

---

# RT-THREAD MBEDTLS User Manual

---

RT-THREAD Documentation Center

Copyright ©2019 Shanghai Ruiside Electronic Technology Co., Ltd.



[WWW.RT-THREAD.ORG](http://WWW.RT-THREAD.ORG)

Friday 28th September, 2018

Versions and Revisions

Date	Version	Author	Note
2018-08-01	v0.1	MurphyZhao	Initial version
2018-08-14	v0.2	MurphyZhao	Update Certificate Add Method increase resource usage
Use the Optimization Manual			

[Table of contents](#)

Versions and Revisions	i
Table of contents	ii
1 Versions and Revisions	1
2 Software Package Introduction	2
2.1 Software framework diagram.	2
2.2 Package directory structure.	3
3 Sample Program	4
3.1 Routine workflow.	4
3.2 Preparation.	5
3.2.1 Obtaining the Software Package . . .	5
3.2.2 Synchronize device time. . .	5
3.3 Startup routine.	6
4 Working Principle	8
4.1 SSL/TLS handshake process . . .	9
4.2 DTLS handshake process.	9
5. Usage Guidelines	11
5.1 menuconfig configuration instructions. . .	11
5.2 Functional configuration description.	13
5.3 Certificate configuration instructions.	13
5.4 Initialize TLS session.	13

5.5 Initialize the SSL/TLS client. . . . .	14
5.6 Initialize the SSL/TLS client context. . . . .	14
5.7 Establishing an SSL/TLS connection. . . . .	15
5.8 Reading and writing data. . . . .	15
5.9 Closing SSL/TLS client connections. . . . .	16
5.10 mbedtls usage paradigm. . . . .	17
5.11 Add new certificates. . . . .	17
5.11.1 Root Certificate Style . . . . .	17
5.11.2 Obtaining the root certificate . . . . .	19
5.11.3 Import the certificate. . . . .	twenty four
5.12 Frequently Asked Questions. . . . .	25
5.12.1 Certificate verification failed. . . . .	25
5.12.2 Certificate time error. . . . .	25
5.12.3 Certificate CN is incorrect. . . . .	25
5.12.4 0x7200 Error . . . . .	26
5.13 References. . . . .	26
<b>6 MbedTLS RAM and ROM resource usage optimization guide</b>	<b>27</b>
6.1 Optimization description. . . . .	27
6.2 Summary of resource usage after optimization. . . . .	28
6.3 Preparation before optimization. . . . .	29
6.4 Optimization configuration overview. . . . .	29
6.4.1 Commonly used optimization configurations. . . . .	29
6.4.2 System related configuration. . . . .	31
6.4.3 Functional component related configuration. . . . .	33
6.4.4 Cipher suite related configuration . . . . .	40
6.4.5 Elliptic curve related configuration. . . . .	43
6.4.6 TLS version selection related configuration. . . . .	44
6.4.7 DTLS related configuration. . . . .	44
6.5 References. . . . .	45

<b>7 API Description</b>	<b>50</b>
7.1 Application layer API . . . . .	50
7.1.1 mbedtls initialization. . . . .	50
7.1.2 Configuring mbedtls context. . . . .	51
7.1.3 Establishing an SSL/TLS connection. . . . .	51
7.1.4 Read data. . . . .	51
7.1.5 Close the mbedtls client. . . . .	52
7.2 mbedtls related APIs . . . . .	53
7.2.1 Setting the debug level. . . . .	53
7.2.2 Initialization phase related API . . . . .	53
7.2.3 Connection phase related API . . . . .	59
7.2.4 Read and write API . . . . .	61

# Chapter 1

## Versions and Revisions

Date	Version	Author	Note
2018-08-01	v0.1	MurphyZhao	Initial version
2018-08-14	v0.2	MurphyZhao	Update Certificate Add Method increase resource usage
Use the Optimization Manual			

## chapter 2

### Software Package Introduction

**mbdttls** The software package is **RT-Thread** based on **ARMmbed/mbedtls** Porting of open source libraries.

**mbedtls** (formerly PolarSSL) is an open-source SSL/TLS algorithm library maintained by ARM. It uses the C programming language to implement SSL/TLS functions and various encryption algorithms with minimal coding space. It is easy to understand, use, integrate and expand, making it easy for developers to use SSL/TLS functions in embedded products.

The **mbedtls** package provides the following capabilities:

- Complete **SSL v3**, **TLS v1.0**, **TLS v1.1** and **TLS v1.2** protocol implementation • **X.509** certificate processing • TLS transport encryption based on TCP • DTLS (Datagram TLS) transport encryption based on UDP •
- Other encryption and decryption library implementations

For more information about mbedtls, see <https://tls.mbed.org>.

#### 2.1 Software Framework Diagram

The mbedtls package provides a set of cryptographic components that can be used and compiled separately. The components and their possible dependencies are shown in the following figure:

