponent

p **:** will hold the first prime (p)

q **:** will hold the second prime (q)

u **:** will hold the coefficient

Returns **:** GNUTLS_E_SUCCESS on success, otherwise an error.

Since 2.4.0

gnutls_openpgp_privkey_export_dsa_raw ()

```
int                 gnutls_openpgp_privkey_export_dsa_raw
                                                    (gnutls_openpgp_privkey_t pkey,
                                                     gnutls_datum_t *p,
                                                     gnutls_datum_t *q,
                                                     gnutls_datum_t *g,
                                                     gnutls_datum_t *y,
                                                     gnutls_datum_t *x);
```

This function will export the DSA private key's parameters found in the given certificate. The new parameters will be allocated using gnutls_malloc() and will be stored in the appropriate datum.

pkey **:** Holds the certificate

p **:** will hold the p

q **:** will hold the q

g **:** will hold the g

y **:** will hold the y

x **:** will hold the x

Returns **:** GNUTLS_E_SUCCESS on success, otherwise an error.

Since 2.4.0

gnutls_openpgp_privkey_export_rsa_raw ()

```
int                 gnutls_openpgp_privkey_export_rsa_raw
                                                    (gnutls_openpgp_privkey_t pkey,
                                                     gnutls_datum_t *m,
                                                     gnutls_datum_t *e,
                                                     gnutls_datum_t *d,
                                                     gnutls_datum_t *p,
                                                     gnutls_datum_t *q,
                                                     gnutls_datum_t *u);
```

This function will export the RSA private key's parameters found in the given structure. The new parameters will be allocated using gnutls_malloc() and will be stored in the appropriate datum.

pkey : Holds the certificate

m : will hold the modulus

e : will hold the public exponent

d : will hold the private exponent

p : will hold the first prime (p)

q : will hold the second prime (q)

u : will hold the coefficient

Returns : GNUTLS_E_SUCCESS on success, otherwise an error.

Since 2.4.0

gnutls_openpgp_privkey_export ()

```
int                 gnutls_openpgp_privkey_export       (gnutls_openpgp_privkey_t key,
                                                         gnutls_openpgp_crt_fmt_t format,
                                                         const char *password,
                                                         unsigned int flags,
                                                         void *output_data,
                                                         size_t *output_data_size);
```

This function will convert the given key to RAW or Base64 format. If the buffer provided is not long enough to hold the output, then GNUTLS_E_SHORT_MEMORY_BUFFER will be returned.

key : Holds the key.

format : One of gnutls_openpgp_crt_fmt_t elements.

password : the password that will be used to encrypt the key. (unused for now)

flags : zero for future compatibility

output_data : will contain the key base64 encoded or raw

output_data_size : holds the size of output_data (and will be replaced by the actual size of parameters)

Returns : GNUTLS_E_SUCCESS on success, or an error code.

Since 2.4.0

gnutls_openpgp_privkey_set_preferred_key_id ()

```
int                 gnutls_openpgp_privkey_set_preferred_key_id
                                                        (gnutls_openpgp_privkey_t key,
                                                         const gnutls_openpgp_keyid_t keyid ↩
                                                             );
```

This allows setting a preferred key id for the given certificate. This key will be used by functions that involve key handling.

key : the structure that contains the OpenPGP public key.

keyid : the selected keyid

Returns : On success, 0 is returned, or an error code.

gnutls_openpgp_privkey_get_preferred_key_id ()

```
int                 gnutls_openpgp_privkey_get_preferred_key_id
                                                        (gnutls_openpgp_privkey_t key,
                                                         gnutls_openpgp_keyid_t keyid);
```

Get the preferred key-id for the key.

key : the structure that contains the OpenPGP public key.

keyid : the struct to save the keyid.

Returns : the 64-bit preferred keyID of the OpenPGP key, or if it hasn't been set it returns GNUTLS_E_INVALID_REQUEST.

gnutls_openpgp_crt_get_auth_subkey ()

```
int                 gnutls_openpgp_crt_get_auth_subkey  (gnutls_openpgp_crt_t crt,
                                                         gnutls_openpgp_keyid_t keyid,
                                                         unsigned int flag);
```

Returns the 64-bit keyID of the first valid OpenPGP subkey marked for authentication. If flag is non zero and no authentication subkey exists, then a valid subkey will be returned even if it is not marked for authentication. Returns the 64-bit keyID of the first valid OpenPGP subkey marked for authentication. If flag is non zero and no authentication subkey exists, then a valid subkey will be returned even if it is not marked for authentication.

crt : the structure that contains the OpenPGP public key.

keyid : the struct to save the keyid.

flag : Non zero indicates that a valid subkey is always returned.

Returns : GNUTLS_E_SUCCESS on success, or an error code.

gnutls_openpgp_keyring_init ()

```
int                 gnutls_openpgp_keyring_init         (gnutls_openpgp_keyring_t *keyring) ↩
    ;
```

This function will initialize an keyring structure.

keyring : The structure to be initialized

Returns : GNUTLS_E_SUCCESS on success, or an error code.

gnutls_openpgp_keyring_deinit ()

```
void                gnutls_openpgp_keyring_deinit       (gnutls_openpgp_keyring_t keyring);
```

This function will deinitialize a keyring structure.

keyring : The structure to be initialized

gnutls_openpgp_keyring_import ()

```
int                 gnutls_openpgp_keyring_import       (gnutls_openpgp_keyring_t keyring,
                                                         const gnutls_datum_t *data,
                                                         gnutls_openpgp_crt_fmt_t format);
```

This function will convert the given RAW or Base64 encoded keyring to the native gnutls_openpgp_keyring_t format. The output will be stored in 'keyring'.

keyring : The structure to store the parsed key.

data : The RAW or BASE64 encoded keyring.

format : One of gnutls_openpgp_keyring_fmt elements.

Returns : GNUTLS_E_SUCCESS on success, or an error code.

gnutls_openpgp_keyring_check_id ()

```
int                 gnutls_openpgp_keyring_check_id     (gnutls_openpgp_keyring_t ring,
                                                         const gnutls_openpgp_keyid_t keyid ↩
                                                             ,
                                                         unsigned int flags);
```

Check if a given key ID exists in the keyring.

ring : holds the keyring to check against

keyid : will hold the keyid to check for.

flags : unused (should be 0)

Returns : GNUTLS_E_SUCCESS on success (if keyid exists) and a negative error code on failure.

gnutls_openpgp_crt_verify_ring ()

```
int                 gnutls_openpgp_crt_verify_ring      (gnutls_openpgp_crt_t key,
                                                         gnutls_openpgp_keyring_t keyring,
                                                         unsigned int flags,
                                                         unsigned int *verify);
```

Verify all signatures in the key, using the given set of keys (keyring).

The key verification output will be put in *verify* and will be one or more of the gnutls_certificate_status_t enumerated elements bitwise or'd.

GNUTLS_CERT_INVALID: A signature on the key is invalid.

GNUTLS_CERT_REVOKED: The key has been revoked.

Note that this function does not verify using any "web of trust". You may use GnuPG for that purpose, or any other external PGP application.

key : the structure that holds the key.

keyring : holds the keyring to check against

flags : unused (should be 0)

verify : will hold the certificate verification output.

Returns : GNUTLS_E_SUCCESS on success, or an error code.

gnutls_openpgp_crt_verify_self ()

```
int                 gnutls_openpgp_crt_verify_self      (gnutls_openpgp_crt_t key,
                                                         unsigned int flags,
                                                         unsigned int *verify);
```

Verifies the self signature in the key. The key verification output will be put in *verify* and will be one or more of the gnutls_certificate_status_t enumerated elements bitwise or'd.

GNUTLS_CERT_INVALID: The self signature on the key is invalid.

key : the structure that holds the key.

flags : unused (should be 0)

verify : will hold the key verification output.

Returns : GNUTLS_E_SUCCESS on success, or an error code.

gnutls_openpgp_keyring_get_crt ()

```
int                 gnutls_openpgp_keyring_get_crt      (gnutls_openpgp_keyring_t ring,
                                                         unsigned int idx,
                                                         gnutls_openpgp_crt_t *cert);
```

This function will extract an OpenPGP certificate from the given keyring. If the index given is out of range GNUTLS_E_REQUESTED_I
will be returned. The returned structure needs to be deinited.

ring : Holds the keyring.

idx : the index of the certificate to export

cert : An uninitialized gnutls_openpgp_crt_t structure

Returns : GNUTLS_E_SUCCESS on success, or an error code.

gnutls_openpgp_keyring_get_crt_count ()

```
int                 gnutls_openpgp_keyring_get_crt_count
                                                        (gnutls_openpgp_keyring_t ring);
```

This function will return the number of OpenPGP certificates present in the given keyring.

ring : is an OpenPGP key ring

Returns : the number of subkeys, or a negative value on error.

gnutls_openpgp_recv_key_func ()

```
int                 (*gnutls_openpgp_recv_key_func)     (gnutls_session_t session,
                                                         unsigned char *keyfpr,
                                                         unsigned int keyfpr_length,
                                                         gnutls_datum_t *key);
```

A callback of this type is used to retrieve OpenPGP keys. Only useful on the server, and will only be used if the peer send a key fingerprint instead of a full key. See also gnutls_openpgp_set_recv_key_function().

session : a TLS session

keyfpr : key fingerprint

keyfpr_length : length of key fingerprint

key : output key.

