GNUTLS

a Transport Layer Security Library

This document applies to *GNUTLS* 0.3.5

By Nikos Mavroyanopoulos and Fabio Fiorina

Contents

Chapter 1

The Library

1.1 Introduction

GNUTLS is a portable library which implements the *TLS 1.0* and *SSL 3.0* protocols. *TLS* stands for 'Transport Layer Security' and is the sucessor of *SSL* [1]. *TLS 1.0* [2] is an Internet protocol, defined by IETF[3] that provides confidentiality, and authentication layers over a reliable transport layer[4]. *GNUTLS* implements the above protocols in reentrant way in order to be used in multiple threads of execution (without the need for Critical Sections and locks). See http://www.gnutls.org/ and http://www.gnu.org/software/gnutls/ for updated versions of the *GNUTLS* software and this document.

Currently *GNUTLS* implements:

- the *TLS 1.0* and *SSL 3.0* protocols, without any weak algorithms[5]
- **X.509** Public Key Infrastructure (with several limitations).
- **SRP** for *TLS* authentication.
- *TLS* **Extension mechanism**

[1] SSL or Secure Sockets Layer is a protocol designed by Netscape. TLS 1.0 is based on SSL 3.0 protocol. SSL 2.0 is a very old protocol which is not considered secure today. SSL 2.0 is not implemented in GNUTLS

[2] described in RFC 2246

[3] IETF or Internet Engineering Task Force is a large open international community of network designers, operators, vendors, and researchers concerned with the evolution of the Internet architecture and the smooth operation of the Internet. It is open to any interested individual.

[4] TLS is mostly used over TCP/IP although this is not restrictive, you may use it over any reliable transport layer.

[5] There are ciphersuites in TLS 1.0 that are considered weak. These ciphersuites are deliberately weak in order to be able to export encryption software from some countries.

1.2 TLS Cipher suites

TLS 1.0 supports ciphersuites like **TLS_X509PKI_DHE_RSA_WITH_3DES_CBC_SHA**. These ciphersuites contain three parameters:

- The authentication method (X.509 PKI in the example)[6]
- The key exchange algorithm (DHE_RSA in the example)
- The Symmetric encryption algorithm and mode (3DES_CBC in this example)
- The MAC[7] algorithm used for authentication. MAC_SHA is used in the above example.

The ciphersuite that will be used in the connection is negotiated at the handshake procedure. However you must note that *TLS 1.0* does not always negotiate the strongest available cipher suite. There are cases where a man in the middle attacker could make the two entities negotiate the least secure method they support. For that reason do not enable ciphers and algorithms that you consider weak.

1.3 Symmetric encryption algorithms

Confidentiality is provided by using block encryption algorithms like **3DES**, **AES**[8], or stream algorithms like **ARCFOUR**[9] See figure 1.1 for a complete list. Ciphers are encryption algorithms that use a single (secret) key to encrypt and decrypt data. Block algorithms in TLS also provide protection against statistical analysis of the data. *GNUTLS* makes use of this property thus, if you're operating in *TLS 1.0* mode, a random number of blocks will be appended to the data. This will prevent eavesdroppers from guessing the actual data size.

[6] This field should be considered as a GNUTLS extension. TLS 1.0 only defines the X.509 PKI authentication method. GNUTLS supports other authentication methods as well.

[7] MAC stands for Message Authentication Code. It can be described as a keyed hash algorithm. See RFC2104.

[8] AES or Advanced Encryption Standard is actually the RIJNDAEL algorithm. This is the algorithm that will replace DES.

[9] ARCFOUR is a compatible algorithm with RSA's RC4 algorithm.

3DES_CBC	3DES_CBC is the DES block cipher algorithm used with multiple (triple) encryption (EDE). Has 64 bits block size and is used in CBC mode.
ARCFOUR	ARCFOUR[10] is a fast stream cipher.
AES_CBC	AES or RIJNDAEL is the block cipher algorithm that replaces the old (and insecure[11]) DES algorithm. Has 128 bits block size and is used in CBC mode. This is not officially supported in TLS (yet).
TWOFISH_CBC	TWOFISH is a block cipher algorithm by Counterpane. Has 128 bits block size and is used in CBC mode. This algorithm is not part of TLS. It is a *GNUTLS* extension.

Figure 1.1: Supported cipher algorithms

MAC_MD5	MD5 is a hash algorithm by Ron Rivest. Outputs 128 bits of data.
MAC_SHA	SHA is a hash algorithm by NSA. Outputs 160 bits of data.

Figure 1.2: Supported MAC algorithms

1.4 Authentication methods

The following authentication schemas are supported in *GNUTLS* :

1. X509 Public Key Infrastructure
2. Anonymous authentication
3. SRP authentication

1.4.1 Authentication using X.509 certificates

If using this kind of authentication then the key exchange methods shown in figure 1.3 are available to use. Authentication in this method is performed using signed certificates by a trusted Certificate Authority (CA). Note that *GNUTLS* is not a generic purpose X.509 toolkit[12]. It does only include the required, in order to use the TLS ciphersuites which require X.509 certificates.

1.4.2 Anonymous authentication

The anonymous key exchanges perform encryption but there is no indication of the identity of the peer. This kind of authentication is vulnerable to man in the middle attack, but this protocol can be used even if there is no prior communication or common trusted parties with the peer. Unless really required,

[12] Aegypten is such a toolkit. See http://www.gnupg.org/aegypten/

X509PKI_RSA	The RSA algorithm is used to encrypt a key and send it to the peer. The certificate must allow the key to be used for encryption.
X509PKI_DHE_RSA	The RSA algorithm is used to sign Ephemeral Diffie Hellman parameters which are send to the peer. The key in the certificate must allow the key to be used for signing
X509PKI_DHE_DSS	The DSS[13] algorithm is used to sign Ephemeral Diffie Hellman parameters which are send to the peer. Currently *GNUTLS* does not support this ciphersuite.

Figure 1.3: Supported X.509 key exchange algorithms

do not use anonymous authentication. Available key exchange methods are shown in figure 1.4.

ANON_DH	This algorithm exchanges Diffie Hellman parameters.

Figure 1.4: Supported anonymous key exchange algorithms

1.4.3 Authentication using SRP

Authentication using the SRP[14] is actually password authentication, since the two peers are identified by the knowledge of a password. This protocol also offers protection against off-line attacks (password file stealing etc). This is achieved since SRP does not use the plain password to perform authentication, but something called a verifier. The verifier is $g^x mod(n)$ and x is a value calculated from the username and the password.

SRP is normaly used with a SHA based hash function, to calculate the value of x. In *GNUTLS* in addition to original SHA hash function, a hash function based on blowfish crypt is also supported. The blowfish crypt function has the property of variable complexity, thus the verifier may resist future attacks based on computational power, by just increasing the complexity of the function (sometimes called 'the cost').

The advantage of SRP authentication, over other proposed secure password authentication schemas, is that SRP does not require the server to hold the user's password. This kind of protection is similar to the one used traditionaly in the *UNIX* 'passwd' file, where the contents of this file did not cause harm to the system security if they were revealed.

Available key exchange methods are shown in figure 1.5.

[14]SRP stands for Secure Password Protocol and is described in RFC2945. The SRP key exchange is not a part of the TLS 1.0 protocol

SRP	Authentication using the SRP protocol.

Figure 1.5: Supported SRP key exchange algorithms

1.5 Resuming Sessions

The gnutls_handshake() (see Section 1.10.18) function, is expensive since a lot of calculations are performed. In order to support many fast connections to the same server a client may use session resuming. **Session resuming** is a feature of the **TLS** protocol which allows a client to connect to a server, after a successful handshake, without the expensive calculations (by using the previously established keys). *GNUTLS* supports this feature, and the example resume client (see Section 1.8.3) illustrates a typical use of it (This is a modification of the simple client example). Servers only need to use the gnutls_db_set_name() (see Section 1.10.42) function if they want to use the gdbm backend to store sessions.

Keep in mind that sessions are expired after some time (for security reasons), thus it may be normal for a server not to resume a session even if you requested that. Also note that you must enable (using the priority functions), at least the algorithms used in the last session.

1.5.1 Resuming internals

The resuming capability (mostly in the server side) is one of the problems of a thread-safe TLS implementations. The problem is that all threads must share information in order to be able to resume sessions. The gnutls approach is, in case of a client, to leave all the burden of resuming to the client (ie. copy and keep the nesessary parameters). See gnutls_session_get_data() on section 1.10.33, gnutls_session_get_id() on section 1.10.34 and gnutls_session_set_data() on section 1.10.35.

The server side is different. Here the server only specifies a DB file, using gnutls_db_set_name() (see Section 1.10.42). This DB file is used to store the sessions' required parameters for resuming. This means that this file contains very sensitive information, such as encryption keys. In a multi-threaded application every thread can read from the DB file and access all previously established sessions, but only one thread can write at a time. The current behaviour of gnutls is not to block to wait for the DB to be ready for writing, but continue the process normally (and do not save the parameters).

GNUTLS also provides callback functions such as: gnutls_db_set_remove_function() (see Section 1.10.37), gnutls_db_set_store_function() (see Section 1.10.38), gnutls_db_set_retrieve_function() (see Section 1.10.36) and gnutls_db_set_ptr() (see Section 1.10.39). These callback functions are required in order to use a session storage method, other than the default gdbm backend.

If an alternative backend is in use, it might be usefull to be able to check for expired sessions in order to remove them, and save space. This is what gnutls_db_clean() (see Section 1.10.44) does for the gdbm backend. *GNUTLS* provides the function gnutls_db_check_entry() (see Section 1.10.43) , which takes as input session data, and returns a negative value if the data are to be removed.

1.6 Transport Layer

GNUTLS can be used above any reliable transport layer. To do this you will only need to set up the gnutls_transport_set_push_func() (see Section 1.10.67) and gnutls_transport_set_pull_func() (see Section 1.10.66) functions. These functions will then be used by gnutls in order to send and receive data. The functions specified should return -1 on error and should set errno appropriately. *GNUTLS* supports EINTR and EAGAIN errno values. These values are usually used in non blocking IO and interrupted system calls. The corresponding values (GNUTLS_E_INTERRUPTED, GNUTLS_E_AGAIN) will be returned to the caller of the gnutls function. *GNUTLS* functions can be resumed (called again), if any of these values is returned.

By default, if none of the above functions are called, gnutls will use the berkeley sockets functions *recv()* and *send()*. In this case gnutls will use some hacks in order for *select()* to work, thus making easy to add *TLS* support to existing servers.

1.7 Error Handling

In *GNUTLS* most functions return an integer type as a result. In almost all cases a zero or a positive number means success, and a negative number indicates failure, or a situation that some action has to be taken. Thus negative error codes may be fatal or not.

Fatal errors terminate the connection immediately and further sends ard receives will be disallowed. An example of a fatal error code is GNUTLS_E_MAC_FAILED. Non-fatal errors may warn about something (ie a warning alert was received), or indicate the some action has to be taken. This is the case with the error code GNUTLS_E_REHANDSHAKE returned by gnutls_read() (see Section 1.10.13). This error code requires a full handshake to be performed again, or an alert to be sent. You can test if an error code is a fatal one by using the gnutls_error_is_fatal() (see Section 1.10.20).

If any non fatal errors, that require reaction, are to be returned by a function, these error codes will be documented in the function's reference.

1.8 Client Examples

This section contains examples of *TLS* and *SSL* clients, using *GNUTLS* .

1.8.1 Simple Client example with X.509 Authentication

Let's assume now that we want to create a client which communicates with servers using the X509 authentication schema. The following client is a very simple *TLS* client, it does not support session resuming nor any other fancy features.