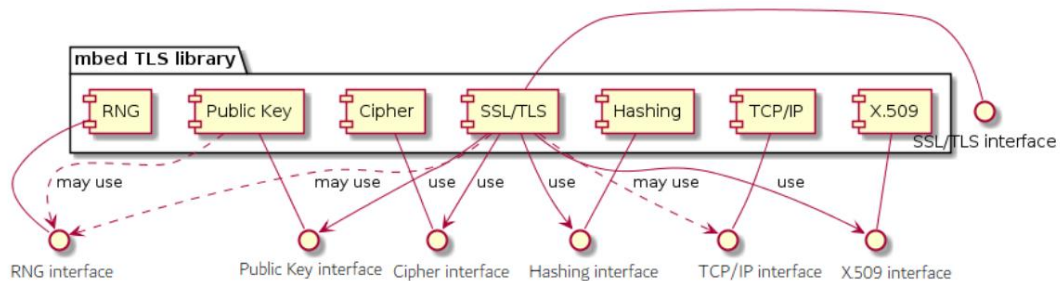


Figure 2.1: *mbedtls*

Software framework diagram

## 2.2 Software package directory structure

The **ports** directory contains the porting files involved in porting the mbedtls package to RT-Thread. Use `scons` to Rebuild the row.

```

mbedtls
  LICENSE                                     // Software package license agreement
  || README.md |                             // Software package instructions
  SConscript                                //RT-Thread default build script
  +---certs                                // The certs root directory stores the user CA certificate
  | +---default                             // The default directory stores the preset CA certificates
  +---docs
  | +---figures || api.md |                 // Document using images
  | introduction.md || |                  // API usage instructions
  principle.md || Footprint-optimization-  // Software package details
    LICENSE                               // License file
  guide.md // Resource Occupancy           // Implementation principle
  Optimization Reference Guide
  || README.md ||                          // Document structure description
  samples.md || user-guide.md              // Package Example
  | +---version.md                         // Instructions for use
                                          // Version description
                                          // Migrate files
+---ports
  | +---inc
  | +---src
+---samples                                // Sample program
+---mbedtls                               // ARM mbedtls source code

```



## Chapter 3

### Sample Program

This sample program provides a simple TLS client to establish a TLS connection with a test website and obtain encrypted data.

Sample Files

Sample program path	illustrate
<code>samples/tls_app_test.c</code>	TLS test routine

### 3.1 Routine Workflow

This example uses the RT-Thread official TLS test website [www.rt-thread.org](http://www.rt-thread.org) and the `mbd_tls_client_write` function to send an HTTP test request. Upon success, the website returns text data, and the test example outputs the parsed data to the console.

- The HTTP request data used by the routine is as follows

```
"GET /download/rt-thread.txt HTTP/1.0\r\n"
"Host: www.rt-thread.org\r\n"
"User-Agent: rtthread/3.1 rt\r\n" "\r\n";
```

- The basic workflow of the mbedTLS test routine is as follows
  - The client connects to the test website [www.rt-thread.org](http://www.rt-thread.org) – The handshake between the client and the server is successful –
  - The client sends a request –
  - The server responds to the request
  - TLS test success/failure

## 3.2 Preparation

### 3.2.1 Obtaining the Software Package

- menuconfig configuration package

Open the ENV tool provided by RT-Thread and use **menuconfig** to configure the software package.

Enable the mbedtls package and configure the enable test example (Enable a mbedtls client example) as shown below:

```
RT-Thread online packages ---> security
packages --->
    Select Root Certificate --->                                #Select the certificate file
[*] mbedtls: An portable and flexible SSL/TLS library #Open mbedtls
    Software Packages
[*] Store the AES tables in ROM                                #Store AES tables in ROM #Maximum
    (2) Maximum window size used                               "window" size for dot multiplication (
        2-7)
    (3584) Maxium fragment length in bytes #Configure data frame size
[*] Enable a mbedtls client example # Enable mbedtls test example
[] Enable Debug log output                                    # Enable debug log output # Select
    version (latest) --- >                                    the software package version, the default is the latest version
```

- Use the `pkgs --update` command to download the software package •

Compile and download

### 3.2.2 Synchronize device time

During the certificate verification process of the SSL/TLS server, the time of initiating the verification request will be authenticated, such as If the time does not meet the server's requirements, the certificate verification will fail. Therefore, we need to synchronize the local time for the device.

- Method 1: Use the `date` command

The `date` command is entered on a device that has not synchronized its time, as shown below:

```
msh />date
Thu Jan 1 00:00:06 1970
```

Use `date` to set the current time as follows:

```
msh />date 2018 08 02 12 23 00
msh />date
Thu Aug 2 12:23:01 2018
msh />
```

- Method 2: Use NTP to synchronize network time

This method requires the NTP toolkit, which can be obtained using `menuconfig` configuration, as shown below:

```
RT-Thread online packages --->
IoT - internet of things --->
  *- netutils: Networking utilities for RT-Thread ---> *- Enable NTP(Network
    Time Protocol) client (8) (cn.ntp.org.cn) NTP server name
    Timezone for calculate local time
```

Use the `ntp_sync` command to synchronize network time

```
msh />ntp_sync
Get local time from NTP server: Thu Aug 2 14:31:30 2018 The system time is
updated. Timezone is 8.
msh />date
Thu Aug 2 14:31:34 2018
```

### 3.3 Startup routine

Use the command `tls_test` in MSH to execute the sample program. After successfully establishing a TLS connection, the device will receive

After getting a set of cipher suites, the device log is as follows:

```
msh />tls_test
MbedTLS test sample!
Memory usage before the handshake connection is established: total memory:
33554408 used memory :
20968 maximum allocated
memory: 20968
Start handshake tick:3313
[tls]mbedtls client struct init success... [tls]Loading the CA
root certificate success... [tls]mbedtls client context init success...
msh />[tls]Connected www.rt-thread.org:443 success...
```

```
[tls]Certificate verified success...
Finish handshake tick:6592
MbedTLS connect success...

Memory usage after the handshake connection is established: total memory: 33554408

used memory : 45480
maximum allocated memory: 50808 Writing HTTP
request success...
Getting HTTP response...
HTTP/1.1 200 OK
Server: nginx/1.10.3 (Ubuntu)
Date: Fri, 31 Aug 2018 08:29:24 GMT Content-Type:
text/plain Content-Length: 267 Last-
Modified: Sat, 04 Aug 2018
02:14:51 GMT Connection: keep-alive ETag: "5b650c1b- 10b"

Strict-Transport-Security: max-age=1800; includeSubdomains; preload Accept-Ranges: bytes

RT-Thread is an open source IoT operating system from China, which has strong scalability: from a tiny kernel
running on a tiny core, for example ARM Cortex-M0, or Cortex-M3/4/7, to a rich feature system running
on MIPS32, ARM Cortex-A8, ARM Cortex-A9 DualCore etc.

MbedTLS connection close success.
```

## Chapter 4

### working principle

The **mbdttls** package is an implementation of the SSL/TLS protocol. Both SSL (Secure Sockets Layer) and TLS (Transport Security Layer) are designed to ensure the security of information during transmission. They encrypt data based on plain text transmission and then transmit data in ciphertext.

mbdTLS needs to go through the following steps to establish a secure communication connection:

- Initialize SSL/TLS context • Establish SSL/TLS

- handshake • Send and receive data •

Interaction completed, close the

connection

Among them, the most critical step is the establishment of the **SSL/TLS** handshake connection, which requires certificate verification.