Returns : On success, GNUTLS_E_SUCCESS (zero) is returned, otherwise an error code is returned.

gnutls_openpgp_set_recv_key_function ()

```
void                gnutls_openpgp_set_recv_key_function
                                                        (gnutls_session_t session,
                                                         gnutls_openpgp_recv_key_func func) ↩
                                                            ;
```

This funtion will set a key retrieval function for OpenPGP keys. This callback is only useful in server side, and will be used if the peer sent a key fingerprint instead of a full key.

session : a TLS session

func : the callback

gnutls_certificate_set_openpgp_key ()

```
int                 gnutls_certificate_set_openpgp_key  (gnutls_certificate_credentials_t  ↩
    res,
                                                         gnutls_openpgp_crt_t key,
                                                         gnutls_openpgp_privkey_t pkey);
```

This function sets a certificate/private key pair in the gnutls_certificate_credentials_t structure. This function may be called more than once (in case multiple keys/certificates exist for the server).

With this function the subkeys of the certificate are not used.

res : is a gnutls_certificate_credentials_t structure.

key : contains an openpgp public key

pkey : is an openpgp private key

Returns : On success, GNUTLS_E_SUCCESS (zero) is returned, otherwise an error code is returned.

gnutls_certificate_set_openpgp_key_file ()

```
int                 gnutls_certificate_set_openpgp_key_file
                                                    (gnutls_certificate_credentials_t  ↩
                                                       res,
                                                     const char *certfile,
                                                     const char *keyfile,
                                                     gnutls_openpgp_crt_fmt_t format);
```

This funtion is used to load OpenPGP keys into the GnuTLS credentials structure. The files should only contain one key which is not encrypted.

res : the destination context to save the data.

certfile : the file that contains the public key.

keyfile : the file that contains the secret key.

format : the format of the keys

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_certificate_set_openpgp_key_mem ()

```
int                 gnutls_certificate_set_openpgp_key_mem
                                                    (gnutls_certificate_credentials_t  ↩
                                                       res,
                                                     const gnutls_datum_t *cert,
                                                     const gnutls_datum_t *key,
                                                     gnutls_openpgp_crt_fmt_t format);
```

This funtion is used to load OpenPGP keys into the GnuTLS credential structure. The files should contain non encrypted keys.

res : the destination context to save the data.

cert : the datum that contains the public key.

key : the datum that contains the secret key.

format : the format of the keys

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_certificate_set_openpgp_key_file2 ()

```
int                 gnutls_certificate_set_openpgp_key_file2
                                                    (gnutls_certificate_credentials_t  ↩
                                                       res,
                                                     const char *certfile,
                                                     const char *keyfile,
                                                     const char *subkey_id,
                                                     gnutls_openpgp_crt_fmt_t format);
```

This funtion is used to load OpenPGP keys into the GnuTLS credential structure. The files should contain non encrypted keys.

The special keyword "auto" is also accepted as *subkey_id*. In that case the gnutls_openpgp_crt_get_auth_subkey() will be used to retrieve the subkey.

res : the destination context to save the data.

***certfile*:** the file that contains the public key.

***keyfile*:** the file that contains the secret key.

***subkey_id*:** a hex encoded subkey id

***format*:** the format of the keys

***Returns*:** On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

Since 2.4.0

gnutls_certificate_set_openpgp_key_mem2 ()

```
int                 gnutls_certificate_set_openpgp_key_mem2
                                                        (gnutls_certificate_credentials_t  ↩
                                                            res,
                                                         const gnutls_datum_t *cert,
                                                         const gnutls_datum_t *key,
                                                         const char *subkey_id,
                                                         gnutls_openpgp_crt_fmt_t format);
```

This funtion is used to load OpenPGP keys into the GnuTLS credentials structure. The files should only contain one key which is not encrypted.

The special keyword "auto" is also accepted as *subkey_id*. In that case the gnutls_openpgp_crt_get_auth_subkey() will be used to retrieve the subkey.

***res*:** the destination context to save the data.

***cert*:** the datum that contains the public key.

***key*:** the datum that contains the secret key.

***subkey_id*:** a hex encoded subkey id

***format*:** the format of the keys

***Returns*:** On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

Since 2.4.0

gnutls_certificate_set_openpgp_keyring_mem ()

```
int                 gnutls_certificate_set_openpgp_keyring_mem
                                                        (gnutls_certificate_credentials_t c ↩
                                                            ,
                                                         unsigned char *data,
                                                         size_t dlen,
                                                         gnutls_openpgp_crt_fmt_t format);
```

The function is used to set keyrings that will be used internally by various OpenPGP functions. For example to find a key when it is needed for an operations. The keyring will also be used at the verification functions.

***c*:** A certificate credentials structure

***data*:** buffer with keyring data.

***dlen*:** length of data buffer.

***format*:** the format of the keyring

***Returns*:** On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

gnutls_certificate_set_openpgp_keyring_file ()

```
int                 gnutls_certificate_set_openpgp_keyring_file
                                                    (gnutls_certificate_credentials_t c ↩
                                                       ,
                                                     const char *file,
                                                     gnutls_openpgp_crt_fmt_t format);
```

The function is used to set keyrings that will be used internally by various OpenPGP functions. For example to find a key when it is needed for an operations. The keyring will also be used at the verification functions.

c : A certificate credentials structure

file : filename of the keyring.

format : format of keyring.

Returns : On success, GNUTLS_E_SUCCESS is returned, otherwise a negative error value.