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <gnutls.h>

#define MAX_BUF 1024
#define CRLFILE "crl.pem"
#define CAFILE "ca.pem"
#define SA struct sockaddr
#define MSG "GET / HTTP/1.0\r\n\r\n"

int main()
{
   const char *PORT = "443";
   const char *SERVER = "127.0.0.1";
   int err, ret;
   int sd, ii;
   struct sockaddr_in sa;
   GNUTLS_STATE state;
   char buffer[MAX_BUF + 1];
   GNUTLS_X509PKI_CLIENT_CREDENTIALS xcred;
   const int protocol_priority[] = { GNUTLS_TLS1, GNUTLS_SSL3, 0 };
   const int kx_priority[] = { GNUTLS_KX_X509PKI_RSA, 0 };
   const int cipher_priority[] = { GNUTLS_CIPHER_3DES_CBC, GNUTLS_CIPHER_ARCFOUR, 0};
   const int comp_priority[] = { GNUTLS_COMP_ZLIB, GNUTLS_COMP_NULL, 0 };
   const int mac_priority[] = { GNUTLS_MAC_SHA, GNUTLS_MAC_MD5, 0 };


   if (gnutls_global_init() < 0) {
      fprintf(stderr, "global state initialization error\n");
      exit(1);
   }
   /* X509 stuff */
   if (gnutls_x509pki_allocate_client_sc(&xcred, 0) < 0) {  /* no client private key */
      fprintf(stderr, "memory error\n");
```

```
      exit(1);
   }
   /* set's the trusted cas file
    */
   gnutls_x509pki_set_client_trust_file(xcred, CAFILE, CRLFILE);

   /* connects to server
    */
   sd = socket(AF_INET, SOCK_STREAM, 0);

   memset(&sa, '\0', sizeof(sa));
   sa.sin_family = AF_INET;
   sa.sin_port = htons(atoi(PORT));
   inet_pton(AF_INET, SERVER, &sa.sin_addr);

   err = connect(sd, (SA *) & sa, sizeof(sa));
   if (err < 0) {
      fprintf(stderr, "Connect error\n");
      exit(1);
   }
   /* Initialize TLS state
    */
   gnutls_init(&state, GNUTLS_CLIENT);

   /* allow both SSL3 and TLS1
    */
   gnutls_protocol_set_priority(state, protocol_priority);

   /* allow only ARCFOUR and 3DES ciphers
    * (3DES has the highest priority)
    */
   gnutls_cipher_set_priority(state, cipher_priority);

   /* only allow null compression
    */
   gnutls_compression_set_priority(state, comp_priority);

   /* use GNUTLS_KX_X509PKI_RSA
    */
   gnutls_kx_set_priority(state, kx_priority);

   /* allow the usage of both SHA and MD5
    */
   gnutls_mac_set_priority(state, mac_priority);
```

```
   /* put the x509 credentials to the current state
    */
   gnutls_cred_set(state, GNUTLS_X509PKI, xcred);


   gnutls_transport_set_ptr( state, sd);
   /* Perform the TLS handshake
    */
   ret = gnutls_handshake( state);

   if (ret < 0) {
      fprintf(stderr, "*** Handshake failed\n");
      gnutls_perror(ret);
      goto end;
   } else {
      printf("- Handshake was completed\n");
   }

   gnutls_write( state, MSG, strlen(MSG));

   ret = gnutls_read( state, buffer, MAX_BUF);
   if (gnutls_error_is_fatal(ret) == 1 || ret == 0) {
      if (ret == 0) {
         printf("- Peer has closed the GNUTLS connection\n");
         goto end;
      } else {
         fprintf(stderr, "*** Received corrupted data(%d) - server has terminated the connection abnormally\n",
                 ret);
         goto end;
      }
   } else {
      if (ret == GNUTLS_E_WARNING_ALERT_RECEIVED || ret == GNUTLS_E_FATAL_ALERT_RECEIVED)
         printf("* Received alert [%d]\n", gnutls_alert_get_last(state));
      if (ret == GNUTLS_E_REHANDSHAKE)
         printf("* Received HelloRequest message (server asked to rehandshake)\n");
         gnutls_alert_send_appropriate( state, ret); /* we don't want rehandshake */
   }

   if (ret > 0) {
      printf("- Received %d bytes: ", ret);
      for (ii = 0; ii < ret; ii++) {
         fputc(buffer[ii], stdout);
      }
      fputs("\n", stdout);
   }
   gnutls_bye( state, GNUTLS_SHUT_RDWR);
```

```
 end:

   shutdown(sd, SHUT_RDWR);     /* no more receptions */
   close(sd);

   gnutls_deinit(state);

   gnutls_x509pki_free_client_sc(xcred);

   gnutls_global_deinit();

   return 0;
}
```

1.8.2 Getting peer's information

The above example was the simplest form of a client, it didn't even check the result of the peer's certificate verification function (ie. if we have an authenticated connection). The following function does check the peer's X509 Certificate, and prints some information about the current state.

This function should be called after a successful gnutls_handshake() (see Section 1.10.18)

```
#define PRINTX(x,y) if (y[0]!=0) printf(" -   %s %s\n", x, y)
#define PRINT_DN(X) PRINTX( "CN:", X.common_name); \
        PRINTX( "OU:", X.organizational_unit_name); \
        PRINTX( "O:", X.organization); \
        PRINTX( "L:", X.locality_name); \
        PRINTX( "S:", X.state_or_province_name); \
        PRINTX( "C:", X.country); \
        PRINTX( "E:", X.email)

/* This function will print some details of the
 * given state.
 */
int print_info(GNUTLS_STATE state)
{
   const char *tmp;
   GNUTLS_CredType cred;
   gnutls_DN dn;
   const gnutls_datum *cert_list;
   GNUTLS_CertificateStatus status;
```

```
int cert_list_size = 0;
GNUTLS_KXAlgorithm kx;


/* print the key exchange's algorithm name
 */
kx = gnutls_kx_get_algo(state);
tmp = gnutls_kx_get_name(kx);
printf("- Key Exchange: %s\n", tmp);

cred = gnutls_auth_get_type(state);
switch (cred) {
case GNUTLS_ANON:
   printf("- Anonymous DH using prime of %d bits\n",
          gnutls_anon_client_get_dh_bits(state));
   break;
case GNUTLS_X509PKI:
   /* in case of X509 PKI
    */
   cert_list = gnutls_x509pki_client_get_peer_certificate_list(state, &cert_list_size);
   status = gnutls_x509pki_client_get_peer_certificate_status(state);

   switch (status) {
   case GNUTLS_CERT_NOT_TRUSTED:
      printf("- Peer's X509 Certificate was NOT verified\n");
      break;
   case GNUTLS_CERT_EXPIRED:
      printf("- Peer's X509 Certificate was verified but is expired\n");
      break;
   case GNUTLS_CERT_TRUSTED:
      printf("- Peer's X509 Certificate was verified\n");
      break;
   case GNUTLS_CERT_NONE:
      printf("- Peer did not send any X509 Certificate.\n");
      break;
   case GNUTLS_CERT_INVALID:
      printf("- Peer's X509 Certificate was invalid\n");
      break;
   }

   /* Check if we have been using ephemeral Diffie Hellman.
    */
   if (kx == GNUTLS_KX_X509PKI_DHE_RSA || kx == GNUTLS_KX_X509PKI_DHE_DSS) {
      printf("\n- Ephemeral DH using prime of %d bits\n",
             gnutls_x509pki_server_get_dh_bits(state));
   }
```

```
/* if the certificate list is available, then
 * print some information about it.
 */
if (cert_list_size > 0) {
   char digest[20];
   char serial[40];
   int digest_size = sizeof(digest), i;
   int serial_size = sizeof(serial);
   char printable[120];
   char *print;

   printf(" - Certificate info:\n");

   /* Print the fingerprint of the certificate
    */
   if (gnutls_fingerprint(GNUTLS_DIG_MD5, &cert_list[0], digest, &digest_size) >= 0) {
      print = printable;
      for (i = 0; i < digest_size; i++) {
         sprintf(print, "%.2x ", (unsigned char) digest[i]);
         print += 3;
      }
      printf(" - Certificate fingerprint: %s\n", printable);
   }

   /* Print the serial number of the certificate.
    */
   if (gnutls_x509pki_extract_certificate_serial(&cert_list[0], serial, &serial_size) >= 0) {
      print = printable;
      for (i = 0; i < serial_size; i++) {
         sprintf(print, "%.2x ", (unsigned char) serial[i]);
         print += 3;
      }
      printf(" - Certificate serial number: %s\n", printable);
   }

   /* Print the version of the X.509
    * certificate.
    */
   printf(" - Certificate version: #%d\n", gnutls_x509pki_extract_certificate_version(&cert_list[0]));

   gnutls_x509pki_extract_certificate_dn(&cert_list[0], &dn);
   PRINT_DN(dn);

   gnutls_x509pki_extract_certificate_issuer_dn(&cert_list[0], &dn);
   printf(" - Certificate Issuer's info:\n");
```

```
         PRINT_DN(dn);
      }
   }

   tmp = gnutls_protocol_get_name(gnutls_protocol_get_version(state));
   printf("- Version: %s\n", tmp);

   tmp = gnutls_compression_get_name(gnutls_compression_get_algo(state));
   printf("- Compression: %s\n", tmp);

   tmp = gnutls_cipher_get_name(gnutls_cipher_get_algo(state));
   printf("- Cipher: %s\n", tmp);

   tmp = gnutls_mac_get_name(gnutls_mac_get_algo(state));
   printf("- MAC: %s\n", tmp);

   return 0;
}
```

1.8.3 Client with Resume capability example

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <gnutls.h>

#define MAX_BUF 1024
#define CRLFILE "crl.pem"
#define CAFILE "ca.pem"
#define SA struct sockaddr
#define MSG "GET / HTTP/1.0\r\n\r\n"

const int protocol_priority[] = { GNUTLS_TLS1, GNUTLS_SSL3, 0 };
const int kx_priority[] = { GNUTLS_KX_X509PKI_RSA, GNUTLS_KX_X509PKI_DHE_RSA, 0 };
const int cipher_priority[] = { GNUTLS_CIPHER_3DES_CBC, GNUTLS_CIPHER_ARCFOUR, 0};
const int comp_priority[] = { GNUTLS_COMP_ZLIB, GNUTLS_COMP_NULL, 0 };
const int mac_priority[] = { GNUTLS_MAC_SHA, GNUTLS_MAC_MD5, 0 };

int main()
{
```

```
const char *PORT = "443";
const char *SERVER = "127.0.0.1";
int err, ret;
int sd, ii;
struct sockaddr_in sa;
GNUTLS_STATE state;
char buffer[MAX_BUF + 1];
GNUTLS_X509PKI_CLIENT_CREDENTIALS xcred;
/* variables used in session resuming */
int t;
char *session;
char *session_id;
int session_size;
int session_id_size;
char *tmp_session_id;
int tmp_session_id_size;

if (gnutls_global_init() < 0) {
   fprintf(stderr, "global state initialization error\n");
   exit(1);
}
/* X509 stuff */
if (gnutls_x509pki_allocate_client_sc(&xcred, 0) < 0) {  /* no client private key */
   fprintf(stderr, "memory error\n");
   exit(1);
}
gnutls_x509pki_set_client_trust_file(xcred, CAFILE, CRLFILE);

for (t = 0; t < 2; t++) {    /* connect 2 times to the server */

   sd = socket(AF_INET, SOCK_STREAM, 0);
   memset(&sa, '\0', sizeof(sa));
   sa.sin_family = AF_INET;
   sa.sin_port = htons(atoi(PORT));
   inet_pton(AF_INET, SERVER, &sa.sin_addr);

   err = connect(sd, (SA *) & sa, sizeof(sa));
   if (err < 0) {
      fprintf(stderr, "Connect error");
      exit(1);
   }
   gnutls_init(&state, GNUTLS_CLIENT);

   gnutls_protocol_set_priority(state, protocol_priority);
   gnutls_cipher_set_priority(state, cipher_priority);
   gnutls_compression_set_priority(state, comp_priority);
```

```
gnutls_kx_set_priority(state, kx_priority);
gnutls_mac_set_priority(state, mac_priority);

gnutls_cred_set(state, GNUTLS_X509PKI, xcred);

if (t > 0) { /* if this is not the first time we connect */
   gnutls_session_set_data(state, session, session_size);
   free(session);
}

gnutls_set_transport_ptr( state, sd);

/* Perform the TLS handshake
 */
ret = gnutls_handshake( state);

if (ret < 0) {
   fprintf(stderr, "*** Handshake failed\n");
   gnutls_perror(ret);
   goto end;
} else {
   printf("- Handshake was completed\n");
}

if (t == 0) { /* the first time we connect */
   /* get the session data size */
   gnutls_session_get_data(state, NULL, &session_size);
   session = malloc(session_size);

   /* put session data to the session variable */
   gnutls_session_get_data(state, session, &session_size);

   /* keep the current session ID. This is only needed
    * in order to check if the server actually resumed this
    * connection.
    */
   gnutls_session_get_id(state, NULL, &session_id_size);
   session_id = malloc(session_id_size);
   gnutls_session_get_id(state, session_id, &session_id_size);

} else { /* the second time we connect */

   /* check if we actually resumed the previous session */
   gnutls_session_get_id(state, NULL, &tmp_session_id_size);
   tmp_session_id = malloc(tmp_session_id_size);
   gnutls_session_get_id(state, tmp_session_id, &tmp_session_id_size);
```

```
      if (memcmp(tmp_session_id, session_id, session_id_size) == 0) {
         printf("- Previous session was resumed\n");
      } else {
         fprintf(stderr, "*** Previous session was NOT resumed\n");
      }
      free(tmp_session_id);
      free(session_id);
   }

   /* This function was defined in a previous example
    */
   print_info(state);

   gnutls_write( state, MSG, strlen(MSG));

   ret = gnutls_read( state, buffer, MAX_BUF);
   if (gnutls_error_is_fatal(ret) == 1 || ret == 0) {
      if (ret == 0) {
         printf("- Peer has closed the GNUTLS connection\n");
         goto end;
      } else {
         fprintf(stderr, "*** Received corrupted data(%d) - server has terminated the connection abnormally\n",
                 ret);
         goto end;
      }
   } else {
      if (ret == GNUTLS_E_WARNING_ALERT_RECEIVED || ret == GNUTLS_E_FATAL_ALERT_RECEIVED)
         printf("* Received alert [%d]\n", gnutls_alert_get_last(state));
      if (ret == GNUTLS_E_REHANDSHAKE) {
         printf("* Received HelloRequest message (server asked to rehandshake)\n");
         gnutls_alert_send_appropriate( state, ret); /* we don't want rehandshake */
      }
   }

   if (ret > 0) {
      printf("- Received %d bytes: ", ret);
      for (ii = 0; ii < ret; ii++) {
         fputc(buffer[ii], stdout);
      }
      fputs("\n", stdout);
   }
   gnutls_bye( state, GNUTLS_SHUT_RDWR);

 end:
```

```
      shutdown(sd, SHUT_RDWR);  /* no more receptions */
      close(sd);

      gnutls_deinit(state);

   }  /* for() */

   gnutls_x509pki_free_client_sc(xcred);

   gnutls_global_deinit();

   return 0;
}
```