4.1 SSL/TLS handshake process

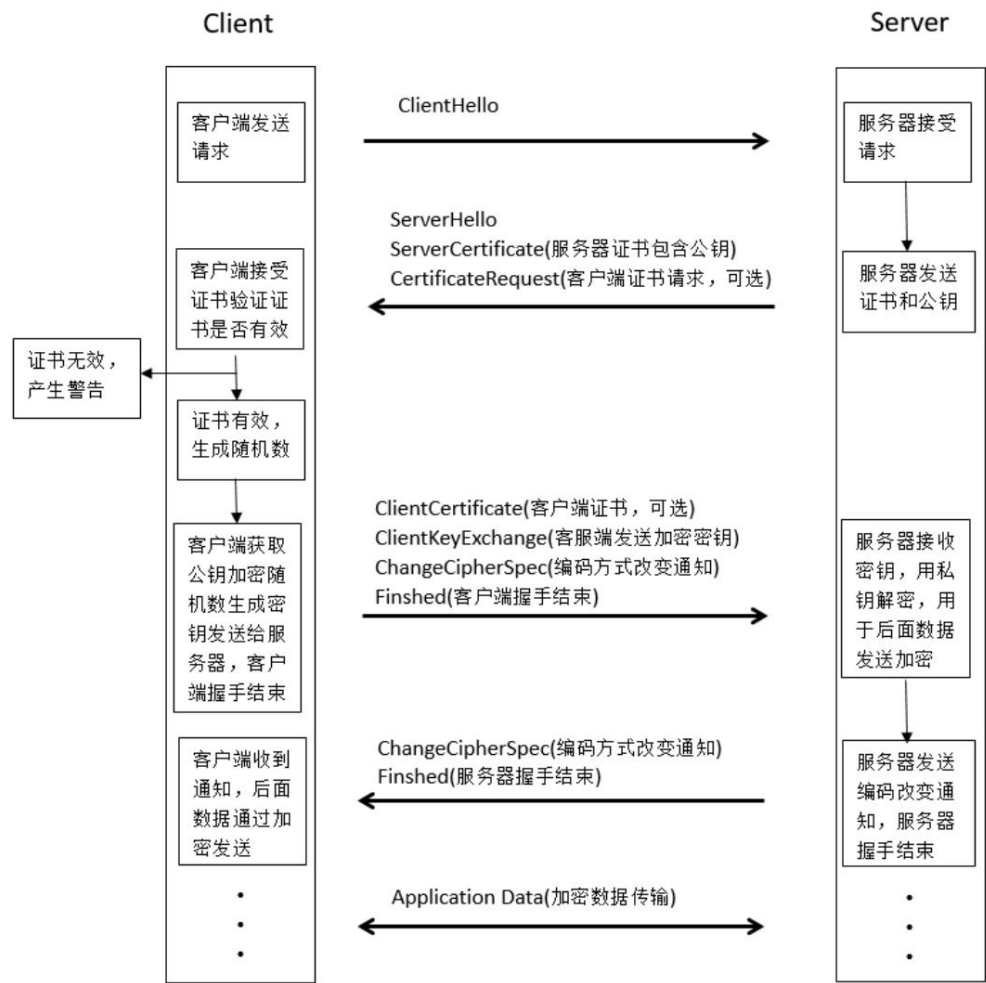


Figure 4.1: SSL/TLS Handshake interaction process

4.2 DTLS handshake process

To avoid denial of service attacks, DTLS uses the same stateless cookie technology as IKE.

After the client hello message, the server sends a HelloVerifyRequest message, which contains the stateless cookie.

After receiving it, the client must retransmit the clienthello with the cookie added.

The DTLS handshake process is shown in the following figure:

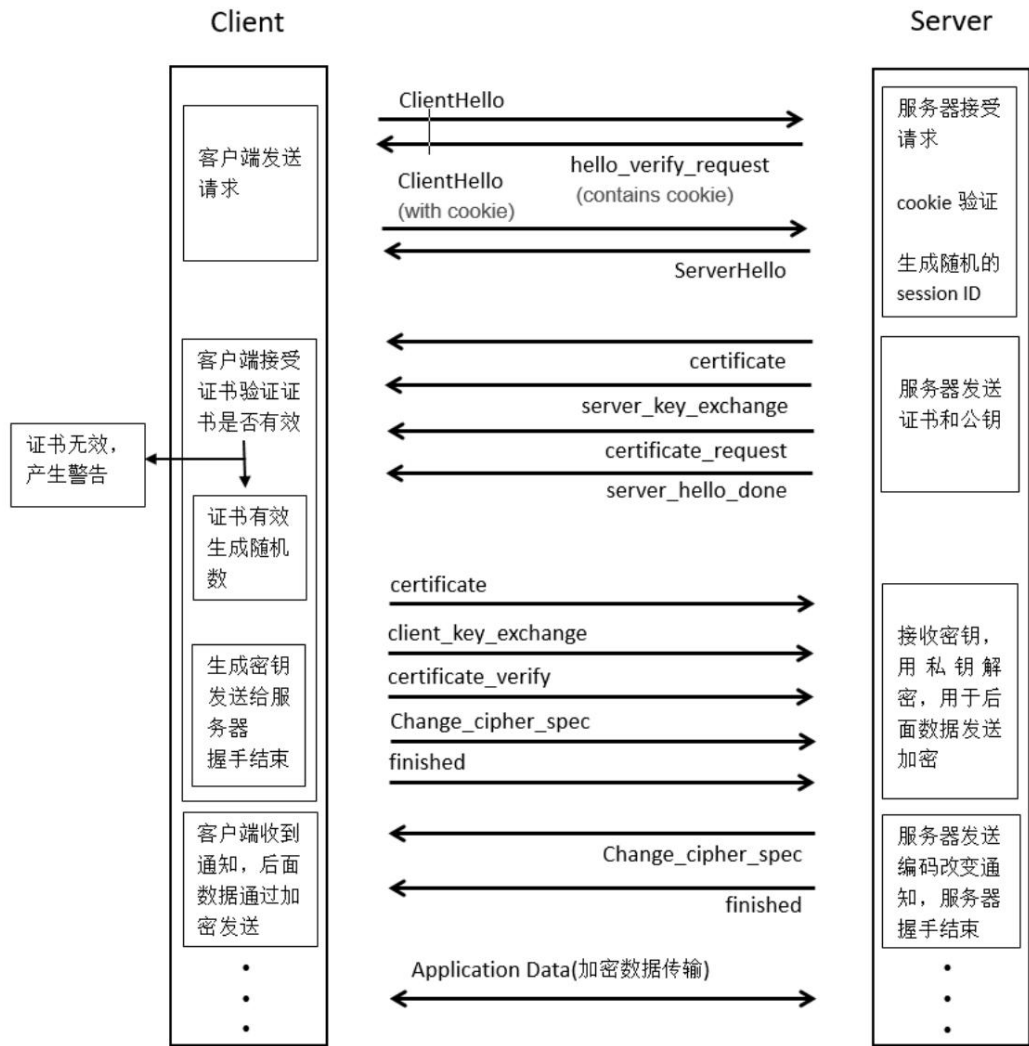


Figure 4.2: DTLS Handshake process

## Chapter 5

### user's guidance

This article mainly introduces the basic usage process of the mbedtls program, and focuses on the structures and important APIs are briefly described.

The basic workflow of mbedtls is as follows:

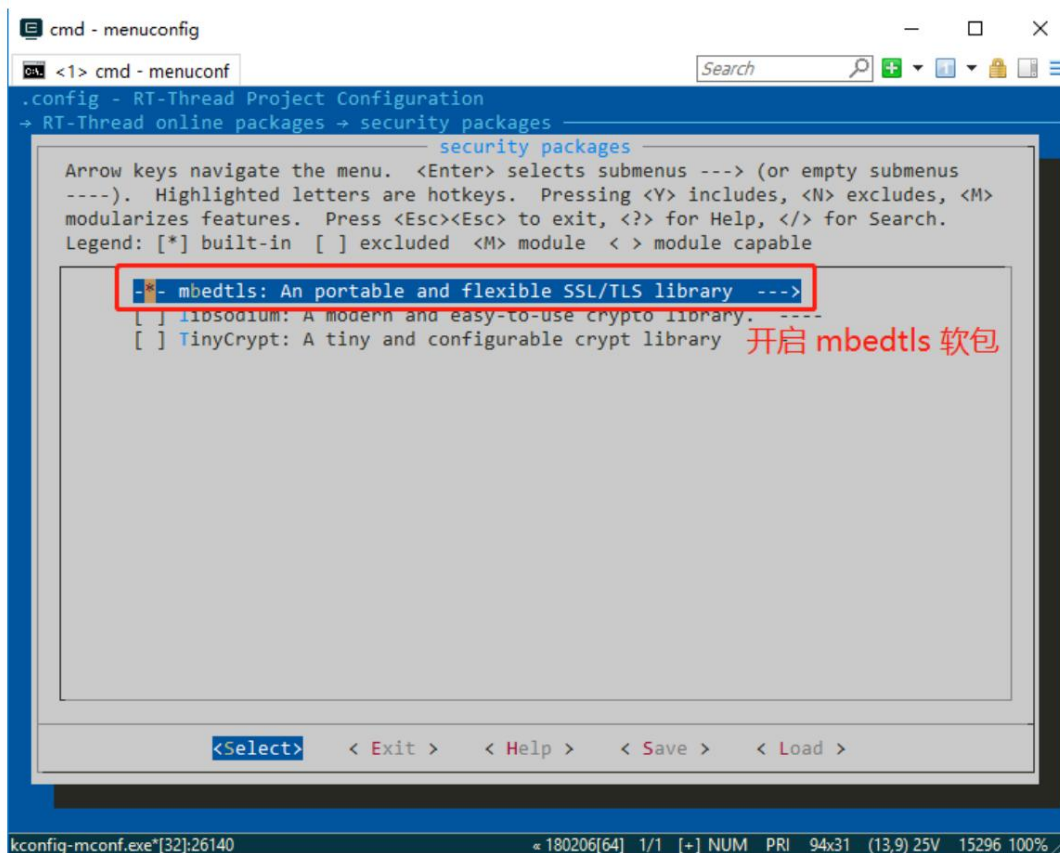
- Initialize SSL/TLS context • Establish SSL/TLS handshake • Send and receive data • Interaction completed, close the connection

#### 5.1 menuconfig configuration instructions

To obtain the mbedtls software package or modify user configuration, you need to use menuconfig . tool, switch the directory to the BSP directory you are using, and use the `menuconfig` command to open the configuration interface.