1.6 crypto

crypto —

Synopsis

```
typedef             gnutls_cipher_hd_t;
int                 gnutls_cipher_init              (gnutls_cipher_hd_t *handle,
                                                     gnutls_cipher_algorithm_t cipher,
                                                     const gnutls_datum_t *key,
                                                     const gnutls_datum_t *iv);
int                 gnutls_cipher_encrypt           (const gnutls_cipher_hd_t handle,
                                                     void *text,
                                                     size_t textlen);
int                 gnutls_cipher_decrypt           (const gnutls_cipher_hd_t handle,
                                                     void *ciphertext,
                                                     size_t ciphertextlen);
void                gnutls_cipher_deinit            (gnutls_cipher_hd_t handle);
int                 gnutls_cipher_get_block_size    (gnutls_cipher_algorithm_t algorit
typedef             gnutls_hash_hd_t;
typedef             gnutls_hmac_hd_t;
int                 gnutls_hmac_init                (gnutls_hmac_hd_t *dig,
                                                     gnutls_digest_algorithm_t algorit
                                                     const void *key,
                                                     size_t keylen);
int                 gnutls_hmac                     (gnutls_hmac_hd_t handle,
                                                     const void *text,
                                                     size_t textlen);
void                gnutls_hmac_output              (gnutls_hmac_hd_t handle,
                                                     void *digest);
void                gnutls_hmac_deinit              (gnutls_hmac_hd_t handle,
                                                     void *digest);
int                 gnutls_hmac_get_len             (gnutls_mac_algorithm_t algorithm)
int                 gnutls_hmac_fast                (gnutls_mac_algorithm_t algorithm,
                                                     const void *key,
                                                     size_t keylen,
```

```
                                                const void *text,
                                                size_t textlen,
                                                void *digest);
int                 gnutls_hash_init            (gnutls_hash_hd_t *dig,
                                                gnutls_digest_algorithm_t algorit
int                 gnutls_hash                 (gnutls_hash_hd_t handle,
                                                const void *text,
                                                size_t textlen);
void                gnutls_hash_output          (gnutls_hash_hd_t handle,
                                                void *digest);
void                gnutls_hash_deinit          (gnutls_hash_hd_t handle,
                                                void *digest);
int                 gnutls_hash_get_len         (gnutls_digest_algorithm_t algorit
int                 gnutls_hash_fast            (gnutls_digest_algorithm_t algorit
                                                const void *text,
                                                size_t textlen,
                                                void *digest);
#define             GNUTLS_CRYPTO_API_VERSION
#define             gnutls_crypto_single_cipher_st
#define             gnutls_crypto_single_mac_st
#define             gnutls_crypto_single_digest_st
int                 (*init)                     (gnutls_cipher_algorithm_t Param1,
                                                void **ctx);
int                 (*setkey)                   (void *ctx,
                                                const void *key,
                                                size_t keysize);
int                 (*setiv)                    (void *ctx,
                                                const void *iv,
                                                size_t ivsize);
int                 (*encrypt)                  (void *ctx,
                                                const void *plain,
                                                size_t plainsize,
                                                void *encr,
                                                size_t encrsize);
int                 (*decrypt)                  (void *ctx,
                                                const void *encr,
                                                size_t encrsize,
                                                void *plain,
                                                size_t plainsize);
void                (*deinit)                   (void *ctx);
int                 (*hash)                     (void *ctx,
                                                const void *text,
                                                size_t textsize);
int                 (*copy)                     (void **dst_ctx,
                                                void *src_ctx);
int                 (*output)                   (void *src_ctx,
                                                void *digest,
                                                size_t digestsize);
typedef             gnutls_crypto_digest_st;
enum                gnutls_rnd_level_t;
enum                gnutls_pk_flag_t;
int                 (*rnd)                      (void *ctx,
                                                int level,
                                                void *data,
                                                size_t datasize);
typedef             bigint_t;
enum                gnutls_bigint_format_t;
```

```
bigint_t            (*bigint_new)                   (int nbits);
void                (*bigint_release)               (bigint_t n);
int                 (*bigint_cmp)                   (const bigint_t m1,
                                                     const bigint_t m2);
int                 (*bigint_cmp_ui)                (const bigint_t m1,
                                                     unsigned long  m2);
bigint_t            (*bigint_mod)                   (const bigint_t a,
                                                     const bigint_t b);
bigint_t            (*bigint_set)                   (bigint_t a,
                                                     const bigint_t b);
bigint_t            (*bigint_set_ui)                (bigint_t a,
                                                     unsigned long  b);
unsigned            int                             ();
bigint_t            (*bigint_powm)                  (bigint_t w,
                                                     const bigint_t b,
                                                     const bigint_t e,
                                                     const bigint_t m);
bigint_t            (*bigint_addm)                  (bigint_t w,
                                                     const bigint_t a,
                                                     const bigint_t b,
                                                     const bigint_t m);
bigint_t            (*bigint_subm)                  (bigint_t w,
                                                     const bigint_t a,
                                                     const bigint_t b,
                                                     const bigint_t m);
bigint_t            (*bigint_mulm)                  (bigint_t w,
                                                     const bigint_t a,
                                                     const bigint_t b,
                                                     const bigint_t m);
bigint_t            (*bigint_mul)                   (bigint_t w,
                                                     const bigint_t a,
                                                     const bigint_t b);
bigint_t            (*bigint_add_ui)                (bigint_t w,
                                                     const bigint_t a,
                                                     unsigned long  b);
bigint_t            (*bigint_sub_ui)                (bigint_t w,
                                                     const bigint_t a,
                                                     unsigned long  b);
bigint_t            (*bigint_mul_ui)                (bigint_t w,
                                                     const bigint_t a,
                                                     unsigned long  b);
bigint_t            (*bigint_div)                   (bigint_t q,
                                                     const bigint_t a,
                                                     const bigint_t b);
int                 (*bigint_prime_check)           (const bigint_t pp);
int                 (*bigint_generate_group)        (gnutls_group_st *gg,
                                                     unsigned int bits);
bigint_t            (*bigint_scan)                  (const void *buf,
                                                     size_t buf_size,
                                                     gnutls_bigint_format_t format);
int                 (*bigint_print)                 (const bigint_t a,
                                                     void *buf,
                                                     size_t *buf_size,
                                                     gnutls_bigint_format_t format);
#define             GNUTLS_MAX_PK_PARAMS
void                gnutls_pk_params_release        (gnutls_pk_params_st *p);
void                gnutls_pk_params_init           (gnutls_pk_params_st *p);
```

```
enum                gnutls_direction_t;
int                 (*sign)                                 (gnutls_pk_algorithm_t Param1,
                                                             gnutls_datum_t *signature,
                                                             const gnutls_datum_t *data,
                                                             const gnutls_pk_params_st *privat
int                 (*verify)                               (gnutls_pk_algorithm_t Param1,
                                                             const gnutls_datum_t *data,
                                                             const gnutls_datum_t *signature,
                                                             const gnutls_pk_params_st *public
int                 (*generate)                             (gnutls_pk_algorithm_t Param1,
                                                             unsigned int nbits,
                                                             gnutls_pk_params_st *Param3);
int                 (*pk_fixup_private_params)              (gnutls_pk_algorithm_t Param1,
                                                             gnutls_direction_t Param2,
                                                             gnutls_pk_params_st *Param3);
#define             gnutls_crypto_single_cipher_register(algo,
                                                             prio,
                                                             st)
#define             gnutls_crypto_single_mac_register   (algo,
                                                             prio,
                                                             st)
#define             gnutls_crypto_single_digest_register(algo,
                                                             prio,
                                                             st)
int                 gnutls_crypto_single_cipher_register2
                                                            (gnutls_cipher_algorithm_t algorit
                                                             int priority,
                                                             int version,
                                                             const gnutls_crypto_single_cipher
int                 gnutls_crypto_single_mac_register2  (gnutls_mac_algorithm_t algorithm,
                                                             int priority,
                                                             int version,
                                                             const gnutls_crypto_single_mac_st
int                 gnutls_crypto_single_digest_register2
                                                            (gnutls_digest_algorithm_t algorit
                                                             int priority,
                                                             int version,
                                                             const gnutls_crypto_single_digest
#define             gnutls_crypto_cipher_register       (prio,
                                                             st)
#define             gnutls_crypto_mac_register          (prio,
                                                             st)
#define             gnutls_crypto_digest_register       (prio,
                                                             st)
int                 gnutls_crypto_cipher_register2      (int priority,
                                                             int version,
                                                             const gnutls_crypto_cipher_st *s)
int                 gnutls_crypto_mac_register2         (int priority,
                                                             int version,
                                                             const gnutls_crypto_mac_st *s);
int                 gnutls_crypto_digest_register2      (int priority,
                                                             int version,
                                                             const gnutls_crypto_digest_st *s)
#define             gnutls_crypto_rnd_register          (prio,
                                                             st)
#define             gnutls_crypto_pk_register           (prio,
                                                             st)
```

```
#define             gnutls_crypto_bigint_register       (prio,
                                                         st)
int                 gnutls_crypto_rnd_register2         (int priority,
                                                         int version,
                                                         const gnutls_crypto_rnd_st *s);
int                 gnutls_crypto_pk_register2          (int priority,
                                                         int version,
                                                         const gnutls_crypto_pk_st *s);
int                 gnutls_crypto_bigint_register2      (int priority,
                                                         int version,
                                                         const gnutls_crypto_bigint_st *s)
```

Description

Details

gnutls_cipher_hd_t

```
typedef struct cipher_hd_st *gnutls_cipher_hd_t;
```

gnutls_cipher_init ()

```
int                 gnutls_cipher_init                  (gnutls_cipher_hd_t *handle,
                                                         gnutls_cipher_algorithm_t cipher,
                                                         const gnutls_datum_t *key,
                                                         const gnutls_datum_t *iv);
```

This function will initialize an context that can be used for encryption/decryption of data. This will effectively use the current crypto backend in use by gnutls or the cryptographic accelerator in use.

handle : is a gnutls_cipher_hd_t structure.

cipher : the encryption algorithm to use

key : The key to be used for encryption

iv : The IV to use (if not applicable set NULL)

Returns : Zero or a negative value on error.

Since 2.10.0

gnutls_cipher_encrypt ()

```
int                 gnutls_cipher_encrypt               (const gnutls_cipher_hd_t handle,
                                                         void *text,
                                                         size_t textlen);
```

This function will encrypt the given data using the algorithm specified by the context.

handle : is a gnutls_cipher_hd_t structure.

text : the data to encrypt

textlen : The length of data to encrypt

Returns : Zero or a negative value on error.

Since 2.10.0

gnutls_cipher_decrypt ()

```
int                 gnutls_cipher_decrypt               (const gnutls_cipher_hd_t handle,
                                                         void *ciphertext,
                                                         size_t ciphertextlen);
```

This function will decrypt the given data using the algorithm specified by the context.

handle : is a gnutls_cipher_hd_t structure.

ciphertext : the data to encrypt

ciphertextlen : The length of data to encrypt

Returns : Zero or a negative value on error.

Since 2.10.0

gnutls_cipher_deinit ()

```
void                gnutls_cipher_deinit                (gnutls_cipher_hd_t handle);
```

This function will deinitialize all resources occupied by the given encryption context.

handle : is a gnutls_cipher_hd_t structure.

Since 2.10.0

gnutls_cipher_get_block_size ()

```
int                 gnutls_cipher_get_block_size        (gnutls_cipher_algorithm_t  ↩
    algorithm);
```

Get block size for encryption algorithm.

algorithm : is an encryption algorithm

Returns : block size for encryption algorithm.

Since 2.10.0

gnutls_hash_hd_t

```
typedef struct hash_hd_st *gnutls_hash_hd_t;
```

gnutls_hmac_hd_t

```
typedef struct hmac_hd_st *gnutls_hmac_hd_t;
```

gnutls_hmac_init ()

```
int                 gnutls_hmac_init                    (gnutls_hmac_hd_t *dig,
                                                         gnutls_digest_algorithm_t ↩
                                                             algorithm,
                                                         const void *key,
                                                         size_t keylen);
```

This function will initialize an context that can be used to produce a Message Authentication Code (MAC) of data. This will effectively use the current crypto backend in use by gnutls or the cryptographic accelerator in use.

dig : is a gnutls_hmac_hd_t structure.

algorithm : the HMAC algorithm to use

key : The key to be used for encryption

keylen : The length of the key

Returns : Zero or a negative value on error.

Since 2.10.0

gnutls_hmac ()

```
int                 gnutls_hmac                         (gnutls_hmac_hd_t handle,
                                                         const void *text,
                                                         size_t textlen);
```

This function will hash the given data using the algorithm specified by the context.

handle : is a gnutls_cipher_hd_t structure.

text : the data to hash

textlen : The length of data to hash

Returns : Zero or a negative value on error.

Since 2.10.0

gnutls_hmac_output ()

```
void                gnutls_hmac_output                  (gnutls_hmac_hd_t handle,
                                                         void *digest);
```

This function will output the current MAC value.

handle : is a gnutls_hmac_hd_t structure.

digest : is the output value of the MAC

Since 2.10.0

gnutls_hmac_deinit ()

```
void                gnutls_hmac_deinit                  (gnutls_hmac_hd_t handle,
                                                         void *digest);
```

This function will deinitialize all resources occupied by the given hmac context.

handle : is a gnutls_hmac_hd_t structure.

digest : is the output value of the MAC

Since 2.10.0

gnutls_hmac_get_len ()

```
int                 gnutls_hmac_get_len                 (gnutls_mac_algorithm_t algorithm);
```

This function will return the length of the output data of the given hmac algorithm.

algorithm : the hmac algorithm to use

Returns : The length or zero on error.