1.8.4 Simple Client example with SRP Authentication

Although **SRP** is not part of the *TLS* standard, *GNUTLS* implements *David Taylor's*[15] proposal for using the SRP algorithm within the *TLS* handshake protocol. The following client is a very simple SRP-TLS client which connects to a server and authenticates using *username* and *password*.

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <gnutls.h>

#define MAX_BUF 1024
#define USERNAME "user"
#define PASSWORD "pass"
#define SA struct sockaddr
#define MSG "GET / HTTP/1.0\r\n\r\n"

const int protocol_priority[] = { GNUTLS_TLS1, GNUTLS_SSL3, 0 };
const int kx_priority[] = { GNUTLS_KX_SRP, 0 };
const int cipher_priority[] = { GNUTLS_CIPHER_3DES_CBC, GNUTLS_CIPHER_ARCFOUR, 0};
const int comp_priority[] = { GNUTLS_COMP_NULL, 0 };
const int mac_priority[] = { GNUTLS_MAC_SHA, GNUTLS_MAC_MD5, 0 };

int main()
```

[15] Work in progress.

```
{
   const char *PORT = "443";
   const char *SERVER = "127.0.0.1";
   int err, ret;
   int sd, ii;
   struct sockaddr_in sa;
   GNUTLS_STATE state;
   char buffer[MAX_BUF + 1];
   GNUTLS_SRP_CLIENT_CREDENTIALS xcred;

   if (gnutls_global_init() < 0) {
      fprintf(stderr, "global state initialization error\n");
      exit(1);
   }
   if (gnutls_srp_allocate_client_sc(&xcred) < 0) {
      fprintf(stderr, "memory error\n");
      exit(1);
   }
   gnutls_srp_set_client_cred(xcred, USERNAME, PASSWORD);

   /* connects to server
    */
   sd = socket(AF_INET, SOCK_STREAM, 0);

   memset(&sa, '\0', sizeof(sa));
   sa.sin_family = AF_INET;
   sa.sin_port = htons(atoi(PORT));
   inet_pton(AF_INET, SERVER, &sa.sin_addr);

   err = connect(sd, (SA *) & sa, sizeof(sa));
   if (err < 0) {
      fprintf(stderr, "Connect error\n");
      exit(1);
   }
   /* Initialize TLS state
    */
   gnutls_init(&state, GNUTLS_CLIENT);

   /* allow both SSL3 and TLS1
    */
   gnutls_protocol_set_priority(state, protocol_priority);

   /* allow only ARCFOUR and 3DES ciphers
    * (3DES has the highest priority)
    */
    gnutls_cipher_set_priority(state, cipher_priority);
```

```
   /* only allow null compression
    */
   gnutls_compression_set_priority(state, comp_priority);

   /* use GNUTLS_KX_SRP
    */
   gnutls_kx_set_priority(state, kx_priority);

   /* allow the usage of both SHA and MD5
    */
   gnutls_mac_set_priority(state, mac_priority);


   /* put the SRP credentials to the current state
    */
   gnutls_cred_set(state, GNUTLS_SRP, xcred);

   gnutls_transport_set_ptr( state, sd);

   /* Perform the TLS handshake
    */
   ret = gnutls_handshake( state);

   if (ret < 0) {
      fprintf(stderr, "*** Handshake failed\n");
      gnutls_perror(ret);
      goto end;
   } else {
      printf("- Handshake was completed\n");
   }

   gnutls_write( state, MSG, strlen(MSG));

   ret = gnutls_read( state, buffer, MAX_BUF);
   if (gnutls_error_is_fatal(ret) == 1 || ret == 0) {
      if (ret == 0) {
         printf("- Peer has closed the GNUTLS connection\n");
         goto end;
      } else {
         fprintf(stderr, "*** Received corrupted data(%d) - server has terminated the connection abnormally\n",
                 ret);
         goto end;
      }
   } else {
      if (ret == GNUTLS_E_WARNING_ALERT_RECEIVED || ret == GNUTLS_E_FATAL_ALERT_RECEIVED)
```

```
         printf("* Received alert [%d]\n", gnutls_alert_get_last(state));
      if (ret == GNUTLS_E_REHANDSHAKE)
         printf("* Received HelloRequest message (server asked to rehandshake)\n");
   }

   if (ret > 0) {
      printf("- Received %d bytes: ", ret);
      for (ii = 0; ii < ret; ii++) {
         fputc(buffer[ii], stdout);
      }
      fputs("\n", stdout);
   }
   gnutls_bye( state, 0);

 end:

   shutdown(sd, SHUT_RDWR);     /* no more receptions */
   close(sd);

   gnutls_deinit(state);

   gnutls_srp_free_client_sc(xcred);

   gnutls_global_deinit();

   return 0;
}
```

1.9 Server Examples

This section contains examples of *TLS* and *SSL* servers, using *GNUTLS* .

1.9.1 Echo Server with X.509 and SRP authentication

The following example is a server which supports both **SRP** and **X509** authentication. This server also supports *session resuming.*

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
```

```
#include <arpa/inet.h>
#include <string.h>
#include <unistd.h>
#include <gnutls.h>

#define KEYFILE "key.pem"
#define CERTFILE "cert.pem"
#define CAFILE "ca.pem"
#define CRLFILE NULL

#define SRP_PASSWD "tpasswd"
#define SRP_PASSWD_CONF "tpasswd.conf"


/* This is a sample TCP echo server.
 */


#define SA struct sockaddr
#define ERR(err,s) if(err==-1) {perror(s);return(1);}
#define MAX_BUF 1024
#define PORT 5556               /* listen to 5556 port */

/* These are global */
GNUTLS_SRP_SERVER_CREDENTIALS srp_cred;
GNUTLS_X509PKI_SERVER_CREDENTIALS x509_cred;

GNUTLS_STATE initialize_state()
{
   GNUTLS_STATE state;
   int ret;
   const int protocol_priority[] = { GNUTLS_TLS1, GNUTLS_SSL3, 0 };
   const int kx_priority[] = { GNUTLS_KX_X509PKI_RSA, GNUTLS_KX_X509PKI_DHE_RSA, GNUTLS_KX_SRP, 0 };
   const int cipher_priority[] = { GNUTLS_CIPHER_RIJNDAEL_CBC, GNUTLS_CIPHER_3DES_CBC, 0};
   const int comp_priority[] = { GNUTLS_COMP_ZLIB, GNUTLS_COMP_NULL, 0 };
   const int mac_priority[] = { GNUTLS_MAC_SHA, GNUTLS_MAC_MD5, 0 };

   gnutls_init(&state, GNUTLS_SERVER);

   /* in order to support session resuming:
    */
   if ((ret = gnutls_db_set_name(state, "gnutls-rsm.db")) < 0)
      fprintf(stderr, "*** DB error (%d)\n\n", ret);

   gnutls_protocol_set_priority(state, protocol_priority);
   gnutls_cipher_set_priority(state, cipher_priority);
```

```
   gnutls_compression_set_priority(state, comp_priority);
   gnutls_kx_set_priority(state, kx_priority);
   gnutls_mac_set_priority(state, mac_priority);

   gnutls_cred_set(state, GNUTLS_SRP, srp_cred);
   gnutls_cred_set(state, GNUTLS_X509PKI, x509_cred);

   /* request client certificate if any.
    */
   gnutls_x509pki_server_set_cert_request( state, GNUTLS_CERT_REQUEST);

   return state;
}

void print_info(GNUTLS_STATE state)
{
   const char *tmp;
   unsigned char sesid[32];
   int sesid_size, i;

   /* print session_id specific data */
   gnutls_session_get_id(state, sesid, &sesid_size);
   printf("\n- Session ID: ");
   for (i = 0; i < sesid_size; i++)
      printf("%.2X", sesid[i]);
   printf("\n");

   /* print srp specific data */
   if (gnutls_get_auth_type(state) == GNUTLS_SRP) {
         printf("\n- User '%s' connected\n",
                gnutls_srp_server_get_username( state));
   }

   /* print state information */
   tmp = gnutls_protocol_get_name(gnutls_protocol_get_version(state));
   printf("- Version: %s\n", tmp);

   tmp = gnutls_kx_get_name(gnutls_kx_get_algo(state));
   printf("- Key Exchange: %s\n", tmp);

   tmp =
       gnutls_compression_get_name
       (gnutls_compression_get_algo(state));
   printf("- Compression: %s\n", tmp);

   tmp = gnutls_cipher_get_name(gnutls_cipher_get_algo(state));
```

```
   printf("- Cipher: %s\n", tmp);

   tmp = gnutls_mac_get_name(gnutls_mac_get_algo(state));
   printf("- MAC: %s\n", tmp);

}



int main()
{
   int err, listen_sd, i;
   int sd, ret;
   struct sockaddr_in sa_serv;
   struct sockaddr_in sa_cli;
   int client_len;
   char topbuf[512];
   GNUTLS_STATE state;
   char buffer[MAX_BUF + 1];
   int optval = 1;
   int http = 0;
   char name[256];

   strcpy(name, "Echo Server");

   /* this must be called once in the program
    */
   if (gnutls_global_init() < 0) {
      fprintf(stderr, "global state initialization error\n");
      exit(1);
   }
   if (gnutls_x509pki_allocate_server_sc(&x509_cred, 1) < 0) {
      fprintf(stderr, "memory error\n");
      exit(1);
   }
   if (gnutls_x509pki_set_server_trust_file(x509_cred, CAFILE, CRLFILE) < 0) {
      fprintf(stderr, "X509 PARSE ERROR\nDid you have ca.pem?\n");
      exit(1);
   }
   if (gnutls_x509pki_set_server_key_file(x509_cred, CERTFILE, KEYFILE) < 0) {
      fprintf(stderr, "X509 PARSE ERROR\nDid you have key.pem and cert.pem?\n");
      exit(1);
   }
   /* SRP_PASSWD a password file (created with the included crypt utility)
    * Read README.crypt prior to using SRP.
    */
```

```
gnutls_srp_allocateserver_sc(&srp_cred);
gnutls_srp_set_server_cred(srp_cred, SRP_PASSWD, SRP_PASSWD_CONF);


/* Socket operations
 */
listen_sd = socket(AF_INET, SOCK_STREAM, 0);
ERR(listen_sd, "socket");

memset(&sa_serv, '\0', sizeof(sa_serv));
sa_serv.sin_family = AF_INET;
sa_serv.sin_addr.s_addr = INADDR_ANY;
sa_serv.sin_port = htons(PORT);  /* Server Port number */

setsockopt(listen_sd, SOL_SOCKET, SO_REUSEADDR, &optval, sizeof(int));

err = bind(listen_sd, (SA *) & sa_serv, sizeof(sa_serv));
ERR(err, "bind");
err = listen(listen_sd, 1024);
ERR(err, "listen");

printf("%s ready. Listening to port '%d'.\n\n", name, PORT);

client_len = sizeof(sa_cli);
for (;;) {
   state = initialize_state();

   sd = accept(listen_sd, (SA *) & sa_cli, &client_len);

   printf("- connection from %s, port %d\n",
          inet_ntop(AF_INET, &sa_cli.sin_addr, topbuf,
                    sizeof(topbuf)), ntohs(sa_cli.sin_port));

   gnutls_transport_set_ptr( state, sd);
   ret = gnutls_handshake( state);
   if (ret < 0) {
      close(sd);
      gnutls_deinit(state);
      fprintf(stderr, "*** Handshake has failed (%s)\n\n",
              gnutls_strerror(ret));
      continue;
   }
   printf("- Handshake was completed\n");

   print_info(state);
```

```
      i = 0;
      for (;;) {
         bzero(buffer, MAX_BUF + 1);
         ret = gnutls_read( state, buffer, MAX_BUF);

         if (gnutls_error_is_fatal(ret) == 1 || ret == 0) {
            if (ret == 0) {
               printf
                   ("\n- Peer has closed the GNUTLS connection\n");
               break;
            } else {
               fprintf(stderr,
                       "\n*** Received corrupted data(%d). Closing the connection.\n\n",
                       ret);
               break;
            }

         }
         if (ret > 0) {
            /* echo data back to the client
             */
            gnutls_write( state, buffer,
                         strlen(buffer));
         }
         if (ret == GNUTLS_E_WARNING_ALERT_RECEIVED || ret == GNUTLS_E_FATAL_ALERT_RECEIVED) {
            ret = gnutls_alert_get_last(state);
            printf("* Received alert '%d'.\n", ret);
         }
      }
      printf("\n");
      gnutls_bye( state, 1); /* do not wait for
                              * the peer to close the connection.
                              */

      close(sd);
      gnutls_deinit(state);

   }
   close(listen_sd);

   gnutls_x509pki_free_server_sc(x509_cred);
   gnutls_srp_free_server_sc(srp_cred);

   gnutls_global_deinit();

   return 0;
```

```
}
```