Select the **mbedtls** package in [RT-Thread online packages](#) > [security packages](#) . The operation interface is as shown below:



Figure 5.1: Open *mbedtls* Software Packages

The detailed configuration description is as follows:

RT-Thread online packages --->	
security packages --->	
Select Root Certificate --->	#Select the certificate file
[*] mbedtls: An portable and flexible SSL/TLS library #Open mbedtls	
Software Packages	
[*] Store the AES tables in ROM	
#Store the AES table in ROM	
(2) Maximum window size used	#Maximum "window" size for dot product (
2-7)	
(3584) Maxium fragment length in bytes	#Configure data frame size
[*] Enable a mbedtls client example # Enable mbedtls test example	
[ ] Enable Debug log output	
# Enable debug log output	
version (latest) --- >	#Select the software package version, the default is the latest version

- Using all default CA configuration option will add all pre-configured certificates in the `certs/default` directory to the compilation.

It will take up a lot of memory

- Using `user CA` configuration option allows users to add their own required certificate files to the compilation, requiring users to copy the certificate files to the `certs` root directory

After selecting the appropriate configuration item, use the `pkgs --update` command to download the software package and update the user configuration.

## 5.2 Function Configuration Description

The opening and closing of the mbedtls function module are defined in the `mbedtls/config.h` and `ports/inc/tls_config.h` files.

`mbedtls/config.h` is the configuration file provided in the mbedtls source code, and `ports/inc/tls_config.h` is the configuration file that RT-Thread cuts and adapts based on the configuration file in the mbedtls source code.

Finally, the user uses the configuration file **`ports/inc/tls_config.h` provided by RT-Thread.**

Users can use the macros in the file to enable or disable some unnecessary functional modules, thereby configuring mbedtls to an appropriate size.

## 5.3 Certificate Configuration Instructions

- The preset CA certificate files are stored in the `certs/default` directory
- The CA certificate files added by the user are stored in the `certs` root directory

The `certs/default` directory already contains most CA root certificates. If the root certificate you use is not in this folder, you need to copy your own CA certificate in **PEM** format to the `certs` root directory. (Only **PEM** format certificates are supported, not **DER** format certificates).

This certificate file already contains most of the CA root certificates. Please refer to the [Add New Certificate](#) section below.

## 5.4 Initialize TLS session

```
typedef struct MbedTLSSession {

    char* host;
    char* port;

    unsigned char *buffer; size_t          // Public data buffer //
    buffer_len;                          Buffer size

    mbedtls_ssl_context ssl;              //Save SSL basic data //Save
    mbedtls_ssl_config conf;              SSL configuration information
}
```

```

    mbedtls_entropy_context entropy;           //Save ssl entropy configuration
    mbedtls_ctr_drbg_context ctr_drbg; // Save random byte generator configuration
    mbedtls_net_context server_fd;           //Save file descriptor//Save
    mbedtls_x509_crt cacert;                 authentication information
} MbedTLSSession;

```

MbedTLSSession is used to save the configuration information when establishing a TLS session connection and pass it in the TLS context.

Before establishing a TLS session, the user must define a structure to store the session content, as shown below:

```

static MbedTLSSession *tls_session = RT_NULL; tls_session =
(MbedTLSSession *)malloc(sizeof(MbedTLSSession));

tls_session->host = strdup(MBEDTLS_WEB_SERVER); tls_session-
>port = strdup(MBEDTLS_WEB_PORT); tls_session->buffer_len
= MBEDTLS_READ_BUFFER; tls_session->buffer =
malloc(tls_session->buffer_len);

```

Here you need to set the host and port of the SSL/TLS server, as well as the data receiving buffer and other configurations.

## 5.5 Initializing the SSL/TLS Client

The application initializes the TLS client using the `mbedtls_client_init` function.

In the initialization phase, you can pass in relevant parameters according to the API parameter definition, which is mainly used to initialize the network interface, certificate, SSL session configuration and other necessary configurations for SSL interaction, as well as setting related callback functions.

The sample code is as follows:

```

char *pers = "hello_world"; // Set random string seed if((ret =
mbedtls_client_init(tls_session, (void *)pers, strlen(pers)))
    != 0)
{
    rt_kprintf("MbedTLSClientInit err return : -0x%x\n", -ret); goto __exit;
}

```

The mbedtls library function actually called is as follows:

## 5.6 Initializing the SSL/TLS Client Context

The application uses the `mbedtls_client_context` function to configure the client context information, including certificate parsing, setting the host name, setting the default SSL configuration, and setting the authentication mode (MBEDTLS\_SSL\_VERIFY\_OPTIONAL by default).

wait.

The sample code is as follows:

```
if((ret = mbedtls_client_context(tls_session)) < 0) {

    rt_kprintf("MbedTLSClientContext err return : -0x%x\n", -ret); goto __exit;

}
```

## 5.7 Establishing an SSL/TLS connection

Use the `mbedtls_client_connect` function to establish a channel for the SSL/TLS connection. This includes the entire handshake Connection process and certificate verification results.

The sample code is as follows:

```
if((ret = mbedtls_client_connect(tls_session)) != 0) {

    rt_kprintf("MbedTLSClientConnect err return : -0x%x\n", -ret);
    goto __exit;

}
```

## 5.8 Reading and Writing Data

Writing data to **SSL/TLS**

The sample code is as follows:

```
static const char *REQUEST = "GET https://www.howsmyssl.com/a/check HTTP
/1.0\r\n"
"Host: www.howsmyssl.com\r\n"
"User-Agent: rtthread/3.1 rt\r\n" "\r\n";

while((ret = mbedtls_client_write(tls_session, (const unsigned char *)
REQUEST, strlen(REQUEST))) <= 0)
{
    if(ret != MBEDTLS_ERR_SSL_WANT_READ && ret !=
MBEDTLS_ERR_SSL_WANT_WRITE)
```

```

    {
        rt_kprintf("mbedtls_ssl_write returned -0x%x\n", -ret); goto __exit;
    }
}

```

Reading data from **SSL/TLS**

The sample code is as follows:

```

memset(tls_session->buffer, 0x00, tls_session->buffer_len); ret =
mbedtls_client_read(tls_session, (unsigned char *)tls_session->
    buffer, len);
if(ret == MBEDTLS_ERR_SSL_WANT_READ || ret == MBEDTLS_ERR_SSL_WANT_WRITE)
    continue;

if(ret == MBEDTLS_ERR_SSL_PEER_CLOSE_NOTIFY)
    break;
if(ret < 0) {

    rt_kprintf("mbedtls_ssl_read returned -0x%x\n", -ret); break;

} if(ret == 0) {

    rt_kprintf("connection closed\n"); break;

}

```

Note that if the read or write interface returns an error, the connection must be closed.