Since 2.10.0

gnutls_hmac_fast ()

```
int                 gnutls_hmac_fast                    (gnutls_mac_algorithm_t algorithm,
                                                         const void *key,
                                                         size_t keylen,
                                                         const void *text,
                                                         size_t textlen,
                                                         void *digest);
```

This convenience function will hash the given data and return output on a single call.

algorithm : the hash algorithm to use

key : the key to use

keylen : The length of the key

text : the data to hash

textlen : The length of data to hash

digest : is the output value of the hash

Returns : Zero or a negative value on error.

Since 2.10.0

gnutls_hash_init ()

```
int                 gnutls_hash_init                    (gnutls_hash_hd_t *dig,
                                                         gnutls_digest_algorithm_t ↩
                                                             algorithm);
```

This function will initialize an context that can be used to produce a Message Digest of data. This will effectively use the current crypto backend in use by gnuls or the cryptographic accelerator in use.

dig : is a gnutls_hash_hd_t structure.

algorithm : the hash algorithm to use

Returns : Zero or a negative value on error.

Since 2.10.0

gnutls_hash ()

```
int                 gnutls_hash                         (gnutls_hash_hd_t handle,
                                                         const void *text,
                                                         size_t textlen);
```

This function will hash the given data using the algorithm specified by the context.

handle : is a gnutls_cipher_hd_t structure.

text : the data to hash

textlen : The length of data to hash

Returns : Zero or a negative value on error.

Since 2.10.0

gnutls_hash_output ()

```
void                gnutls_hash_output                  (gnutls_hash_hd_t handle,
                                                         void *digest);
```

This function will output the current hash value.

handle : is a gnutls_hash_hd_t structure.

digest : is the output value of the hash

Since 2.10.0

gnutls_hash_deinit ()

```
void                gnutls_hash_deinit                  (gnutls_hash_hd_t handle,
                                                         void *digest);
```

This function will deinitialize all resources occupied by the given hash context.

handle : is a gnutls_hash_hd_t structure.

digest : is the output value of the hash

Since 2.10.0

gnutls_hash_get_len ()

```
int                 gnutls_hash_get_len                 (gnutls_digest_algorithm_t  ↩
    algorithm);
```

This function will return the length of the output data of the given hash algorithm.

algorithm **:** the hash algorithm to use

Returns **:** The length or zero on error.

Since 2.10.0

gnutls_hash_fast ()

```
int                 gnutls_hash_fast                    (gnutls_digest_algorithm_t  ↩
    algorithm,
                                                         const void *text,
                                                         size_t textlen,
                                                         void *digest);
```

This convenience function will hash the given data and return output on a single call.

algorithm **:** the hash algorithm to use

text **:** the data to hash

textlen **:** The length of data to hash

digest **:** is the output value of the hash

Returns **:** Zero or a negative value on error.

Since 2.10.0

GNUTLS_CRYPTO_API_VERSION

```
#define GNUTLS_CRYPTO_API_VERSION 0x02
```

gnutls_crypto_single_cipher_st

```
#define gnutls_crypto_single_cipher_st gnutls_crypto_cipher_st
```

gnutls_crypto_single_mac_st

```
#define gnutls_crypto_single_mac_st gnutls_crypto_mac_st
```

gnutls_crypto_single_digest_st

```
#define gnutls_crypto_single_digest_st gnutls_crypto_digest_st
```

init ()

```
int                 (*init)                             (gnutls_cipher_algorithm_t Param1,
                                                         void **ctx);
```

Param1 :

ctx :

Returns :

setkey ()

```
int                 (*setkey)                           (void *ctx,
                                                         const void *key,
                                                         size_t keysize);
```

ctx :

key :

keysize :

Returns :

setiv ()

```
int                 (*setiv)                            (void *ctx,
                                                         const void *iv,
                                                         size_t ivsize);
```

ctx :

iv :

ivsize :

Returns :

encrypt ()

```
int                 (*encrypt)                          (void *ctx,
                                                         const void *plain,
                                                         size_t plainsize,
                                                         void *encr,
                                                         size_t encrsize);
```

ctx :

plain :

plainsize :

encr :

encrsize :

Returns :

decrypt ()

```
int                 (*decrypt)                          (void *ctx,
                                                         const void *encr,
                                                         size_t encrsize,
                                                         void *plain,
                                                         size_t plainsize);
```

ctx :

encr :

encrsize :

plain :

plainsize :

Returns :

deinit ()

```
void                (*deinit)                           (void *ctx);
```

ctx :

hash ()

```
int                 (*hash)                             (void *ctx,
                                                         const void *text,
                                                         size_t textsize);
```

ctx :

text :

textsize :

Returns :

copy ()

```
int                 (*copy)                             (void **dst_ctx,
                                                         void *src_ctx);
```

dst_ctx :

src_ctx :

Returns :

output ()

```
int                 (*output)                           (void *src_ctx,
                                                         void *digest,
                                                         size_t digestsize);
```

src_ctx :

digest :

digestsize :

Returns :

gnutls_crypto_digest_st

```
typedef gnutls_crypto_mac_st gnutls_crypto_digest_st;
```

enum gnutls_rnd_level_t

```
typedef enum gnutls_rnd_level
{
  GNUTLS_RND_NONCE = 0,
  GNUTLS_RND_RANDOM = 1,
  GNUTLS_RND_KEY = 2
} gnutls_rnd_level_t;
```

Enumeration of random quality levels.

GNUTLS_RND_NONCE Non-predictable random number. Fatal in parts of session if broken, i.e., vulnerable to statistical analysis.

GNUTLS_RND_RANDOM Pseudo-random cryptographic random number. Fatal in session if broken.

GNUTLS_RND_KEY Fatal in many sessions if broken.

enum gnutls_pk_flag_t

```
typedef enum
{
  GNUTLS_PK_FLAG_NONE = 0
} gnutls_pk_flag_t;
```

Enumeration of public-key flag.

GNUTLS_PK_FLAG_NONE No flag.

rnd ()

```
int                 (*rnd)                              (void *ctx,
                                                         int level,
                                                         void *data,
                                                         size_t datasize);
```

ctx :

level :

data :

datasize :

Returns :

bigint_t

```
typedef void *bigint_t;
```

enum gnutls_bigint_format_t

```
typedef enum
{
  /* raw unsigned integer format */
  GNUTLS_MPI_FORMAT_USG = 0,
  /* raw signed integer format - always a leading zero when positive */
  GNUTLS_MPI_FORMAT_STD = 1,
  /* the pgp integer format */
  GNUTLS_MPI_FORMAT_PGP = 2
} gnutls_bigint_format_t;
```

Enumeration of different bignum integer encoding formats.

GNUTLS_MPI_FORMAT_USG Raw unsigned integer format.

GNUTLS_MPI_FORMAT_STD Raw signed integer format, always a leading zero when positive.

GNUTLS_MPI_FORMAT_PGP The pgp integer format.

bigint_new ()

```
bigint_t            (*bigint_new)                       (int nbits);
```

nbits :

Returns :

bigint_release ()

```
void                (*bigint_release)                   (bigint_t n);
```

n :

bigint_cmp ()

```
int                 (*bigint_cmp)                       (const bigint_t m1,
                                                         const bigint_t m2);
```

m1 :

m2 :

Returns :

bigint_cmp_ui ()

```
int                 (*bigint_cmp_ui)                    (const bigint_t m1,
                                                         unsigned long  m2);
```

m1 :

m2 :

Returns :

bigint_mod ()

```
bigint_t            (*bigint_mod)                       (const bigint_t a,
                                                         const bigint_t b);
```

a :

b :

Returns :

bigint_set ()

```
bigint_t            (*bigint_set)                       (bigint_t a,
                                                         const bigint_t b);
```

a :

b :

Returns :

bigint_set_ui ()

```
bigint_t            (*bigint_set_ui)                    (bigint_t a,
                                                         unsigned long  b);
```

a :

b :

Returns :

int ()

```
unsigned            int                                 ();
```

Returns :

bigint_powm ()

```
bigint_t            (*bigint_powm)                      (bigint_t w,
                                                         const bigint_t b,
                                                         const bigint_t e,
                                                         const bigint_t m);
```

w :

b :

e :

m :

Returns :

bigint_addm ()

```
bigint_t            (*bigint_addm)                      (bigint_t w,
                                                         const bigint_t a,
                                                         const bigint_t b,
                                                         const bigint_t m);
```

w :

a :

b :

m :

Returns :

bigint_subm ()

```
bigint_t            (*bigint_subm)                      (bigint_t w,
                                                         const bigint_t a,
                                                         const bigint_t b,
                                                         const bigint_t m);
```

w :

a :

b :

m :

Returns :

bigint_mulm ()

```
bigint_t            (*bigint_mulm)                      (bigint_t w,
                                                         const bigint_t a,
                                                         const bigint_t b,
                                                         const bigint_t m);
```

w :

a :

b :

m :

Returns :

bigint_mul ()

```
bigint_t            (*bigint_mul)                       (bigint_t w,
                                                         const bigint_t a,
                                                         const bigint_t b);
```

w :

a :

b :

Returns :

bigint_add_ui ()

```
bigint_t            (*bigint_add_ui)                    (bigint_t w,
                                                         const bigint_t a,
                                                         unsigned long  b);
```

w :

a :

b :

Returns :

bigint_sub_ui ()

```
bigint_t            (*bigint_sub_ui)                    (bigint_t w,
                                                         const bigint_t a,
                                                         unsigned long  b);
```

w :

a :

b :