1.10 Function Reference

1.10.1 gnutls_protocol_get_version

GNUTLS_Version **gnutls_protocol_get_version** (*GNUTLS_STATE* **state**)

Arguments

- *GNUTLS_STATE* **state**: is a *GNUTLS_*STATE structure.

Description

Returns the version of the currently used protocol.

1.10.2 gnutls_transport_set_lowat

void **gnutls_transport_set_lowat** (*GNUTLS_STATE* **state** , *int* **num**)

Arguments

- *GNUTLS_STATE* **state**: is a *GNUTLS_*STATE structure.
- *int* **num**: is the low water value.

Description

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 usefull if using berkeley style sockets. Otherwise it must be called and set lowat to zero.

1.10.3 gnutls_transport_set_ptr

void **gnutls_transport_set_ptr** (*GNUTLS_STATE* **state** , *GNUTLS_SOCKET_PTR* **ptr**)

Arguments

- *GNUTLS_STATE* **state**: is a *GNUTLS_*STATE structure.
- *GNUTLS_SOCKET_PTR* **ptr**: is the value.

Description

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.

1.10.4 gnutls_transport_get_ptr

GNUTLS_SOCKET_PTR **gnutls_transport_get_ptr** (*GNUTLS_STATE* **state**)

Arguments

- *GNUTLS_STATE* **state**: is a *GNUTLS*_STATE structure.

Description

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

1.10.5 gnutls_init

int **gnutls_init** (*GNUTLS_STATE ** **state** , *ConnectionEnd* **con_end**)

Arguments

- *GNUTLS_STATE ** **state**: is a pointer to a *GNUTLS*_STATE structure.
- *ConnectionEnd* **con_end**: is used to indicate if this state is to be used for server or client. Can be one of GNUTLS_CLIENT and GNUTLS_SERVER.

Description

This function initializes the current state to null. Every state 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.

1.10.6 gnutls_deinit

void **gnutls_deinit** (*GNUTLS_STATE* **state**)

Arguments

- *GNUTLS_STATE* **state**: is a *GNUTLS*_STATE structure.

Description

This function clears all buffers associated with the *state*.

1.10.7 gnutls_bye

int **gnutls_bye** (*GNUTLS_STATE* **state** , *CloseRequest* **how**)

Arguments

- *GNUTLS_STATE* **state**: is a *GNUTLS*_STATE structure.
- *CloseRequest* **how**: is an integer

Description

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.

This function may also return GNUTLS_E_AGAIN, or GNUTLS_E_INTERRUPTED.

1.10.8 gnutls_cipher_get_algo

BulkCipherAlgorithm **gnutls_cipher_get_algo** (*GNUTLS_STATE* **state**)

Arguments

- *GNUTLS_STATE* **state**: is a *GNUTLS_*STATE structure.

Description

Returns the currently used cipher.

1.10.9 gnutls_kx_get_algo

KXAlgorithm **gnutls_kx_get_algo** (*GNUTLS_STATE* **state**)

Arguments

- *GNUTLS_STATE* **state**: is a *GNUTLS_*STATE structure.

Description

Returns the key exchange algorithm used in the last handshake.

1.10.10 gnutls_mac_get_algo

MACAlgorithm **gnutls_mac_get_algo** (*GNUTLS_STATE* **state**)

Arguments

- *GNUTLS_STATE* **state**: is a *GNUTLS_*STATE structure.

Description

Returns the currently used mac algorithm.

1.10.11 gnutls_compression_get_algo

CompressionMethod **gnutls_compression_get_algo** (*GNUTLS_STATE* **state**)

Arguments

- *GNUTLS_STATE* **state**: is a *GNUTLS_*STATE structure.

Description

Returns the currently used compression method.

1.10.12 gnutls_write

ssize_t **gnutls_write** (*GNUTLS_STATE* **state** , *const void ** **data** , *size_t* **sizeofdata**)

Arguments

- *GNUTLS_STATE* **state**: is a *GNUTLS_*STATE structure.
- *const void ** **data**: contains the data to send
- *size_t* **sizeofdata**: is the length of the data

Description

This function has the similar semantics to write(). The only difference is that is accepts a GNUTLS state, and uses different error codes.

If the EINTR is returned by the internal push function (write()) 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. Otherwise the write operation will be corrupted and the connection will be terminated.

Returns the number of bytes sent, or a negative error code.

1.10.13 gnutls_read

ssize_t **gnutls_read** (*GNUTLS_STATE* **state** , *void ** **data** , *size_t* **sizeofdata**)

Arguments

- *GNUTLS_STATE* **state**: is a *GNUTLS_*STATE structure.
- *void ** **data**: contains the data to send
- *size_t* **sizeofdata**: is the length of the data

Description

This function has the similar semantics to read(). The only difference is that is accepts a GNUTLS state. Also returns the number of bytes received, zero on EOF, but a negative error code in case of an error.

If this function returns GNUTLS_E_REHANDSHAKE, then you must either send an alert containing NO_RENEGOTIATION, or perform a handshake. (only a client may receive this message)

1.10.14 gnutls_record_get_max_size

size_t **gnutls_record_get_max_size** (*GNUTLS_STATE* **state**)

Arguments

- *GNUTLS_STATE* **state**: is a *GNUTLS_*STATE structure.

Description

This function returns the maximum record size in this connection. The maximum record size is negotiated by the client after the first handshake message.

1.10.15 gnutls_record_set_max_size

size_t **gnutls_record_set_max_size** (*GNUTLS_STATE* **state** , *size_t* **size**)

Arguments

- *GNUTLS_STATE* **state**: is a *GNUTLS_*STATE structure.
- *size_t* **size**: is the new size

Description

This function sets the maximum record 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 2^9, 2^{10}, 2^{11} and 2^{12}. Returns 0 on success. The requested record size does not get in effect immediately. It will be used after a successful handshake.

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

1.10.16 gnutls_check_pending

int **gnutls_check_pending** (*GNUTLS_STATE* **state**)

Arguments

- *GNUTLS_STATE* **state**: is a *GNUTLS_*STATE structure.

Description

This function checks if there are any data to receive in the gnutls buffers. Returns the size of that data or 0. Notice that you may also use select() to check for data in the TCP connection, instead of this function. (gnutls leaves some data in the tcp buffer in order for select to work).

1.10.17 gnutls_rehandshake

int **gnutls_rehandshake** (*GNUTLS_STATE* **state**)

Arguments

- *GNUTLS_STATE* **state**: is a a *GNUTLS*_STATE structure.

Description

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.

If the client does not wish to renegotiate parameters he will reply with an alert message, thus the return code will be GNUTLS_E_WARNING_ALERT_RECEIVED and the alert will be GNUTLS_A_NO_RENEGOTIATION.

1.10.18 gnutls_handshake

int **gnutls_handshake** (*GNUTLS_STATE* **state**)

Arguments

- *GNUTLS_STATE* **state**: is a a *GNUTLS*_STATE structure.

Description

This function does the handshake of the TLS/SSL protocol, and initializes the TLS connection. Here the identity of the peer is checked automatically. This function will fail if any problem is encountered, and will return a negative error code. In case of a client, if it has been asked to resume a session, but the server didn't, then a full handshake will be performed.

This function may also return the non-fatal errors GNUTLS_E_AGAIN, or GNUTLS_E_INTERRUPTED. In that case you may resume the handshake (call this function again, until it returns ok)

1.10.19 gnutls_handshake_set_max_data_buffer_size

void **gnutls_handshake_set_max_data_buffer_size** (*GNUTLS_STATE* **state** , *int* **max**)

Arguments

- *GNUTLS_STATE* **state**: is a a *GNUTLS_*STATE structure.

- *int* **max**: is the maximum number.

Description

This function will set the maximum size of the handshake message sequence. Since the handshake messages are kept into memory until the handshake is successful this function allows you to set the maximum number of bytes that will be kept. The default value is 128kb which is large enough. Set this to 0 if you do not want to set an upper limit.

1.10.20 gnutls_error_is_fatal

int **gnutls_error_is_fatal** (*int* **error**)

Arguments

- *int* **error**: is an error returned by a gnutls function. Error should be a negative value.

Description

If a function returns a negative value you may feed that value to this function to see if it is fatal. Returns 1 for a fatal error 0 otherwise. However you may want to check the error code manualy, since some non-fatal errors to the protocol may be fatal for you (your program).

1.10.21 gnutls_perror

void **gnutls_perror** (*int* **error**)

Arguments

- *int* **error**: is an error returned by a gnutls function. Error is always a negative value.

Description

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

1.10.22 gnutls_strerror

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

Arguments

- *int* **error**: is an error returned by a gnutls function. Error is always a negative value.

Description

This function is similar to strerror(). The only difference is that it accepts an error (number) returned by a gnutls function.

1.10.23 gnutls_mac_get_name

*const char ** **gnutls_mac_get_name** (*MACAlgorithm* **algorithm**)

Arguments

- *MACAlgorithm* **algorithm**: is a MAC algorithm

Description

Returns a string that contains the name of the specified MAC algorithm.

1.10.24 gnutls_compression_get_name

*const char ** **gnutls_compression_get_name** (*CompressionMethod* **algorithm**)

Arguments

- *CompressionMethod* **algorithm**: is a Compression algorithm

Description

Returns a pointer to a string that contains the name of the specified compression algorithm.

1.10.25 gnutls_cipher_get_name

*const char ** **gnutls_cipher_get_name** (*BulkCipherAlgorithm* **algorithm**)

Arguments

- *BulkCipherAlgorithm* **algorithm**: is an encryption algorithm

Description

Returns a pointer to a string that contains the name of the specified cipher.

1.10.26 gnutls_kx_get_name

*const char ** **gnutls_kx_get_name** (*KXAlgorithm* **algorithm**)

Arguments

- *KXAlgorithm* **algorithm**: is a key exchange algorithm

Description

Returns a pointer to a string that contains the name of the specified key exchange algorithm.

1.10.27 gnutls_protocol_get_name

*const char ** **gnutls_protocol_get_name** (*GNUTLS_Version* **version**)

Arguments

- *GNUTLS_Version* **version**: is a (gnutls) version number

Description

Returns a string that contains the name of the specified TLS version.