## 5.9 Closing SSL/TLS Client Connections

When the client actively closes the connection or closes the connection due to an abnormal error, you need to use `mbedtls_client_close` to close the connection and release resources.

The sample code is as follows:

```

mbedtls_client_close(tls_session);

```

## 5.10 mbedTLS usage paradigm

Refer to the sample program `samples/tls_app_test.c`.

## 5.11 Adding a new certificate

There are two common formats for CA certificates: **PEM** format and **DER** format. Currently, RT-Thread mbedTLS only supports The certificate file is in **PEM** format.

- Certificate in PEM format

**PEM** format certificates usually end with `.pem` and `.cer` suffixes.

After opening it with a text editor, the file content starts with `-----BEGIN CERTIFICATE-----` and ends with `-----END CERTIFICATE-----`.

- DER format certificate

**DER** format certificates are binary file types.

### 5.11.1 Root Certificate Style

Double-click the CA file ending with `.cer` (Windows system) to see the issuing authority and validity period of the certificate.

As shown below:



Figure 5.2: Root certificate information

The content of the certificate file in **PEM** format is as follows:

```
-----BEGIN CERTIFICATE-----
MIIDdTCCA12gAwIBAgILBAAAAAABFUtaW5QwDQYJKoZIhvcNAQEFBQAwVzELMAkG
A1UEBhMCQkUxGTAXBgNVBAoTEEdsb2JhbFNpZ24gYm9vdCBkQTAeFw05ODA5MDExMjAw
b3QgQ0ExGzAZBgNVBAMTEkdsb2JhbFNpZ24gUm9vdCBkQTAeFw05ODA5MDExMjAw
MDBaFw0yODAxMjg0MjAwMDBaMFcxZzAJBgNVBAYTAkFMRkwFwYDVQQKExBHbG9i
YWxTaWduIG52LXNhMRAwDgYDVQQLEwdSb290IENBMRSwGQQYDVQQDExJHbG9iYWxT
aWduIFJvb3QgQ0ExGzAZBgNVBAMTEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoI
BAQDADuaZjc6j40+Kfvvxi4Mla+plH/EqsLmVEQS98GPR4mdmxzdxtlK+6NiY6arym
AZavp xy0Sy6scTHAHoT0KMM0VjU/43dSMUBUc71DuxC73/
OIS8pF94G3VNTCOXkNz8kHp
1Wrjsok6Vjk4bwY8iGlbKk3Fp1S4bInMm/k8yuX9ifUSPJJ4ltbcdG6TRGHRjcdG
snUOhugZitVtbNV4FpWi6cgKOOvyJBNCp1STE4U6G7weNLWLBYY5d4ux2x8kasJ
U26Qzns3dLlwR5EiUWMWea6xrEmCMgZK9FGqjWZCrXgzT/LCrBbBIDSgeF59N8 9i
Fo7+ryUp9/k5DPAgMBAAGjQjBAMA4GA1UdDwEB/wQEAwIBBjAPBgNVHRMBAf8E
BTADAQH/MB0GA1UdDgQWBBrge2YaRQ2XyolQL30EzTS0//z9SzaNBgkqhkiG9w0B
AQUFAAOCAQEA1nPfE920I2/7LqivjTFKDK1fPxsCwrvQmeU79rXqoRSLbICKOz
```

```
yj1hTdNGCbM+w6DjY1Ub8rrvrTnhQ7k4o+YviiY776BQVvnGCv04zcQLcFGUI5gE
38NfiNUVyRRBnMRddWQVdf9VMOyGj/8N7yy5Y0b2qvzfGn9LhJlZJrglfCm7ymP
AbEVtQwdpf5pLGkkeB6zpxxxYu7KyJesF12KwvhHhm4qxFYxldBniYUr+WymXUad
DKqC5JIR3XC321Y9YeRq4VzW9v493kHMB65jUr9TU/Qr6cf9tveCX4XSQRjbgbME
HMUfplBvFSDJ3gyICh3WZIXi/EjKSp4A==
-----END CERTIFICATE-----
```

### 5.11.2 Obtaining the root certificate

- Request directly from the service provider

Request a **base64** -encoded **X.509** -encoded **PEM**- format certificate file from the service provider .

- Export from service provider website

– Open the service provider's website in the browser, take <https://www.rt-thread.org/> as an example – Click Security in the browser address bar , then click Certificate