Returns :

bigint_mul_ui ()

```
bigint_t            (*bigint_mul_ui)                    (bigint_t w,
                                                         const bigint_t a,
                                                         unsigned long  b);
```

w :

a :

b :

Returns :

bigint_div ()

```
bigint_t            (*bigint_div)                       (bigint_t q,
                                                         const bigint_t a,
                                                         const bigint_t b);
```

q :

a :

b :

Returns :

bigint_prime_check ()

```
int                 (*bigint_prime_check)               (const bigint_t pp);
```

pp :

Returns :

bigint_generate_group ()

```
int                 (*bigint_generate_group)            (gnutls_group_st *gg,
                                                         unsigned int bits);
```

gg :

bits :

Returns :

bigint_scan ()

```
bigint_t            (*bigint_scan)                      (const void *buf,
                                                         size_t buf_size,
                                                         gnutls_bigint_format_t format);
```

buf :

buf_size :

format :

Returns :

bigint_print ()

```
int                 (*bigint_print)                     (const bigint_t a,
                                                         void *buf,
                                                         size_t *buf_size,
                                                         gnutls_bigint_format_t format);
```

a :

buf :

buf_size :

format :

Returns :

GNUTLS_MAX_PK_PARAMS

```
#define GNUTLS_MAX_PK_PARAMS 6
```

gnutls_pk_params_release ()

```
void                gnutls_pk_params_release            (gnutls_pk_params_st *p);
```

p :

gnutls_pk_params_init ()

```
void                gnutls_pk_params_init               (gnutls_pk_params_st *p);
```

p :

enum gnutls_direction_t

```
typedef enum
{
  GNUTLS_IMPORT = 0,
  GNUTLS_EXPORT = 1
} gnutls_direction_t;
```

Enumeration of different directions.

GNUTLS_IMPORT Import direction.

GNUTLS_EXPORT Export direction.

sign ()

```
int                 (*sign)                             (gnutls_pk_algorithm_t Param1,
                                                         gnutls_datum_t *signature,
                                                         const gnutls_datum_t *data,
                                                         const gnutls_pk_params_st *private ↩
                                                             );
```

Param1 :

signature :

data :

private :

Returns :

verify ()

```
int                 (*verify)                           (gnutls_pk_algorithm_t Param1,
                                                         const gnutls_datum_t *data,
                                                         const gnutls_datum_t *signature,
                                                         const gnutls_pk_params_st *public) ↩
                                                             ;
```

Param1 :

data :

signature :

public :

Returns :

generate ()

```
int                 (*generate)                         (gnutls_pk_algorithm_t Param1,
                                                         unsigned int nbits,
                                                         gnutls_pk_params_st *Param3);
```

Param1 :

nbits :

Param3 :

Returns :

pk_fixup_private_params ()

```
int                 (*pk_fixup_private_params)          (gnutls_pk_algorithm_t Param1,
                                                         gnutls_direction_t Param2,
                                                         gnutls_pk_params_st *Param3);
```

Param1 :

Param2 :

Param3 :

Returns :

gnutls_crypto_single_cipher_register()

```
#define             gnutls_crypto_single_cipher_register(algo, prio, st)
```

***algo*:**

***prio*:**

***st*:**

gnutls_crypto_single_mac_register()

```
#define             gnutls_crypto_single_mac_register(algo, prio, st)
```

***algo*:**

***prio*:**

***st*:**

gnutls_crypto_single_digest_register()

```
#define             gnutls_crypto_single_digest_register(algo, prio, st)
```

***algo*:**

***prio*:**

***st*:**

gnutls_crypto_single_cipher_register2 ()

```
int                 gnutls_crypto_single_cipher_register2
                                                (gnutls_cipher_algorithm_t  ↩
                                                   algorithm,
                                                 int priority,
                                                 int version,
                                                 const  ↩
                                                     gnutls_crypto_single_cipher_st  ↩
                                                     *s);
```

This function will register a cipher algorithm to be used by gnutls. Any algorithm registered will override the included algorithms and by convention kernel implemented algorithms have priority of 90. The algorithm with the lowest priority will be used by gnutls.

This function should be called before gnutls_global_init().

For simplicity you can use the convenience gnutls_crypto_single_cipher_register() macro.

***algorithm*:** is the gnutls algorithm identifier

***priority*:** is the priority of the algorithm

***version*:** should be set to GNUTLS_CRYPTO_API_VERSION

***s*:** is a structure holding new cipher's data

***Returns*:** GNUTLS_E_SUCCESS on success, otherwise an error.

Since 2.6.0

gnutls_crypto_single_mac_register2 ()

```
int                 gnutls_crypto_single_mac_register2  (gnutls_mac_algorithm_t algorithm,
                                                         int priority,
                                                         int version,
                                                         const gnutls_crypto_single_mac_st ↩
                                                             *s);
```

This function will register a MAC algorithm to be used by gnutls. Any algorithm registered will override the included algorithms and by convention kernel implemented algorithms have priority of 90. The algorithm with the lowest priority will be used by gnutls.

This function should be called before gnutls_global_init().

For simplicity you can use the convenience gnutls_crypto_single_mac_register() macro.

algorithm : is the gnutls algorithm identifier

priority : is the priority of the algorithm

version : should be set to GNUTLS_CRYPTO_API_VERSION

s : is a structure holding new algorithms's data

Returns : GNUTLS_E_SUCCESS on success, otherwise an error.

Since 2.6.0

gnutls_crypto_single_digest_register2 ()

```
int                 gnutls_crypto_single_digest_register2
                                                        (gnutls_digest_algorithm_t ↩
                                                            algorithm,
                                                         int priority,
                                                         int version,
                                                         const ↩
                                                             gnutls_crypto_single_digest_st ↩
                                                             *s);
```

This function will register a digest (hash) algorithm to be used by gnutls. Any algorithm registered will override the included algorithms and by convention kernel implemented algorithms have priority of 90. The algorithm with the lowest priority will be used by gnutls.

This function should be called before gnutls_global_init().

For simplicity you can use the convenience gnutls_crypto_single_digest_register() macro.

algorithm : is the gnutls algorithm identifier

priority : is the priority of the algorithm

version : should be set to GNUTLS_CRYPTO_API_VERSION

s : is a structure holding new algorithms's data

Returns : GNUTLS_E_SUCCESS on success, otherwise an error.

Since 2.6.0

gnutls_crypto_cipher_register()

```
#define             gnutls_crypto_cipher_register(prio, st)
```

prio :

st :

gnutls_crypto_mac_register()

```
#define             gnutls_crypto_mac_register(prio, st)
```

prio :

st :

gnutls_crypto_digest_register()

```
#define             gnutls_crypto_digest_register(prio, st)
```

prio :

st :

gnutls_crypto_cipher_register2 ()

```
int                 gnutls_crypto_cipher_register2      (int priority,
                                                         int version,
                                                         const gnutls_crypto_cipher_st *s);
```

This function will register a cipher interface to be used by gnutls. Any interface registered will override the included engine and by convention kernel implemented interfaces should have priority of 90. The interface with the lowest priority will be used by gnutls.

This function should be called before gnutls_global_init().

For simplicity you can use the convenience gnutls_crypto_cipher_register() macro.

priority : is the priority of the cipher interface

version : should be set to GNUTLS_CRYPTO_API_VERSION

s : is a structure holding new interface's data

Returns : GNUTLS_E_SUCCESS on success, otherwise an error.

Since 2.6.0

gnutls_crypto_mac_register2 ()

```
int                 gnutls_crypto_mac_register2         (int priority,
                                                         int version,
                                                         const gnutls_crypto_mac_st *s);
```

This function will register a mac interface to be used by gnutls. Any interface registered will override the included engine and by convention kernel implemented interfaces should have priority of 90. The interface with the lowest priority will be used by gnutls.

This function should be called before gnutls_global_init().

For simplicity you can use the convenience gnutls_crypto_digest_register() macro.

priority : is the priority of the mac interface

version : should be set to GNUTLS_CRYPTO_API_VERSION

s : is a structure holding new interface's data

Returns : GNUTLS_E_SUCCESS on success, otherwise an error.

Since 2.6.0

gnutls_crypto_digest_register2 ()

```
int                 gnutls_crypto_digest_register2      (int priority,
                                                         int version,
                                                         const gnutls_crypto_digest_st *s);
```

This function will register a digest interface to be used by gnutls. Any interface registered will override the included engine and by convention kernel implemented interfaces should have priority of 90. The interface with the lowest priority will be used by gnutls.

This function should be called before gnutls_global_init().

For simplicity you can use the convenience gnutls_crypto_digest_register() macro.

priority : is the priority of the digest interface

version : should be set to GNUTLS_CRYPTO_API_VERSION

s : is a structure holding new interface's data

Returns : GNUTLS_E_SUCCESS on success, otherwise an error.

Since 2.6.0

gnutls_crypto_rnd_register()

```
#define             gnutls_crypto_rnd_register(prio, st)
```

prio :

st :

gnutls_crypto_pk_register()

```
#define             gnutls_crypto_pk_register(prio, st)
```

prio :

st :

gnutls_crypto_bigint_register()

```
#define             gnutls_crypto_bigint_register(prio, st)
```

prio :

st :

gnutls_crypto_rnd_register2 ()

```
int                 gnutls_crypto_rnd_register2         (int priority,
                                                         int version,
                                                         const gnutls_crypto_rnd_st *s);
```

This function will register a random generator to be used by gnutls. Any generator registered will override the included generator and by convention kernel implemented generators have priority of 90. The generator with the lowest priority will be used by gnutls.