1.10.28 gnutls_cipher_set_priority

int **gnutls_cipher_set_priority** (*GNUTLS_STATE* **state** , *GNUTLS_LIST* **list**)

Arguments

- *GNUTLS_STATE* **state**: is a *GNUTLS_*STATE structure.
- *GNUTLS_LIST* **list**: is a 0 terminated list of BulkCipherAlgorithm elements.

Description

Sets the priority on the ciphers supported by gnutls. Priority is higher for ciphers specified before others. After specifying the ciphers you want, you should add 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.

1.10.29 gnutls_kx_set_priority

int **gnutls_kx_set_priority** (*GNUTLS_STATE* **state** , *GNUTLS_LIST* **list**)

Arguments

- *GNUTLS_STATE* **state**: is a *GNUTLS_*STATE structure.

- *GNUTLS_LIST* **list**: is a 0 terminated list of KXAlgorithm elements.

Description

Sets the priority on the key exchange algorithms supported by gnutls. Priority is higher for algorithms specified before others. After specifying the algorithms you want, you should add 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.

1.10.30 gnutls_mac_set_priority

int **gnutls_mac_set_priority** (*GNUTLS_STATE* **state** , *GNUTLS_LIST* **list**)

Arguments

- *GNUTLS_STATE* **state**: is a *GNUTLS_*STATE structure.
- *GNUTLS_LIST* **list**: is a 0 terminated list of MACAlgorithm elements.

Description

Sets the priority on the mac algorithms supported by gnutls. Priority is higher for algorithms specified before others. After specifying the algorithms you want, you should add 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.

1.10.31 gnutls_compression_set_priority

int **gnutls_compression_set_priority** (*GNUTLS_STATE* **state** , *GNUTLS_LIST* **list**)

Arguments

- *GNUTLS_STATE* **state**: is a *GNUTLS_*STATE structure.
- *GNUTLS_LIST* **list**: is a 0 terminated list of CompressionMethod elements.

Description

Sets the priority on the compression algorithms supported by gnutls. Priority is higher for algorithms specified before others. After specifying the algorithms you want, you should add 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.

1.10.32 gnutls_protocol_set_priority

int **gnutls_protocol_set_priority** (*GNUTLS_STATE* **state** , *GNUTLS_LIST* **list**)

Arguments

- *GNUTLS_STATE* **state**: is a *GNUTLS_*STATE structure.
- *GNUTLS_LIST* **list**: is a 0 terminated list of GNUTLS_Version elements.

Description

Sets the priority on the protocol versions supported by gnutls. Priority is higher for protocols specified before others. After specifying the protocols you want, you should add 0. Note that the priority is set on the client. The server does not use the protocols's priority except for disabling protocols that were not specified.

1.10.33 gnutls_session_get_data

int **gnutls_session_get_data** (*GNUTLS_STATE* **state** , *opaque** **session** , *int ** **session_size**)

Arguments

- *GNUTLS_STATE* **state**: is a *GNUTLS_*STATE structure.
- *opaque** **session**: is a pointer to space to hold the session.
- *int ** **session_size**: is the session's size, or it will be set by the function.

Description

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 succesful one.

1.10.34 gnutls_session_get_id

int **gnutls_session_get_id** (*GNUTLS_STATE* **state** , *void** **session** , *int ** **session_size**)

Arguments

- *GNUTLS_STATE* **state**: is a *GNUTLS_*STATE structure.
- *void** **session**: is a pointer to space to hold the session id.

- *int ** **session_size**: is the session id's size, or it will be set by the function.

Description

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 session id should not be more than 32 bytes.

1.10.35 gnutls_session_set_data

int **gnutls_session_set_data** (*GNUTLS_STATE* **state** , *opaque** **session** , *int* **session_size**)

Arguments

- *GNUTLS_STATE* **state**: is a *GNUTLS_*STATE structure.
- *opaque** **session**: is a pointer to space to hold the session.
- *int* **session_size**: is the session's size

Description

Sets all session parameters - in order to support resuming session 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.

1.10.36 gnutls_db_set_retrieve_function

void **gnutls_db_set_retrieve_function** (*GNUTLS_STATE* **state** , *DB_RETR_FUNC* **retr_func**)

Arguments

- *GNUTLS_STATE* **state**: is a *GNUTLS_*STATE structure.
- *DB_RETR_FUNC* **retr_func**: is the function.

Description

Sets the function that will be used to retrieve data from the resumed sessions database. This function must return a gnutls_datum containing the data on success, or a gnutls_datum containing null and 0 on failure. This function should only be used if you do not plan to use the included gdbm backend.

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

1.10.37 gnutls_db_set_remove_function

void **gnutls_db_set_remove_function** (*GNUTLS_STATE* **state** , *DB_REMOVE_FUNC* **rem_func**)

Arguments

- *GNUTLS_STATE* **state**: is a *GNUTLS_*STATE structure.
- *DB_REMOVE_FUNC* **rem_func**: is the function.

Description

Sets the function that will be used to remove data from the resumed sessions database. This function must return 0 on success. This function should only be used if you do not plan to use the included gdbm backend.

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

1.10.38 gnutls_db_set_store_function

void **gnutls_db_set_store_function** (*GNUTLS_STATE* **state** , *DB_STORE_FUNC* **store_func**)

Arguments

- *GNUTLS_STATE* **state**: is a *GNUTLS_*STATE structure.
- *DB_STORE_FUNC* **store_func**: is the function

Description

Sets the function that will be used to store data from the resumed sessions database. This function must remove 0 on success. This function should only be used if you do not plan to use the included gdbm backend.

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

1.10.39 gnutls_db_set_ptr

void **gnutls_db_set_ptr** (*GNUTLS_STATE* **state** , *void** **ptr**)

Arguments

- *GNUTLS_STATE* **state**: is a *GNUTLS_*STATE structure.
- *void** **ptr**: is the pointer

Description

Sets the pointer that will be sent to db store, retrieve and delete functions, as the first argument. Should only be called if not using the gdbm backend.

1.10.40 gnutls_db_get_ptr

*void** **gnutls_db_get_ptr** (*GNUTLS_STATE* **state**)
Arguments

- *GNUTLS_STATE* **state**: is a *GNUTLS_*STATE structure.

Description
Returns the pointer that will be sent to db store, retrieve and delete functions, as the first argument. Should only be used if not using the default (gdbm) backend.

1.10.41 gnutls_db_set_cache_expiration

void **gnutls_db_set_cache_expiration** (*GNUTLS_STATE* **state** , *int* **seconds**)
Arguments

- *GNUTLS_STATE* **state**: is a *GNUTLS_*STATE structure.
- *int* **seconds**: is the number of seconds.

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

1.10.42 gnutls_db_set_name

int **gnutls_db_set_name** (*GNUTLS_STATE* **state** , *char** **filename**)
Arguments

- *GNUTLS_STATE* **state**: is a *GNUTLS_*STATE structure.
- *char** **filename**: is the filename for the database

Description
Sets the name of the (gdbm) database to be used to keep the sessions to be resumed. This function also creates the database - if it does not exist - and opens it for reading. You should not call this function if using an other backend than gdbm (ie. called function gnutls_db_set_store_func() etc.)

1.10.43 gnutls_db_check_entry

int **gnutls_db_check_entry** (*GNUTLS_STATE* **state** , *gnutls_datum* **session_entry**)
Arguments

- *GNUTLS_STATE* **state**: is a *GNUTLS_*STATE structure.
- *gnutls_datum* **session_entry**: is the session data (not key)

Description

This function should only be used if not using the gdbm backend. This function returns GNUTLS_E_EXPIRED, if the database entry has expired or 0 otherwise. This function is to be used when you want to clear unnesessary session which occupy space in your backend.

1.10.44 gnutls_db_clean

int **gnutls_db_clean** (*GNUTLS_STATE* **state**)

Arguments

- *GNUTLS_STATE* **state**: is a *GNUTLS_*STATE structure.

Description

This function Deletes all expired records in the resumed sessions' database. This database may become huge if this function is not called. This function is also quite expensive. This function should only be called if using the gdbm backend.

1.10.45 gnutls_b64_encode_fmt

int **gnutls_b64_encode_fmt** (*const char** **msg** , *const gnutls_datum ** **data** , *char** **result** , *int** **result_size**)

Arguments

- *const char** **msg**: is a message to be put in the header
- *const gnutls_datum ** **data**: contain the raw data
- *char** **result**: the place where base64 data will be copied
- *int** **result_size**: holds the size of the result

Description

This function will convert the given data to printable data, using the base64 encoding. This is the encoding used in PEM messages.

1.10.46 gnutls_b64_decode_fmt

int **gnutls_b64_decode_fmt** (*const gnutls_datum ** **b64_data** , *char** **result** , *int** **result_size**)

Arguments

- *const gnutls_datum ** **b64_data**: contain the encoded data
- *char** **result**: the place where decoded data will be copied
- *int** **result_size**: holds the size of the result

Description

This function will decode the given encoded data.

1.10.47 gnutls_cred_set

int **gnutls_cred_set** (*GNUTLS_STATE* **state** , *CredType* **type** , *void** **cred**)

Arguments

- *GNUTLS_STATE* **state**: is a *GNUTLS_*STATE structure.
- *CredType* **type**: is the type of the credentials
- *void** **cred**: is a pointer to a structure.

Description

Sets the needed credentials for the specified type. Eg username, password - or public and private keys etc. The (void* cred) parameter is a structure that depends on the specified type and on the current state (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_ANON cred should be ANON_CLIENT_CREDENTIALS in case of a client. In case of a server it should be ANON_SERVER_CREDENTIALS.

For GNUTLS_SRP cred should be SRP_CLIENT_CREDENTIALS in case of a client, and SRP_SERVER_CREDENTIALS, in case of a server.

For GNUTLS_X509PKI cred should be X509PKI_CLIENT_CREDENTIALS in case of a client, and X509PKI_SERVER_CREDENTIALS, in case of a server.

1.10.48 gnutls_auth_get_type

CredType **gnutls_auth_get_type** (*GNUTLS_STATE* **state**)

Arguments

- *GNUTLS_STATE* **state**: is a *GNUTLS_*STATE structure.

Description

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 X509PKI ciphersuites (key exchange algorithms: KX_RSA, KX_DHE_RSA), the same function are to be used to access the authentication data.

1.10.49 gnutls_srp_free_client_sc

void **gnutls_srp_free_client_sc** (*GNUTLS_SRP_CLIENT_CREDENTIALS* **sc**)

Arguments

- *GNUTLS_SRP_CLIENT_CREDENTIALS* **sc**: is an *GNUTLS_*SRP_CLIENT_CREDENTIALS structure.

Description

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

1.10.50 gnutls_srp_allocate_client_sc

int **gnutls_srp_allocate_client_sc** (*GNUTLS_SRP_CLIENT_CREDENTIALS * ***sc**)

Arguments

- *GNUTLS_SRP_CLIENT_CREDENTIALS ** **sc**: is a pointer to an *GNUTLS_*SRP_CLIENT_CREDENTIALS structure.

Description

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

1.10.51 gnutls_srp_set_client_cred

int **gnutls_srp_set_client_cred** (*GNUTLS_SRP_CLIENT_CREDENTIALS* **res** , *char ** **username** , *char ** **password**)

Arguments

- *GNUTLS_SRP_CLIENT_CREDENTIALS* **res**: is an *GNUTLS_*SRP_CLIENT_CREDENTIALS structure.
- *char ** **username**: is the user's userid
- *char ** **password**: is the user's password

1.10.52 gnutls_srp_free_server_sc

void **gnutls_srp_free_server_sc** (*GNUTLS_SRP_SERVER_CREDENTIALS* **sc**)

Arguments

- *GNUTLS_SRP_SERVER_CREDENTIALS* **sc**: is an *GNUTLS*_SRP_SERVER_CREDENTIALS structure.

Description

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

1.10.53 gnutls_srp_allocate_server_sc

int **gnutls_srp_allocate_server_sc** (*GNUTLS_SRP_SERVER_CREDENTIALS * * **sc**)

Arguments

- *GNUTLS_SRP_SERVER_CREDENTIALS ** **sc**: is a pointer to an *GNUTLS*_SRP_SERVER_CREDENTIALS structure.