This function should be called before gnutls_global_init().

For simplicity you can use the convenience gnutls_crypto_rnd_register() macro.

priority : is the priority of the generator

version : should be set to GNUTLS_CRYPTO_API_VERSION

s : is a structure holding new generator's data

Returns : GNUTLS_E_SUCCESS on success, otherwise an error.

Since 2.6.0

gnutls_crypto_pk_register2 ()

```
int                 gnutls_crypto_pk_register2          (int priority,
                                                         int version,
                                                         const gnutls_crypto_pk_st *s);
```

This function will register an interface for gnutls to operate on public key operations. Any interface registered will override the included interface. The interface with the lowest priority will be used by gnutls.

Note that the bigint interface must interoperate with the bigint interface. Thus if this interface is updated the gnutls_crypto_bigint_register should also be used.

This function should be called before gnutls_global_init().

For simplicity you can use the convenience gnutls_crypto_pk_register() macro.

priority : is the priority of the interface

version : should be set to GNUTLS_CRYPTO_API_VERSION

s : is a structure holding new interface's data

Returns : GNUTLS_E_SUCCESS on success, otherwise an error.

Since 2.6.0

gnutls_crypto_bigint_register2 ()

```
int                 gnutls_crypto_bigint_register2      (int priority,
                                                         int version,
                                                         const gnutls_crypto_bigint_st *s);
```

This function will register an interface for gnutls to operate on big integers. Any interface registered will override the included interface. The interface with the lowest priority will be used by gnutls.

Note that the bigint interface must interoperate with the public key interface. Thus if this interface is updated the gnutls_crypto_pk_registe should also be used.

This function should be called before gnutls_global_init().

For simplicity you can use the convenience gnutls_crypto_bigint_register() macro.

priority : is the priority of the interface

version : should be set to GNUTLS_CRYPTO_API_VERSION

s : is a structure holding new interface's data

Returns : GNUTLS_E_SUCCESS on success, otherwise an error.

Since 2.6.0

1.7 openssl

openssl —

Synopsis

```
#define             GNUTLS_X509_CN_SIZE
#define             GNUTLS_X509_C_SIZE
#define             GNUTLS_X509_O_SIZE
#define             GNUTLS_X509_OU_SIZE
#define             GNUTLS_X509_L_SIZE
#define             GNUTLS_X509_S_SIZE
#define             GNUTLS_X509_EMAIL_SIZE
#define             OPENSSL_VERSION_NUMBER
#define             SSLEAY_VERSION_NUMBER
#define             OPENSSL_VERSION_TEXT
#define             SSL_ERROR_NONE
#define             SSL_ERROR_SSL
#define             SSL_ERROR_WANT_READ
#define             SSL_ERROR_WANT_WRITE
#define             SSL_ERROR_SYSCALL
#define             SSL_ERROR_ZERO_RETURN
#define             SSL_FILETYPE_PEM
```

```
#define             SSL_VERIFY_NONE
#define             SSL_ST_OK
#define             X509_V_ERR_CERT_NOT_YET_VALID
#define             X509_V_ERR_CERT_HAS_EXPIRED
#define             X509_V_ERR_DEPTH_ZERO_SELF_SIGNED_CERT
#define             SSL_OP_ALL
#define             SSL_OP_NO_TLSv1
#define             SSL_MODE_ENABLE_PARTIAL_WRITE
#define             SSL_MODE_ACCEPT_MOVING_WRITE_BUFFER
#define             SSL_MODE_AUTO_RETRY
typedef             X509_NAME;
typedef             X509;
                    SSL;
#define             current_cert
#define             X509_STORE_CTX_get_current_cert     (ctx)
int                 (*verify_callback)                  (...,
                                                         X509_STORE_CTX *Param2);
#define             rbio
struct              rsa_st;
typedef             RSA;
#define             MD5_CTX
#define             RIPEMD160_CTX
#define             OpenSSL_add_ssl_algorithms
#define             SSLeay_add_ssl_algorithms
#define             SSLeay_add_all_algorithms
#define             SSL_get_cipher_name                 (ssl)
#define             SSL_get_cipher                      (ssl)
#define             SSL_get_cipher_bits                 (ssl,
                                                         bp)
#define             SSL_get_cipher_version              (ssl)
int                 SSL_library_init                    (void);
void                OpenSSL_add_all_algorithms          (void);
SSL_CTX *           SSL_CTX_new                         (SSL_METHOD *method);
void                SSL_CTX_free                        (SSL_CTX *ctx);
int                 SSL_CTX_set_default_verify_paths    (SSL_CTX *ctx);
int                 SSL_CTX_use_certificate_file        (SSL_CTX *ctx,
                                                         const char *certfile,
                                                         int type);
int                 SSL_CTX_use_PrivateKey_file         (SSL_CTX *ctx,
                                                         const char *keyfile,
                                                         int type);
void                SSL_CTX_set_verify                  (SSL_CTX *ctx,
                                                         int verify_mode,
                                                         int (verify_callbackint, X509_STO
unsigned long       SSL_CTX_set_options                 (SSL_CTX *ctx,
                                                         unsigned long  options);
long                SSL_CTX_set_mode                    (SSL_CTX *ctx,
                                                         long mode Param2);
int                 SSL_CTX_set_cipher_list             (SSL_CTX *ctx,
                                                         const char *list);
long                SSL_CTX_sess_number                 (SSL_CTX *ctx);
long                SSL_CTX_sess_connect                (SSL_CTX *ctx);
long                SSL_CTX_sess_connect_good           (SSL_CTX *ctx);
long                SSL_CTX_sess_connect_renegotiate    (SSL_CTX *ctx);
long                SSL_CTX_sess_accept                 (SSL_CTX *ctx);
long                SSL_CTX_sess_accept_good            (SSL_CTX *ctx);
long                SSL_CTX_sess_accept_renegotiate     (SSL_CTX *ctx);
```

```
long                SSL_CTX_sess_hits                (SSL_CTX *ctx);
long                SSL_CTX_sess_misses              (SSL_CTX *ctx);
long                SSL_CTX_sess_timeouts            (SSL_CTX *ctx);
SSL *               SSL_new                          (SSL_CTX *ctx);
void                SSL_free                         (SSL *ssl);
void                SSL_load_error_strings           (void);
int                 SSL_get_error                    (SSL *ssl,
                                                      int ret);
int                 SSL_set_fd                       (SSL *ssl,
                                                      int fd);
int                 SSL_set_rfd                      (SSL *ssl,
                                                      int fd);
int                 SSL_set_wfd                      (SSL *ssl,
                                                      int fd);
void                SSL_set_bio                      (SSL *ssl,
                                                      BIO *rbio,
                                                      BIO *wbio);
void                SSL_set_connect_state            (SSL *ssl);
int                 SSL_pending                      (SSL *ssl);
void                SSL_set_verify                   (SSL *ssl,
                                                      int verify_mode,
                                                      int (verify_callbackint, X509_STO
const X509 *        SSL_get_peer_certificate         (SSL *ssl);
int                 SSL_connect                      (SSL *ssl);
int                 SSL_accept                       (SSL *ssl);
int                 SSL_shutdown                     (SSL *ssl);
int                 SSL_read                         (SSL *ssl,
                                                      void *buf,
                                                      int len);
int                 SSL_write                        (SSL *ssl,
                                                      const void *buf,
                                                      int len);
int                 SSL_want                         (SSL *ssl);
#define             SSL_NOTHING
#define             SSL_WRITING
#define             SSL_READING
#define             SSL_X509_LOOKUP
#define             SSL_want_nothing                 (s)
#define             SSL_want_read                    (s)
#define             SSL_want_write                   (s)
#define             SSL_want_x509_lookup             (s)
SSL_METHOD *        SSLv23_client_method             (void);
SSL_METHOD *        SSLv23_server_method             (void);
SSL_METHOD *        SSLv3_client_method              (void);
SSL_METHOD *        SSLv3_server_method              (void);
SSL_METHOD *        TLSv1_client_method              (void);
SSL_METHOD *        TLSv1_server_method              (void);
SSL_CIPHER *        SSL_get_current_cipher           (SSL *ssl);
const char *        SSL_CIPHER_get_name              (SSL_CIPHER *cipher);
int                 SSL_CIPHER_get_bits              (SSL_CIPHER *cipher,
                                                      int *bits);
const char *        SSL_CIPHER_get_version           (SSL_CIPHER *cipher);
char *              SSL_CIPHER_description           (SSL_CIPHER *cipher,
                                                      char *buf,
                                                      int size);
X509_NAME *         X509_get_subject_name            (const X509 *cert);
X509_NAME *         X509_get_issuer_name             (const X509 *cert);
```

```
char *              X509_NAME_oneline               (gnutls_x509_dn *name,
                                                     char *buf,
                                                     int len);
void                X509_free                       (const X509 *cert);
void                BIO_get_fd                      (gnutls_session_t gnutls_state,
                                                     int *fd);
BIO *               BIO_new_socket                  (int sock,
                                                     int close_flag);
unsigned long       ERR_get_error                   (void);
const char *        ERR_error_string                (unsigned long  e,
                                                     char *buf);
int                 RAND_status                     (void);
void                RAND_seed                       (const void *buf,
                                                     int num);
int                 RAND_bytes                      (unsigned char *buf,
                                                     int num);
int                 RAND_pseudo_bytes               (unsigned char *buf,
                                                     int num);
const char *        RAND_file_name                  (char *buf,
                                                     size_t len);
int                 RAND_load_file                  (const char *name,
                                                     long maxbytes Param2);
int                 RAND_write_file                 (const char *name);
int                 RAND_egd_bytes                  (const char *path,
                                                     int bytes);
#define             RAND_egd                        (p)
#define             MD5_DIGEST_LENGTH
void                MD5_Init                        (MD5_CTX *ctx);
void                MD5_Update                      (MD5_CTX *ctx,
                                                     const void *buf,
                                                     int len);
void                MD5_Final                       (unsigned char *md,
                                                     MD5_CTX *ctx);
unsigned char *     MD5                             (unsigned char *buf,
                                                     unsigned long  len,
                                                     unsigned char *md);
void                RIPEMD160_Init                  (RIPEMD160_CTX *ctx);
void                RIPEMD160_Update                (RIPEMD160_CTX *ctx,
                                                     const void *buf,
                                                     int len);
void                RIPEMD160_Final                 (unsigned char *md,
                                                     RIPEMD160_CTX *ctx);
unsigned char *     RIPEMD160                       (unsigned char *buf,
                                                     unsigned long  len,
                                                     unsigned char *md);
```