Description

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

1.10.54 gnutls_srp_set_server_cred_file

int **gnutls_srp_set_server_cred_file** (*GNUTLS_SRP_SERVER_CREDENTIALS* **res** , *char ** **password_file** , *char ** **password_conf_file**)

Arguments

- *GNUTLS_SRP_SERVER_CREDENTIALS* **res**: is an *GNUTLS*_SRP_SERVER_CREDENTIALS structure.
- *char ** **password_file**: is the SRP password file (tpasswd)
- *char ** **password_conf_file**: is the SRP password conf file (tpasswd.conf)

1.10.55 gnutls_x509pki_free_sc

void **gnutls_x509pki_free_sc** (*GNUTLS_X509PKI_CREDENTIALS* **sc**)

Arguments

- *GNUTLS_X509PKI_CREDENTIALS* **sc**: is an *GNUTLS*_X509PKI_CREDENTIALS structure.

Description

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

1.10.56 gnutls_x509pki_allocate_sc

int **gnutls_x509pki_allocate_sc** (*GNUTLS_X509PKI_CREDENTIALS ** **res** , *int* **ncerts**)

Arguments

- *GNUTLS_X509PKI_CREDENTIALS ** **res**: is a pointer to an *GNUTLS_*X509PKI_CREDENTIALS structure.
- *int* **ncerts**: this is the number of certificate/private key pair you're going to use. This should be 1 in common sites.

Description

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

1.10.57 gnutls_x509pki_set_key_file

int **gnutls_x509pki_set_key_file** (*GNUTLS_X509PKI_CREDENTIALS* **res** , *char ** **CERTFILE** , *char ** **KEYFILE**)

Arguments

- *GNUTLS_X509PKI_CREDENTIALS* **res**: is an *GNUTLS_*X509PKI_CREDENTIALS structure.
- *char ** **CERTFILE**: is a PEM encoded file containing the certificate list (path) for the specified private key
- *char ** **KEYFILE**: is a PEM encoded file containing a private key

Description

This function sets a certificate/private key pair in the GNUTLS_X509PKI_CREDENTIALS structure. This function may be called more than once (in case multiple keys/certificates exist for the server).

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

1.10.58 gnutls_x509pki_set_trust_file

int **gnutls_x509pki_set_trust_file** (*GNUTLS_X509PKI_CREDENTIALS* **res** , *char ** **CAFILE** , *char ** **CRLFILE**)

Arguments

- *GNUTLS_X509PKI_CREDENTIALS* **res**: is an *GNUTLS_*X509PKI_CREDENTIALS structure.

- *char ** **CAFILE**: is a PEM encoded file containing trusted CAs
- *char ** **CRLFILE**: is a PEM encoded file containing CRLs (ignored for now)

Description

This function sets the trusted CAs in order to verify client certificates.

1.10.59 gnutls_x509pki_set_dh_bits

void **gnutls_x509pki_set_dh_bits** (*GNUTLS_STATE* **state** , *int* **bits**)

Arguments

- *GNUTLS_STATE* **state**: is a *GNUTLS*_STATE structure.
- *int* **bits**: is the number of bits

Description

This function sets the number of bits, for use in a Diffie Hellman key exchange. This value will only be used in case of DHE ciphersuite.

1.10.60 gnutls_x509pki_server_set_cert_request

void **gnutls_x509pki_server_set_cert_request** (*GNUTLS_STATE* **state** , *CertificateRequest* **req**)

Arguments

- *GNUTLS_STATE* **state**: is an *GNUTLS*_STATE structure.
- *CertificateRequest* **req**: is one of GNUTLS_CERT_REQUEST, GNUTLS_CERT_REQUIRE

Description

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.

1.10.61 gnutls_x509pki_set_client_cert_callback

void **gnutls_x509pki_set_client_cert_callback** (*GNUTLS_STATE* **state** , *x509pki_client_cert_callback_func ** **func**)

Arguments

- *GNUTLS_STATE* **state**: is a *GNUTLS*_STATE structure.
- *x509pki_client_cert_callback_func ** **func**: is the callback function

Description

The callback's function form is: int (*callback)(GNUTLS_STATE, gnutls_datum *client_cert, int ncerts, gnutls_datum* req_ca_cert, int nreqs);

'client_cert' contains 'ncerts' gnutls_datum structures which hold the DER encoded X.509 certificates of the client.

'req_ca_cert' contains a list with the CA names that the server considers trusted. Normaly 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_x509pki_extract_dn().

This function specifies what we, in case of a client, are going to do when we have to send a certificate. If this callback function is not provided then gnutls will automaticaly try to find an appropriate certificate to send.

If the callback function is provided then gnutls will call it once with NULL parameters. If the callback function returns a positive or zero number then gnutls will attempt to automaticaly choose the appropriate certificate. If gnutls fails to find an appropriate certificate, then it will call the callback function again with the appropriate parameters.

In case the callback returned a negative number then gnutls will not attempt to choose the appropriate certificate and will call again the callback function with the appropriate parameters, and rely only to the return value of the callback function.

The callback function should return the index of the certificate choosen by the user. -1 indicates that the user does not want to use client authentication.

This function returns 0 on success.

1.10.62 gnutls_x509pki_set_server_cert_callback

void **gnutls_x509pki_set_server_cert_callback** (*GNUTLS_STATE* **state** , *x509pki_server_cert_callback_func ** **func**)

Arguments

- *GNUTLS_STATE* **state**: is a *GNUTLS_*STATE structure.
- *x509pki_server_cert_callback_func ** **func**: is the callback function

Description

The callback's function form is: int (*callback)(GNUTLS_STATE, gnutls_datum *server_cert, int ncerts);

'server_cert' contains 'ncerts' gnutls_datum structures which hold the DER encoded X.509 certificates of the server.

This function specifies what we, in case of a server, are going to do when we have to send a certificate. If this callback function is not provided then gnutls will automaticaly try to find an appropriate certificate to send. (actually send the first in the list)

In case the callback returned a negative number then gnutls will not attempt to choose the appropriate certificate and the caller function will fail.

The callback function will only be called once per handshake. The callback function should return the index of the certificate choosen by the server. -1 indicates an error.

1.10.63 gnutls_global_set_log_func

void **gnutls_global_set_log_func** (*LOG_FUNC* **log_func**)

Arguments

- *LOG_FUNC* **log_func**: it's a log function

Description

This is the function were you set the logging function gnutls is going to use. This function only accepts a character array. Normaly you may not use this function since it is only used for debugging reasons. LOG_FUNC is of the form, void (*LOG_FUNC)(const char*);

1.10.64 gnutls_global_init

int **gnutls_global_init** (**void**)

Arguments

- **void**:

Description

This function initializes the global state to defaults. Every gnutls application has a global state which holds common parameters shared by gnutls state structures. You must call gnutls_global_deinit() when gnutls usage is no longer needed Returns zero on success.

1.10.65 gnutls_global_deinit

void **gnutls_global_deinit** (**void**)

Arguments

- **void**:

Description

This function deinitializes the global state.

1.10.66 gnutls_transport_set_pull_func

void **gnutls_transport_set_pull_func** (*GNUTLS_STATE* **state** , *PULL_FUNC* **pull_func**)

Arguments

- *GNUTLS_STATE* **state**: gnutls state
- *PULL_FUNC* **pull_func**: it's a function like read

Description

This is the function where you set a function for gnutls to receive data. Normaly, if you use berkeley style sockets, you may not use this function since the default (recv(2)) will probably be ok. This function should be called once and after gnutls_global_init(). PULL_FUNC is of the form, ssize_t (*PULL_FUNC)(GNUTLS_SOCKET_PTR, const void*, size_t);

1.10.67 gnutls_transport_set_push_func

void **gnutls_transport_set_push_func** (*GNUTLS_STATE* **state** , *PUSH_FUNC* **push_func**)

Arguments

- *GNUTLS_STATE* **state**: gnutls state
- *PUSH_FUNC* **push_func**: it's a function like write

Description

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 may not 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.

This function should be called once and after gnutls_global_init(). PUSH_FUNC is of the form, ssize_t (*PUSH_FUNC)(GNUTLS_SOCKET_PTR, const void*, size_t);

1.10.68 gnutls_anon_free_server_sc

void **gnutls_anon_free_server_sc** (*GNUTLS_ANON_SERVER_CREDENTIALS* **sc**)

Arguments

- *GNUTLS_ANON_SERVER_CREDENTIALS* **sc**: is an *GNUTLS_*ANON_SERVER_CREDENTIALS structure.

Description

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

1.10.69 gnutls_anon_allocate_server_sc

int **gnutls_anon_allocate_server_sc** (*GNUTLS_ANON_SERVER_CREDENTIALS * * **sc**)

Arguments

- *GNUTLS_ANON_SERVER_CREDENTIALS ** **sc**: is a pointer to an *GNUTLS*_ANON_SERVER_CREDENTIALS structure.

Description

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

1.10.70 gnutls_anon_set_server_cred

int **gnutls_anon_set_server_cred** (*GNUTLS_ANON_SERVER_CREDENTIALS* **res** , *int* **dh_bits**)

Arguments

- *GNUTLS_ANON_SERVER_CREDENTIALS* **res**: is an *GNUTLS*_ANON_SERVER_CREDENTIALS structure.
- *int* **dh_bits**: is the number of bits in DH key exchange

Description

Used to set the number of bits to use in an anonymous Diffie-Hellman, key exchange.

1.10.71 gnutls_anon_free_client_sc

void **gnutls_anon_free_client_sc** (*GNUTLS_ANON_CLIENT_CREDENTIALS* **sc**)

Arguments

- *GNUTLS_ANON_CLIENT_CREDENTIALS* **sc**: is an *GNUTLS*_ANON_CLIENT_CREDENTIALS structure.

Description

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

1.10.72 gnutls_anon_allocate_client_sc

int **gnutls_anon_allocate_client_sc** (*GNUTLS_ANON_CLIENT_CREDENTIALS * **sc***)

Arguments

- *GNUTLS_ANON_CLIENT_CREDENTIALS ** **sc**: is a pointer to an *GNUTLS*_ANON_CLIENT_CREDENTIALS structure.

Description

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

1.10.73 gnutls_x509pki_extract_dn

int **gnutls_x509pki_extract_dn** (*const gnutls_datum ** **idn** , *gnutls_DN ** **rdn**)

Arguments

- *const gnutls_datum ** **idn**: should contain a DER encoded RDN sequence
- *gnutls_DN ** **rdn**: a pointer to a structure to hold the name

Description

This function will return the name of the given RDN sequence. The name will be returned as a gnutls_DN structure. Returns a negative error code in case of an error.

1.10.74 gnutls_x509pki_extract_certificate_dn

int **gnutls_x509pki_extract_certificate_dn** (*const gnutls_datum ** **cert** , *gnutls_DN ** **ret**)

Arguments

- *const gnutls_datum ** **cert**: should contain an X.509 DER encoded certificate
- *gnutls_DN ** **ret**: a pointer to a structure to hold the peer's name

Description

This function will return the name of the certificate holder. The name is gnutls_DN structure and is a obtained by the peer's certificate. If the certificate send by the peer is invalid, or in any other failure this function returns error. Returns a negative error code in case of an error.

1.10.75 gnutls_x509pki_extract_certificate_issuer_dn

int **gnutls_x509pki_extract_certificate_issuer_dn** (*const gnutls_datum ** **cert** , *gnutls_DN ** **ret**)

Arguments

- *const gnutls_datum ** **cert**: should contain an X.509 DER encoded certificate
- *gnutls_DN ** **ret**: a pointer to a structure to hold the issuer's name

Description

This function will return the name of the issuer stated in the certificate. The name is a gnutls_DN structure and is a obtained by the peer's certificate. If the certificate send by the peer is invalid, or in any other failure this function returns error. Returns a negative error code in case of an error.

1.10.76 gnutls_x509pki_extract_subject_dns_name

int **gnutls_x509pki_extract_subject_dns_name** (*const gnutls_datum ** **cert** , *char ** **ret** , *int ** **ret_size**)

Arguments

- *const gnutls_datum ** **cert**: should contain an X.509 DER encoded certificate
- *char ** **ret**: is the place where dns name will be copied to
- *int ** **ret_size**: holds the size of ret.

Description

This function will return the alternative name (the dns part of it), contained in the given certificate.

This is specified in X509v3 Certificate Extensions. GNUTLS will only return the dnsName of the Alternative name, or a negative error code. Returns GNUTLS_E_MEMORY_ERROR if ret_size is not enough to hold dns name, or the size of dns name if everything was ok.

If the certificate does not have a DNS name then returns GNUTLS_E_DATA_NOT_AVAILABLE;

1.10.77 gnutls_x509pki_extract_certificate_activation_time

time_t **gnutls_x509pki_extract_certificate_activation_time** (*const gnutls_datum ** **cert**)

Arguments

- *const gnutls_datum ** **cert**: should contain an X.509 DER encoded certificate

Description

This function will return the certificate's activation time in UNIX time (ie seconds since 00:00:00 UTC January 1, 1970). Returns a (time_t) -1 in case of an error.

1.10.78 gnutls_x509pki_extract_certificate_expiration_time

time_t **gnutls_x509pki_extract_certificate_expiration_time** (*const gnutls_datum ** **cert**)

Arguments

- *const gnutls_datum ** **cert**: should contain an X.509 DER encoded certificate

Description

This function will return the certificate's expiration time in UNIX time (ie seconds since 00:00:00 UTC January 1, 1970). Returns a (time_t) -1 in case of an error.

1.10.79 gnutls_x509pki_extract_certificate_version

int **gnutls_x509pki_extract_certificate_version** (*const gnutls_datum ** **cert**)

Arguments

- *const gnutls_datum ** **cert**: is an X.509 DER encoded certificate

Description

This function will return the X.509 certificate's version (1, 2, 3). This is obtained by the X509 Certificate Version field. Returns a negative value in case of an error.

1.10.80 gnutls_x509pki_get_peer_certificate_status

int **gnutls_x509pki_get_peer_certificate_status** (*GNUTLS_STATE* **state**)

Arguments

- *GNUTLS_STATE* **state**: is a gnutls state

Description

This function will try to verify the peer's certificate and return it's status (TRUSTED, EXPIRED etc.). The return value (status) should be one of the

CertificateStatus enumerated elements. However you must also check the peer's name in order to check if the verified certificate belongs to the actual peer. Returns a negative error code in case of an error, or GNUTLS_CERT_NONE if no certificate was sent.

1.10.81 gnutls_x509pki_extract_certificate_serial

int **gnutls_x509pki_extract_certificate_serial** (*const gnutls_datum* * **cert** , *char** **result** , *int** **result_size**)

Arguments

- *const gnutls_datum* * **cert**: is an X.509 DER encoded certificate
- *char** **result**: The place where the serial number will be copied
- *int** **result_size**: Holds the size of the result field.

Description

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. Returns a negative value in case of an error.

1.10.82 gnutls_x509pki_verify_certificate

int **gnutls_x509pki_verify_certificate** (*const gnutls_datum** **cert_list** , *int* **cert_list_length** , *const gnutls_datum* * **CA_list** , *int* **CA_list_length** , *const gnutls_datum** **CRL_list** , *int* **CRL_list_length**)

Arguments

- *const gnutls_datum** **cert_list**: is the certificate list to be verified
- *int* **cert_list_length**: holds the number of certificate in cert_list
- *const gnutls_datum* * **CA_list**: is the CA list which will be used in verification
- *int* **CA_list_length**: holds the number of CA certificate in CA_list
- *const gnutls_datum** **CRL_list**: not used
- *int* **CRL_list_length**: not used

Description

This function will try to verify the given certificate list and return it's status (TRUSTED, EXPIRED etc.). The return value (status) should be one of the CertificateStatus enumerated elements. However you must also check the peer's name in order to check if the verified certificate belongs to the actual peer. Returns a negative error code in case of an error.

1.10.83 gnutls_srp_server_get_username

*const char ** **gnutls_srp_server_get_username** (*GNUTLS_STATE* **state**)

Arguments

- *GNUTLS_STATE* **state**: is a gnutls state

Description

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.

1.10.84 gnutls_anon_server_get_dh_bits

int **gnutls_anon_server_get_dh_bits** (*GNUTLS_STATE* **state**)

Arguments

- *GNUTLS_STATE* **state**: is a gnutls state

Description

This function will return the bits used in the Diffie Hellman authentication with the peer. This should only be called in case of a server. Returns a negative value in case of an error.

1.10.85 gnutls_anon_client_get_dh_bits

int **gnutls_anon_client_get_dh_bits** (*GNUTLS_STATE* **state**)

Arguments

- *GNUTLS_STATE* **state**: is a gnutls state

Description

This function will return the bits used in the Diffie Hellman authentication with the peer. This should only be called in case of a client. Returns a negative value in case of an error.

1.10.86 gnutls_x509pki_get_peer_certificate_list

*const gnutls_datum ** **gnutls_x509pki_get_peer_certificate_list** (*GNUTLS_STATE* **state** , *int ** **list_size**)

Arguments

- *GNUTLS_STATE* **state**: is a gnutls state
- *int ** **list_size**: is the length of the certificate list

Description

This function will return the peer's raw certificate list as sent by the peer. These certificates are DER encoded. The first certificate in the list is the peer's certificate, following the issuer's certificate, then the issuer's issuer etc. Returns NULL in case of an error, or if no certificate was sent.

1.10.87 gnutls_x509pki_get_dh_bits

int **gnutls_x509pki_get_dh_bits** (*GNUTLS_STATE* **state**)

Arguments

- *GNUTLS_STATE* **state**: is a gnutls state

Description

This function will return the number of bits used in a Diffie Hellman Handshake. This will only occur in case of DHE_* ciphersuites. The return value may be zero if no applicable ciphersuite was used. Returns a negative value in case of an error.

1.10.88 gnutls_x509pki_get_certificate_request_status

int **gnutls_x509pki_get_certificate_request_status** (*GNUTLS_STATE* **state**)

Arguments

- *GNUTLS_STATE* **state**: is a gnutls state

Description

This function will return 0 if the peer (server) did not request client authentication or 1 otherwise. Returns a negative value in case of an error.

1.10.89 gnutls_fingerprint

int **gnutls_fingerprint** (*DigestAlgorithm* **algo** , *const gnutls_datum** **data** , *char** **result** , *int** **result_size**)

Arguments

- *DigestAlgorithm* **algo**: is a digest algorithm
- *const gnutls_datum** **data**: is the data
- *char** **result**: is the place where the result will be copied.
- *int** **result_size**: should hold the size of the result. The actual size of the returned result will also be copied there.

Description

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. Returns a negative value in case of an error.

1.10.90 gnutls_dh_replace_params

int **gnutls_dh_replace_params** (*gnutls_datum* **prime** , *gnutls_datum* **generator** , *int* **bits**)

Arguments

- *gnutls_datum* **prime**: holds the new prime
- *gnutls_datum* **generator**: holds the new generator
- *int* **bits**: is the prime's number of bits

Description

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. This function should not be called while a key exchange is in progress.

Note that the bits value should be one of 1024, 2048, 3072 or 4096.

1.10.91 gnutls_dh_generate_params

int **gnutls_dh_generate_params** (*gnutls_datum** **prime** , *gnutls_datum** **generator** , *int* **bits**)

Arguments

- *gnutls_datum** **prime**: will hold the new prime
- *gnutls_datum** **generator**: will hold the new generator
- *int* **bits**: is the prime's number of bits

Description

This function will generate a new pair of prime and generator for use in the Diffie-Hellman key exchange. The new parameters will be stored in the appropriate gnutls_datum. This function is normally very slow. An other function (gnutls_dh_replace_params()) should be called in order to replace the included DH primes in the gnutls library.

Note that the bits value should be one of 1024, 2048, 3072 or 4096. Also note that the generation of new DH parameters is only usefull to servers. Clients use the parameters sent by the server, thus it's no use calling this in client side.

1.10.92 gnutls_alert_send

int **gnutls_alert_send** (*GNUTLS_STATE* **state** , *AlertLevel* **level** , *AlertDescription* **desc**)

Arguments

- *GNUTLS_STATE* **state**: is a *GNUTLS_*STATE structure.
- *AlertLevel* **level**: is the level of the alert
- *AlertDescription* **desc**: is the alert description

Description

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. Returns 0 on success.

1.10.93 gnutls_alert_send_appropriate

int **gnutls_alert_send_appropriate** (*GNUTLS_STATE* **state** , *int* **err**)

Arguments

- *GNUTLS_STATE* **state**: is a *GNUTLS_*STATE structure.
- *int* **err**: is an integer

Description

Sends an alert to the peer depending on the error code returned by a gnutls function. All alerts sent by this function are fatal, so connection should be considered terminated after calling this function. The only exception is when err == GNUTLS_E_REHANDSHAKE, then a warning alert is sent to the peer indicating the no renegotiation will be performed.

This function may also return GNUTLS_E_AGAIN, or GNUTLS_E_INTERRUPTED.

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

1.10.94 gnutls_alert_get_last

AlertDescription **gnutls_alert_get_last** (*GNUTLS_STATE* **state**)

Arguments

- *GNUTLS_STATE* **state**: is a *GNUTLS_*STATE structure.

Description

Returns 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.

Chapter 2

ASN.1 Structure Handling

2.1 Introduction

This file describes the forth version of ASN.1 parser I developed[1]. The main difference from the first version is the use of pointers and the possibility to save/get ASN1 definitions in/from a C vector. Other differences are:

- write_value function for type ANY
- the introduction of ENUMERATED type,
- negative integer are allowed in ASN.1 syntax files,
- PKIX1Implicit88.txt instead of Certificate.txt for the Certificate description
- functions naming
- an easier way to set INTEGER and get OBJECT IDENTFIER

2.2 ASN.1 Syntax

The parser is case sensitive. The comments begin with ”– ” and end at the end of line. An example is in ”Certificate.txt” file. The ASN.1 declarations must have this form:

```
object_name {<object definition>}

DEFINITIONS <EXPLICIT or IMPLICIT> TAGS ::=
```

[1] ASN.1 Structure handling in GNUTLS is developed by Fabio Fiorina

```
BEGIN

<type and constants definitions>

END
```

The token "::=" must be separate from others elements, so this is a wrong declaration: Version ::=INTEGER the correct one is : Version ::= INTEGER Here is the list of types that the parser can manage:

- INTEGER
- ENUMERATED
- BOOLEAN
- OBJECT IDENTIFIER
- NULL
- BIT STRING
- OCTET STRING
- UTCTime
- GeneralizedTime
- SEQUENCE
- SEQUENCE OF
- SET
- SET OF
- CHOICE
- ANY
- ANY DEFINED BY

This version doesn't manage REAL type. It also not allow the use of "EXPORT" and "IMPORT" sections.

The SIZE constraints are allowed but no check is done on them.

2.3 Naming

If you have this definitions:

```
Example { 1 2 3 4 }

DEFINITIONS EXPLICIT TAGS ::=

BEGIN

Group ::= SEQUENCE {
   id   OBJECT IDENTIFIER,
   value  Value
}

Value ::= SEQUENCE {
   value1  INTEGER,
   value2  BOOLEAN
}

END
```

to identify the type 'Group' you have to use the null terminated string "Example.Group". Others examples: Field 'id' in 'Group' type : "Example.Group.id" Field 'value1' in filed 'value' in type 'Group': "Example.Group.value.value1" These strings are used in functions that are described below. Elements of structured types that don't have a name, receve the name "?1","?2", and so on. The name "?LAST" indicates the last element of a SET_OF or SEQUENCE_OF.

2.4 Future Developments

1. type REAL
2. improve the error signaling with strings that give you more details. Examples: in case of ASN1 syntax error you will have the line number where the error is, if creating a der encoding the result is ASN_VALUE_NOT_FOUND you will have the name of the element without the value.
3. improve the 'visit_tree' function and change the output from stdout to a null terminated string.

2.5 Function Reference

2.5.1 asn1_parser_asn1

int **asn1_parser_asn1** (*char* * **file_name** , *node_asn* ** **pointer**)

Arguments

- *char* * **file_name**: specify the path and the name of file that contains ASN.1 declarations.
- *node_asn* ** **pointer**: return the pointer to the structure created from "file_name" ASN.1 declarations.

Description

Creates the structures needed to manage the definitions included in *FILE_NAME file.

Returns

ASN_OK

the file has a correct syntax and every identifier is known.

ASN_FILE_NOT_FOUND

an error occured while opening FILE_NAME.

ASN_SYNTAX_ERROR

the syntax is not correct.

ASN_IDENTIFIER_NOT_FOUND

in the file there is an identifier that is not defined.

2.5.2 asn1_parser_asn1_file_c

int **asn1_parser_asn1_file_c** (*char* * **file_name**)

Arguments

- *char* * **file_name**: specify the path and the name of file that contains ASN.1 declarations.

Description

Creates a file containing a C vector to use to manage the definitions included in *FILE_NAME file. If *FILE_NAME is "/aa/bb/xx.yy" the file created is "/aa/bb/xx_asn1_tab.c", and the vector is "xx_asn1_tab".

Returns

ASN_OK

the file has a correct syntax and every identifier is known.

ASN_FILE_NOT_FOUND

an error occured while opening FILE_NAME.