Description

Details

GNUTLS_X509_CN_SIZE

```
#define GNUTLS_X509_CN_SIZE 256
```

GNUTLS_X509_C_SIZE

```
#define GNUTLS_X509_C_SIZE 3
```

GNUTLS_X509_O_SIZE

```
#define GNUTLS_X509_O_SIZE 256
```

GNUTLS_X509_OU_SIZE

```
#define GNUTLS_X509_OU_SIZE 256
```

GNUTLS_X509_L_SIZE

```
#define GNUTLS_X509_L_SIZE 256
```

GNUTLS_X509_S_SIZE

```
#define GNUTLS_X509_S_SIZE 256
```

GNUTLS_X509_EMAIL_SIZE

```
#define GNUTLS_X509_EMAIL_SIZE 256
```

OPENSSL_VERSION_NUMBER

```
#define OPENSSL_VERSION_NUMBER (0x0090604F)
```

SSLEAY_VERSION_NUMBER

```
#define SSLEAY_VERSION_NUMBER OPENSSL_VERSION_NUMBER
```

OPENSSL_VERSION_TEXT

```
#define OPENSSL_VERSION_TEXT ("GNUTLS " GNUTLS_VERSION " ")
```

SSL_ERROR_NONE

```
#define SSL_ERROR_NONE        (0)
```

SSL_ERROR_SSL

```
#define SSL_ERROR_SSL         (1)
```

SSL_ERROR_WANT_READ

```
#define SSL_ERROR_WANT_READ   (2)
```

SSL_ERROR_WANT_WRITE

```
#define SSL_ERROR_WANT_WRITE  (3)
```

SSL_ERROR_SYSCALL

```
#define SSL_ERROR_SYSCALL     (5)
```

SSL_ERROR_ZERO_RETURN

```
#define SSL_ERROR_ZERO_RETURN (6)
```

SSL_FILETYPE_PEM

```
#define SSL_FILETYPE_PEM (GNUTLS_X509_FMT_PEM)
```

SSL_VERIFY_NONE

```
#define SSL_VERIFY_NONE (0)
```

SSL_ST_OK

```
#define SSL_ST_OK (1)
```

X509_V_ERR_CERT_NOT_YET_VALID

```
#define X509_V_ERR_CERT_NOT_YET_VALID          (1)
```

X509_V_ERR_CERT_HAS_EXPIRED

```
#define X509_V_ERR_CERT_HAS_EXPIRED            (2)
```

X509_V_ERR_DEPTH_ZERO_SELF_SIGNED_CERT

```
#define X509_V_ERR_DEPTH_ZERO_SELF_SIGNED_CERT (3)
```

SSL_OP_ALL

```
#define SSL_OP_ALL (0x000FFFFF)
```

SSL_OP_NO_TLSv1

```
#define SSL_OP_NO_TLSv1 (0x0400000)
```

SSL_MODE_ENABLE_PARTIAL_WRITE

```
#define SSL_MODE_ENABLE_PARTIAL_WRITE (0x1)
```

SSL_MODE_ACCEPT_MOVING_WRITE_BUFFER

```
#define SSL_MODE_ACCEPT_MOVING_WRITE_BUFFER (0x2)
```

SSL_MODE_AUTO_RETRY

```
#define SSL_MODE_AUTO_RETRY (0x4)
```

X509_NAME

```
  typedef gnutls_x509_dn X509_NAME;
```

X509

```
  typedef gnutls_datum_t X509;
```

SSL

```
typedef struct {
    gnutls_session_t gnutls_state;

    gnutls_certificate_client_credentials gnutls_cred;

    SSL_CTX *ctx;
    SSL_CIPHER ciphersuite;

    int last_error;
    int shutdown;
    int state;
    unsigned long options;

    int (*verify_callback) (int, X509_STORE_CTX *);
    int verify_mode;

    gnutls_transport_ptr_t rfd;
    gnutls_transport_ptr_t wfd;
} SSL;
```

current_cert

```
#define current_cert cert_list
```

X509_STORE_CTX_get_current_cert()

```
#define X509_STORE_CTX_get_current_cert(ctx) ((ctx)->current_cert)
```

ctx **:**

verify_callback ()

```
int                 (*verify_callback)                  (...,
                                                         X509_STORE_CTX *Param2);
```

... **:**

Param2 **:**

Returns **:**

rbio

```
#define rbio gnutls_state
```

struct rsa_st

```
struct rsa_st;
```

RSA

```
  typedef struct rsa_st RSA;
```

MD5_CTX

```
#define MD5_CTX MD_CTX
```

RIPEMD160_CTX

```
#define RIPEMD160_CTX MD_CTX
```

OpenSSL_add_ssl_algorithms

```
#define OpenSSL_add_ssl_algorithms()  SSL_library_init()
```

SSLeay_add_ssl_algorithms

```
#define SSLeay_add_ssl_algorithms()   SSL_library_init()
```

SSLeay_add_all_algorithms

```
#define SSLeay_add_all_algorithms()   OpenSSL_add_all_algorithms()
```

SSL_get_cipher_name()

```
#define SSL_get_cipher_name(ssl) SSL_CIPHER_get_name(SSL_get_current_cipher(ssl))
```

ssl :

SSL_get_cipher()

```
#define SSL_get_cipher(ssl) SSL_get_cipher_name(ssl)
```

ssl :

SSL_get_cipher_bits()

```
#define SSL_get_cipher_bits(ssl,bp) SSL_CIPHER_get_bits(SSL_get_current_cipher(ssl),(bp))
```

ssl :

bp :

SSL_get_cipher_version()

```
#define SSL_get_cipher_version(ssl) SSL_CIPHER_get_version(SSL_get_current_cipher(ssl))
```

ssl :

SSL_library_init ()

```
int                 SSL_library_init                    (void);
```

Returns :

OpenSSL_add_all_algorithms ()

```
void                OpenSSL_add_all_algorithms          (void);
```

SSL_CTX_new ()

```
SSL_CTX *           SSL_CTX_new                         (SSL_METHOD *method);
```

method :

Returns :

SSL_CTX_free ()

```
void                SSL_CTX_free                        (SSL_CTX *ctx);
```

ctx :

SSL_CTX_set_default_verify_paths ()

```
int                 SSL_CTX_set_default_verify_paths    (SSL_CTX *ctx);
```

ctx :

Returns :

SSL_CTX_use_certificate_file ()

```
int                 SSL_CTX_use_certificate_file        (SSL_CTX *ctx,
                                                         const char *certfile,
                                                         int type);
```

ctx :

certfile :

type :

Returns :

SSL_CTX_use_PrivateKey_file ()

```
int                 SSL_CTX_use_PrivateKey_file         (SSL_CTX *ctx,
                                                         const char *keyfile,
                                                         int type);
```

ctx :

keyfile :

type :

Returns :

SSL_CTX_set_verify ()

```
void                SSL_CTX_set_verify                  (SSL_CTX *ctx,
                                                         int verify_mode,
                                                         int (verify_callbackint,  ↩
                                                             X509_STORE_CTX *) ());
```

ctx :

verify_mode : *int*, X509_STORE_CTX *:

SSL_CTX_set_options ()

```
unsigned long       SSL_CTX_set_options                 (SSL_CTX *ctx,
                                                         unsigned long  options);
```

ctx :

options :

Returns :

SSL_CTX_set_mode ()

```
long                SSL_CTX_set_mode                    (SSL_CTX *ctx,
                                                         long mode Param2);
```

ctx :

Param2 :

Returns :

SSL_CTX_set_cipher_list ()

```
int                 SSL_CTX_set_cipher_list             (SSL_CTX *ctx,
                                                         const char *list);
```

ctx :

list :

Returns :

SSL_CTX_sess_number ()

```
long                SSL_CTX_sess_number                 (SSL_CTX *ctx);
```

ctx :

Returns :

SSL_CTX_sess_connect ()

```
long                SSL_CTX_sess_connect                (SSL_CTX *ctx);
```

ctx :

Returns :

SSL_CTX_sess_connect_good ()

```
long                SSL_CTX_sess_connect_good           (SSL_CTX *ctx);
```

ctx :

Returns :

SSL_CTX_sess_connect_renegotiate ()

```
long                SSL_CTX_sess_connect_renegotiate    (SSL_CTX *ctx);
```

ctx :

Returns :

SSL_CTX_sess_accept ()

```
long                SSL_CTX_sess_accept                 (SSL_CTX *ctx);
```

ctx :

Returns :

SSL_CTX_sess_accept_good ()

```
long                SSL_CTX_sess_accept_good            (SSL_CTX *ctx);
```

ctx :

Returns :

SSL_CTX_sess_accept_renegotiate ()

```
long                SSL_CTX_sess_accept_renegotiate     (SSL_CTX *ctx);
```

ctx :

Returns :

SSL_CTX_sess_hits ()

```
long                SSL_CTX_sess_hits                   (SSL_CTX *ctx);
```

ctx :

Returns :

SSL_CTX_sess_misses ()

```
long                SSL_CTX_sess_misses             (SSL_CTX *ctx);
```

ctx :

Returns :

SSL_CTX_sess_timeouts ()

```
long                SSL_CTX_sess_timeouts           (SSL_CTX *ctx);
```

ctx :

Returns :

SSL_new ()

```
SSL *               SSL_new                         (SSL_CTX *ctx);
```

ctx :

Returns :

SSL_free ()

```
void                SSL_free                        (SSL *ssl);
```

ssl :

SSL_load_error_strings ()

```
void                SSL_load_error_strings          (void);
```

SSL_get_error ()

```
int                 SSL_get_error                   (SSL *ssl,
                                                     int ret);
```

ssl :

ret :

Returns :

SSL_set_fd ()

```
int                 SSL_set_fd                          (SSL *ssl,
                                                         int fd);
```

ssl :

fd :

Returns :

SSL_set_rfd ()

```
int                 SSL_set_rfd                         (SSL *ssl,
                                                         int fd);
```

ssl :

fd :

Returns :

SSL_set_wfd ()

```
int                 SSL_set_wfd                         (SSL *ssl,
                                                         int fd);
```

ssl :

fd :

Returns :

SSL_set_bio ()

```
void                SSL_set_bio                         (SSL *ssl,
                                                         BIO *rbio,
                                                         BIO *wbio);
```

ssl :

rbio :

wbio :

SSL_set_connect_state ()

```
void                SSL_set_connect_state               (SSL *ssl);
```

ssl :

SSL_pending ()

```
int                 SSL_pending                         (SSL *ssl);
```

ssl :

Returns :

SSL_set_verify ()

```
void                SSL_set_verify                      (SSL *ssl,
                                                         int verify_mode,
                                                         int (verify_callbackint,  ↩
                                                             X509_STORE_CTX *) ());
```

ssl :

verify_mode : *int*, X509_STORE_CTX *:

SSL_get_peer_certificate ()

```
const X509 *        SSL_get_peer_certificate            (SSL *ssl);
```

ssl :

Returns :

SSL_connect ()

```
int                 SSL_connect                         (SSL *ssl);
```

ssl :

Returns :

SSL_accept ()

```
int                 SSL_accept                          (SSL *ssl);
```

ssl :

Returns :

SSL_shutdown ()

```
int                 SSL_shutdown                        (SSL *ssl);
```

ssl :

Returns :

SSL_read ()

```
int                 SSL_read                            (SSL *ssl,
                                                         void *buf,
                                                         int len);
```

ssl :

buf :

len :

Returns :

SSL_write ()

```
int                 SSL_write                           (SSL *ssl,
                                                         const void *buf,
                                                         int len);
```

ssl :

buf :

len :

Returns :

SSL_want ()

```
int                 SSL_want                            (SSL *ssl);
```

ssl :

Returns :

SSL_NOTHING

```
#define SSL_NOTHING (1)
```

SSL_WRITING

```
#define SSL_WRITING (2)
```

SSL_READING

```
#define SSL_READING (3)
```

SSL_X509_LOOKUP

```
#define SSL_X509_LOOKUP (4)
```

SSL_want_nothing()

```
#define SSL_want_nothing(s) (SSL_want(s) == SSL_NOTHING)
```

s :

SSL_want_read()

```
#define SSL_want_read(s) (SSL_want(s) == SSL_READING)
```

s :

SSL_want_write()

```
#define SSL_want_write(s) (SSL_want(s) == SSL_WRITING)
```

s :

SSL_want_x509_lookup()

```
#define SSL_want_x509_lookup(s) (SSL_want(s) == SSL_X509_LOOKUP)
```

s :

SSLv23_client_method ()

```
SSL_METHOD *        SSLv23_client_method                (void);
```

Returns :

SSLv23_server_method ()

```
SSL_METHOD *        SSLv23_server_method                (void);
```

Returns :

SSLv3_client_method ()

```
SSL_METHOD *        SSLv3_client_method                 (void);
```

Returns :

SSLv3_server_method ()

```
SSL_METHOD *        SSLv3_server_method                 (void);
```

Returns :

TLSv1_client_method ()

```
SSL_METHOD *        TLSv1_client_method                 (void);
```

Returns :

TLSv1_server_method ()

```
SSL_METHOD *        TLSv1_server_method                 (void);
```

Returns :

SSL_get_current_cipher ()

```
SSL_CIPHER *        SSL_get_current_cipher              (SSL *ssl);
```

ssl :

Returns :

SSL_CIPHER_get_name ()

```
const char *        SSL_CIPHER_get_name                 (SSL_CIPHER *cipher);
```

cipher :

Returns :

SSL_CIPHER_get_bits ()

```
int                 SSL_CIPHER_get_bits                 (SSL_CIPHER *cipher,
                                                         int *bits);
```

cipher :

bits :

Returns :

SSL_CIPHER_get_version ()

```
const char *        SSL_CIPHER_get_version              (SSL_CIPHER *cipher);
```

cipher :

Returns :

SSL_CIPHER_description ()

```
char *              SSL_CIPHER_description              (SSL_CIPHER *cipher,
                                                         char *buf,
                                                         int size);
```

cipher :

buf :

size :

Returns :

X509_get_subject_name ()

```
X509_NAME *         X509_get_subject_name               (const X509 *cert);
```

cert :

Returns :

X509_get_issuer_name ()

```
X509_NAME *         X509_get_issuer_name                (const X509 *cert);
```

cert :

Returns :

X509_NAME_oneline ()

```
char *              X509_NAME_oneline                   (gnutls_x509_dn *name,
                                                         char *buf,
                                                         int len);
```

name :

buf :

len :

Returns :

X509_free ()

```
void                X509_free                           (const X509 *cert);
```

cert :

BIO_get_fd ()

```
void                BIO_get_fd                      (gnutls_session_t gnutls_state,
                                                     int *fd);
```

gnutls_state :

fd :

BIO_new_socket ()

```
BIO *               BIO_new_socket                  (int sock,
                                                     int close_flag);
```

sock :

close_flag :

Returns :

ERR_get_error ()

```
unsigned long       ERR_get_error                   (void);
```

Returns :

ERR_error_string ()

```
const char *        ERR_error_string                (unsigned long  e,
                                                     char *buf);
```

e :

buf :

Returns :

RAND_status ()

```
int                 RAND_status                     (void);
```

Returns :

RAND_seed ()

```
void                RAND_seed                       (const void *buf,
                                                     int num);
```

buf :

num :

RAND_bytes ()

```
int                 RAND_bytes                          (unsigned char *buf,
                                                         int num);
```

buf :

num :

Returns :

RAND_pseudo_bytes ()

```
int                 RAND_pseudo_bytes                   (unsigned char *buf,
                                                         int num);
```

buf :

num :

Returns :

RAND_file_name ()

```
const char *        RAND_file_name                      (char *buf,
                                                         size_t len);
```

buf :

len :

Returns :

RAND_load_file ()

```
int                 RAND_load_file                      (const char *name,
                                                         long maxbytes Param2);
```

name :

Param2 :

Returns :

RAND_write_file ()

```
int                 RAND_write_file                     (const char *name);
```

name :

Returns :

RAND_egd_bytes ()

```
int                 RAND_egd_bytes                      (const char *path,
                                                         int bytes);
```

path :

bytes :

Returns :

RAND_egd()

```
#define RAND_egd(p) RAND_egd_bytes((p), 255)
```

p :

MD5_DIGEST_LENGTH

```
#define MD5_DIGEST_LENGTH 16
```

MD5_Init ()

```
void                MD5_Init                            (MD5_CTX *ctx);
```

ctx :

MD5_Update ()

```
void                MD5_Update                          (MD5_CTX *ctx,
                                                         const void *buf,
                                                         int len);
```

ctx :

buf :

len :

MD5_Final ()

```
void                MD5_Final                           (unsigned char *md,
                                                         MD5_CTX *ctx);
```

md :

ctx :

MD5 ()

```
unsigned char *     MD5                                 (unsigned char *buf,
                                                         unsigned long  len,
                                                         unsigned char *md);
```

buf :

len :

md :

Returns :

RIPEMD160_Init ()

```
void                RIPEMD160_Init                      (RIPEMD160_CTX *ctx);
```

ctx :

RIPEMD160_Update ()

```
void                RIPEMD160_Update                    (RIPEMD160_CTX *ctx,
                                                         const void *buf,
                                                         int len);
```

ctx :

buf :

len :

RIPEMD160_Final ()

```
void                RIPEMD160_Final                     (unsigned char *md,
                                                         RIPEMD160_CTX *ctx);
```

md :

ctx :

RIPEMD160 ()

```
unsigned char *     RIPEMD160                           (unsigned char *buf,
                                                         unsigned long  len,
                                                         unsigned char *md);
```

buf :

len :

md :

Returns :

Chapter 2

Index