ASN_SYNTAX_ERROR

the syntax is not correct.

ASN_IDENTIFIER_NOT_FOUND

in the file there is an identifier that is not defined.

2.5.3 asn1_create_tree

int **asn1_create_tree** (*const static_asn* * **root** , *node_asn* ** **pointer**)

Arguments

- *const static_asn* * **root**: specify vector that contains ASN.1 declarations
- *node_asn* ** **pointer**: return the pointer to the structure created by *ROOT ASN.1 declarations

Description

Creates the structures needed to manage the ASN1 definitions. ROOT is a vector created by 'asn1_parser_asn1_file_c' function.

Returns

ASN_OK

structure created correctly.

ASN_GENERIC_ERROR

an error occured while structure creation

2.5.4 asn1_visit_tree

void **asn1_visit_tree** (*node_asn* * **pointer** , *char* * **name**)

Arguments

- *node_asn* * **pointer**: pointer to the structure that you want to delete.
- *char* * **name**: an element of the structure

Description

Prints on the standard output the structure's tree starting from the NAME element inside the structure *POINTER.

2.5.5 asn1_delete_structure

int **asn1_delete_structure** (*node_asn* * **root**)

Arguments

- *node_asn* * **root**: pointer to the structure that you want to delete.

Description

Deletes the structure *POINTER.

Returns

ASN_OK

everything OK

ASN_ELEMENT_NOT_FOUND

pointer==NULL.

2.5.6 asn1_create_structure

int **asn1_create_structure** (*node_asn* * **root** , *char* * **source_name** , *node_asn* ** **pointer** , *char* * **dest_name**)

Arguments

- *node_asn* * **root**: pointer to the structure returned by "parser_asn1" function
- *char* * **source_name**: the name of the type of the new structure (must be inside p_structure).
- *node_asn* ** **pointer**: pointer to the structure created.
- *char* * **dest_name**: the name of the new structure.

Description

Creates a structure called DEST_NAME of type SOURCE_NAME.

Returns

ASN_OK

creation OK

ASN_ELEMENT_NOT_FOUND

SOURCE_NAME isn't known

Example

using "pkix.asn" result=asn1_create_structure(cert_def,"PKIX1Implicit88.Certificate",*cert*,"certificate1");

2.5.7 asn1_write_value

int **asn1_write_value** (*node_asn* * **node_root** , *char* * **name** , *unsigned char* * **value** , *int* **len**)

Arguments

- *node_asn* * **node_root**: pointer to a structure
- *char* * **name**: the name of the element inside the structure that you want to set.
- *unsigned char* * **value**: vector used to specify the value to set. If len is ¿0, VALUE must be a two's complement form integer. if len=0 *VALUE must be a null terminated string with an integer value.
- *int* **len**: number of bytes of *value to use to set the value: value[0]..value[len-1] or 0 if value is a null terminated string

Description

Set the value of one element inside a structure.

Returns

ASN_OK

set value OK

ASN_ELEMENT_NOT_FOUND

NAME is not a valid element.

ASN_VALUE_NOT_VALID

VALUE has a wrong format.

Examples

description for each type

INTEGER

VALUE must contain a two's complement form integer. value[0]=0xFF , len=1 → integer=-1 value[0]=0xFF value[1]=0xFF , len=2 → integer=-1 value[0]=0x01 , len=1 → integer= 1 value[0]=0x00 value[1]=0x01 , len=2 → integer= 1 value="123" , len=0 → integer= 123

ENUMERATED

as INTEGER (but only with not negative numbers)

BOOLEAN

VALUE must be the null terminated string "TRUE" or "FALSE" and LEN != 0 value="TRUE" , len=1 → boolean=TRUE value="FALSE" , len=1 → boolean=FALSE

OBJECT IDENTIFIER

VALUE must be a null terminated string with each number separated by a blank (e.g. "1 2 3 543 1"). LEN != 0 value="1 2 840 10040 4 3" , len=1 → OID=dsa-with-sha

UTCTime

VALUE must be a null terminated string in one of these formats: "YYMMDDhhmmssZ" "YYMMDDhhmmssZ" "YYMMDDhhmmss+hh'mm'" "YYMMDDhhmmss-hh'mm'" "YYMMDDhhmm+hh'mm'" "YYMMDDhhmm-hh'mm'". LEN != 0 value="9801011200Z" , len=1 → time=Jannuary 1st, 1998 at 12h 00m Greenwich Mean Time

GeneralizedTime

VALUE must be in one of this format: "YYYYMMDDhhmmss.sZ" "YYYYMMDDhhmmss.sZ" "YYYYMMDDhhmmss.s+hh'mm'" "YYYYMMDDhhmmss.s-hh'mm'" "YYYYMMDDhhmm+hh'mm'" "YYYYMMDDhhmm-hh'mm'" where ss.s indicates the seconds with any precision like "10.1" or "01.02". LEN != 0 value="2001010112001.12-0700" , len=1 → time=Jannuary 1st, 2001 at 12h 00m 01.12s Pacific Daylight Time

OCTET STRING

VALUE contains the octet string and LEN is the number of octet. value="\x01\x02\x03" , len=3 → three bytes octet string

BIT STRING

VALUE contains the bit string organized by bytes and LEN is the number of bits. value="\xCF" , len=6 → bit string="110011" (six bits)

CHOICE

if NAME indicates a choice type, VALUE must specify one of the alternatives with a null terminated string. LEN != 0 Using "pkix.asn": result=asn1_write_value(cert,"certificate1.tbsCertificate.subject","rdnSequence",1);

ANY

VALUE indicates the der encoding of a structure. LEN != 0

SEQUENCE OF

VALUE must be the null terminated string "NEW" and LEN != 0. With this instruction another element is appended in the sequence. The name of this element will be "?1" if it's the first one, "?2" for the second and so on. Using "pkix.asn": result=asn1_write_value(cert,"certificate1.tbsCertificate.subject.rdnSequence","NEW",1);

SET OF

the same as SEQUENCE OF. Using "pkix.asn": result=asn1_write_value(cert,"certificate1.tbsCertificate.subject.rdnSequence.?LAST","NEW",1);

If an element is OPTIONAL and you want to delete it, you must use the value=NULL and len=0. Using "pkix.asn": result=asn1_write_value(cert,"certificate1.tbsCertificate.issuerUniqueID",NULL,0);

2.5.8 asn1_read_value

int **asn1_read_value** (*node_asn* * **root** , *char* * **name** , *unsigned char* * **value** , *int* * **len**)

Arguments

- *node_asn* * **root**: pointer to a structure
- *char* * **name**: the name of the element inside a structure that you want to read.
- *unsigned char* * **value**: vector that will contain the element's content. VALUE must be a pointer to memory cells already allocated.
- *int* * **len**: number of bytes of *value: value[0]..value[len-1]

Description

Returns the value of one element inside a structure.

Returns

ASN_OK

set value OK

ASN_ELEMENT_NOT_FOUND

NAME is not a valid element.

ASN_VALUE_NOT_FOUND

there isn't any value for the element selected.

Examples

a description for each type

INTEGER

VALUE will contain a two's complement form integer. integer=-1 → value[0]=0xFF , len=1 integer=1 → value[0]=0x01 , len=1

ENUMERATED

as INTEGER (but only with not negative numbers)

BOOLEAN

VALUE will be the null terminated string "TRUE" or "FALSE" and LEN=5 or LEN=6

OBJECT IDENTIFIER

VALUE will be a null terminated string with each number separated by a blank (i.e. "1 2 3 543 1"). LEN = strlen(VALUE)+1

UTCTime

VALUE will be a null terminated string in one of these formats: "YYMMDDhhmmss+hh'mm'" or "YYMMDDhhmmss-hh'mm'" LEN=strlen(VALUE)+1

GeneralizedTime

VALUE will be a null terminated string in the same format used to set the value

OCTET STRING

VALUE will contain the octet string and LEN will be the number of octet.

BIT STRING

VALUE will contain the bit string organized by bytes and LEN will be the number of bits.

CHOICE

if NAME indicates a choice type, VALUE will specify the alternative selected

ANY

if NAME indicates an any type, VALUE will indicate the DER encoding of the structure actually used.

If an element is OPTIONAL and the function "read_value" returns ASN_ELEMENT_NOT_FOUND, it means that this element wasn't present in the der encoding that created the structure. The first element of a SEQUENCE_OF or SET_OF is named "?1". The second one "?2" and so on.

2.5.9 asn1_create_der

int **asn1_create_der** (*node_asn* * **root** , *char* * **name** , *unsigned char* * **der** , *int* * **len**)

Arguments

- *node_asn* * **root**: pointer to a structure
- *char* * **name**: the name of the structure you want to encode (it must be inside *POINTER).
- *unsigned char* * **der**: vector that will contain the DER encoding. DER must be a pointer to memory cells already allocated.
- *int* * **len**: number of bytes of *der: der[0]..der[len-1]

Description

Creates the DER encoding for the NAME structure (inside *POINTER structure).

Returns

ASN_OK

DER encoding OK

ASN_ELEMENT_NOT_FOUND

NAME is not a valid element.

ASN_VALUE_NOT_FOUND

there is an element without a value.

2.5.10 asn1_get_der

int **asn1_get_der** (*node_asn* * **root** , *unsigned char* * **der** , *int* **len**)

Arguments

- *node_asn* * **root**: pointer to a structure
- *unsigned char* * **der**: vector that contains the DER encoding.
- *int* **len**: number of bytes of *der: der[0]..der[len-1]

Description

Fill the structure *POINTER with values of a DER encoding string. The sructure must just be created with function 'create_stucture'.

Returns

ASN_OK

DER encoding OK

ASN_ELEMENT_NOT_FOUND

NAME is not a valid element. ASN_TAG_ERROR,ASN_DER_ERROR: the der encoding doesn't match the structure NAME.

2.5.11 asn1_get_start_end_der

int **asn1_get_start_end_der** (*node_asn* * **root** , *unsigned char* * **der** , *int* **len** , *char* * **name_element** , *int* * **start** , *int* * **end**)

Arguments

- *node_asn* * **root**: pointer to a structure
- *unsigned char* * **der**: vector that contains the DER encoding.
- *int* **len**: number of bytes of *der: der[0]..der[len-1]
- *char* * **name_element**: an element of NAME structure.
- *int* * **start**: the position of the first byte of NAME_ELEMENT decoding (der[*start])
- *int* * **end**: the position of the last byte of NAME_ELEMENT decoding (der[*end])

Description

Find the start and end point of an element in a DER encoding string. I mean that if you have a der encoding and you have already used the function "get_der" to fill a structure, it may happen that you want to find the piece of string concerning an element of the structure.

Example

the sequence "tbsCertificate" inside an X509 certificate.

Returns

ASN_OK

DER encoding OK

ASN_ELEMENT_NOT_FOUND

NAME or NAME_ELEMENT is not a valid element. ASN_TAG_ERROR,ASN_DER_ERROR: the der encoding doesn't match the structure NAME.

Chapter 3

GNU Free Documentation License

Version 1.1, March 2000

Preamble

The purpose of this License is to make a manual, textbook, or other written document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

3.1 Applicability and Definitions

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you".

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, “Title Page” means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

3.2 Verbatim Copying

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3.3 Copying in Quantity

If you publish printed copies of the Document numbering more than 100, and the Document’s license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

3.4 Modifications

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five).
- State on the Title page the name of the publisher of the Modified Version, as the publisher.
- Preserve all the copyright notices of the Document.
- Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- Include an unaltered copy of this License.
- Preserve the section entitled "History", and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

- Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

- In any section entitled "Acknowledgements" or "Dedications", preserve the section's title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

- Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

- Delete any section entitled "Endorsements". Such a section may not be included in the Modified Version.

- Do not retitle any existing section as "Endorsements" or to conflict in title with any Invariant Section.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties – for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

3.5 Combining Documents

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled "History" in the various original documents, forming one section entitled "History"; likewise combine any sections entitled "Acknowledgements", and any sections entitled "Dedications". You must delete all sections entitled "Endorsements."

3.6 Collections of Documents

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

3.7 Aggregation With Independent Works

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an "aggregate", and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document's Cover Texts may be placed on covers that surround only the

Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

3.8 Translation

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

3.9 Termination

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

3.10 Future Revisions of This License

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright © YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have no Invariant Sections, write "with no Invariant Sections" instead of saying which ones are invariant. If you have no Front-Cover Texts, write "no Front-Cover Texts" instead of "Front-Cover Texts being LIST"; likewise for Back-Cover Texts.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.