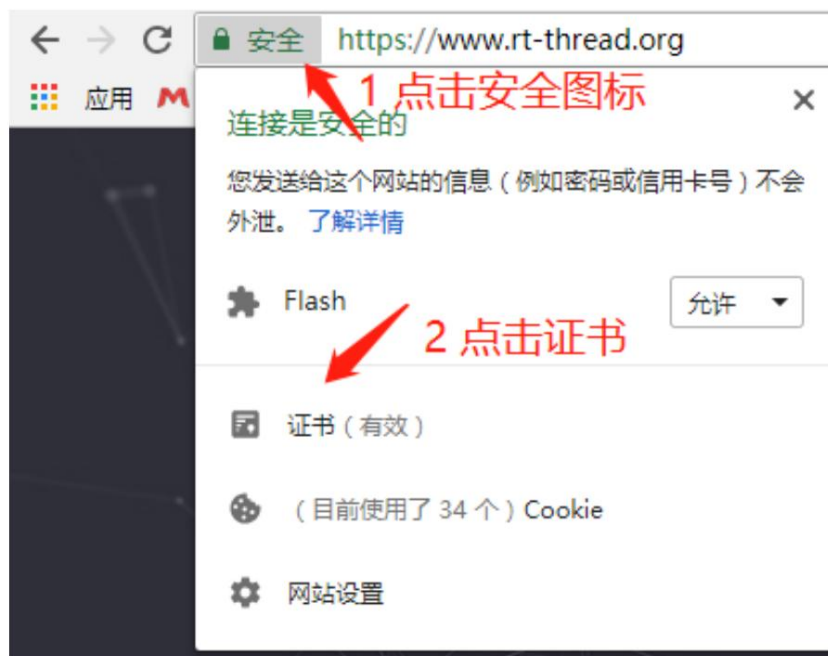


Figure 5.3: Obtaining the website root certificate

– View certificate details



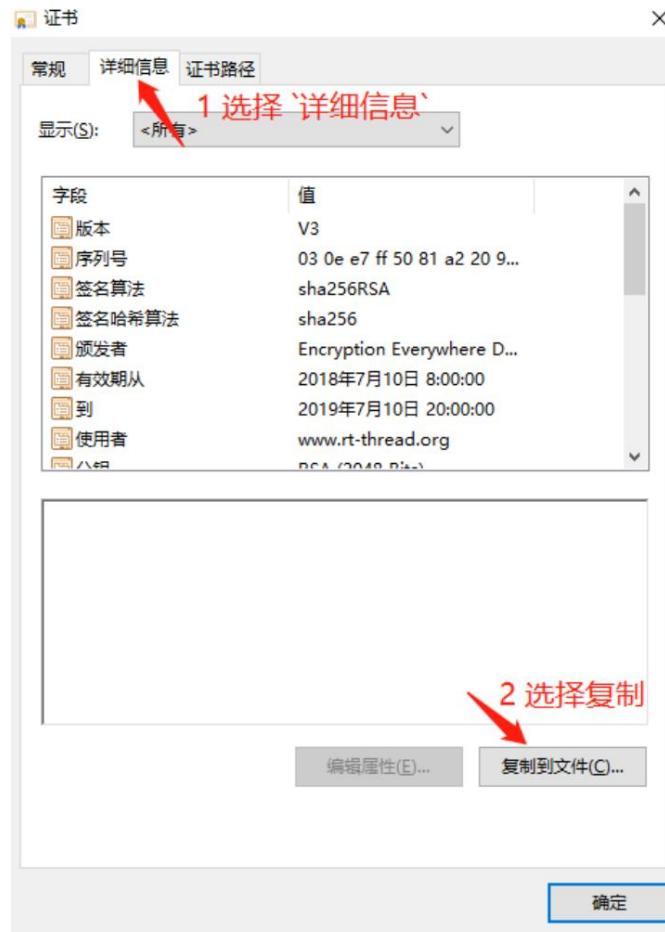


Figure 5.4: View certificate details

– Root Certificate Export Wizard



Figure 5.5: Export Root Certificate Wizard

- Select Export Base64 Encoded Certificate

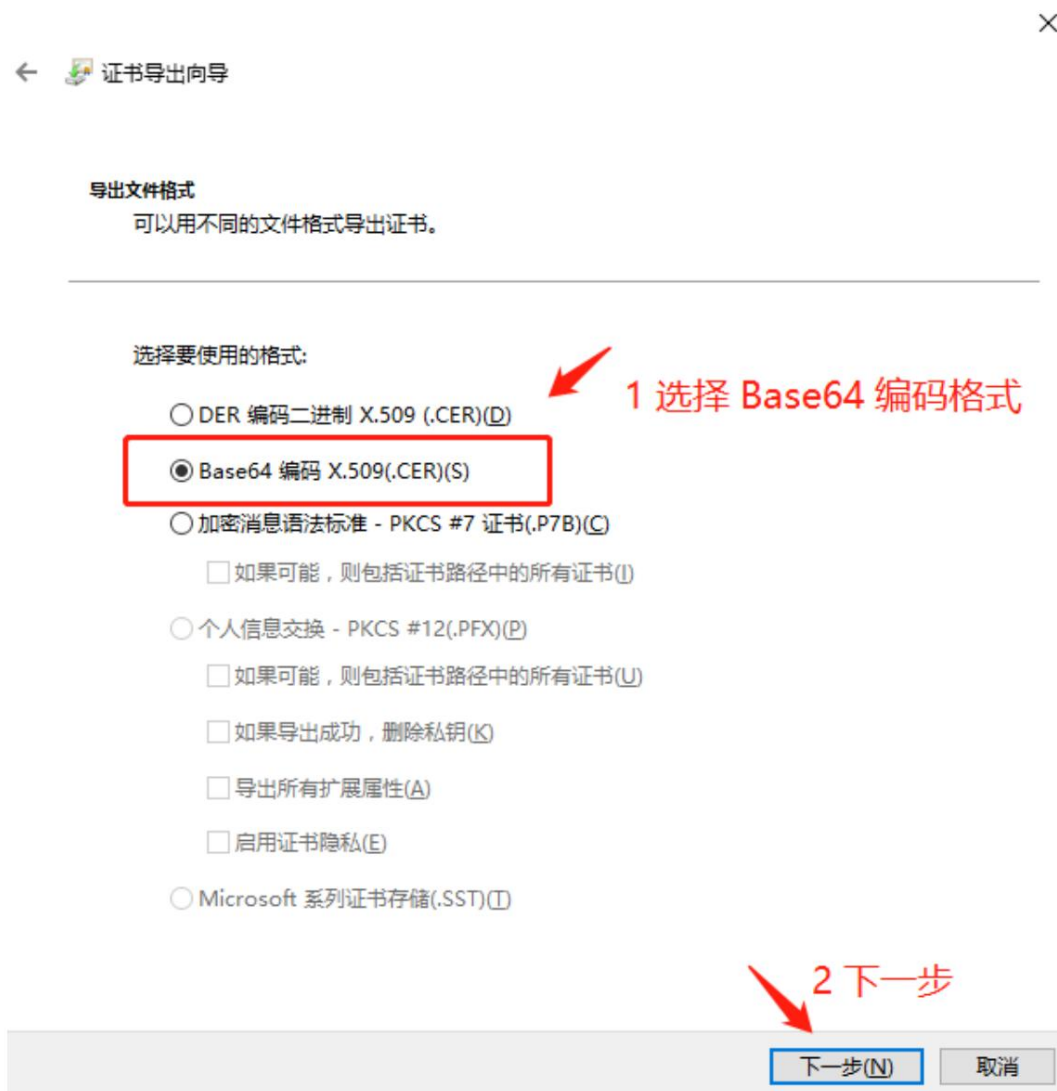


Figure 5.6: Select the root certificate encoding format

– Select the certificate storage location



Figure 5.7: Select the root certificate store location

– Complete the certificate file export

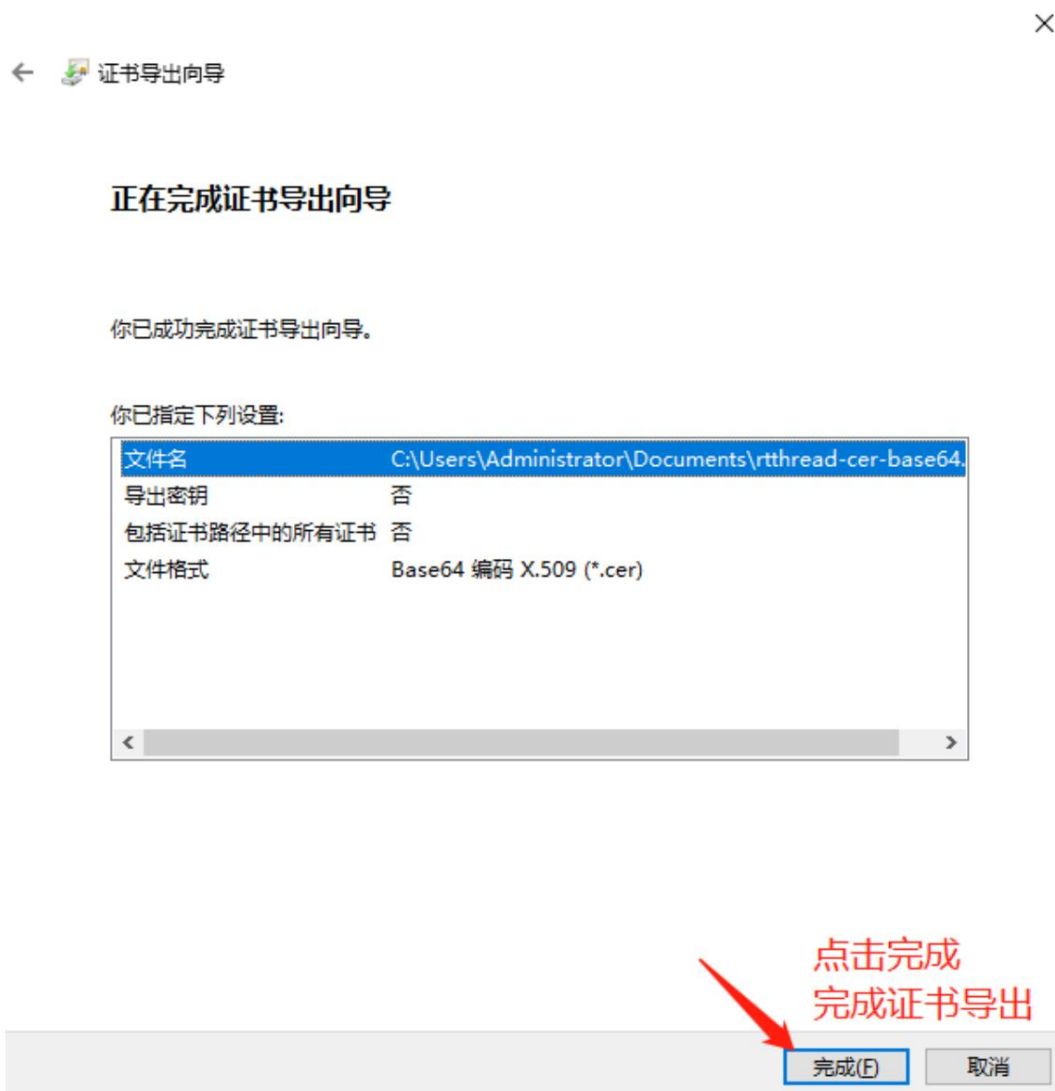


Figure 5.8: Complete the root certificate export

The certificate export is completed. Assume that the certificate file name is **USER\_ROOT\_CA.cer**.

### 5.11.3 Importing Certificates

- Use a text editor to open the root certificate file **USER\_ROOT\_CA.cer** exported in the previous step
- Copy the **USER\_ROOT\_CA.cer** file to the [certs](#) root directory
- Recompile using [scons](#) command

Note:

After the [scons](#) command is compiled, the certificate file will be automatically copied to `const char mbedtls_root_certificate[]`

In the array.

## 5.12 Frequently Asked Questions

### 5.12.1 Certificate Verification Failed

[tls]verification info: ! The CRL is not correctly **signed** by the trusted CA

- reason

The mbedtls package supports multiple mainstream CA root certificates, but some CAs are not supported.

- Solution

If the test fails for other TLS website certificates, manually obtain the root certificate of the test website and add it to the mbedtls/tls\_certificate.c file

### 5.12.2 Certificate time error

[tls]verify peer certificate fail.... [tls]verification info:  
! The certificate validity starts in the future

- reason

TLS handshake is a certificate verification that takes time. The local time is incorrectly obtained.

- Solution

Check whether the RTC device is supported, check whether the `RT_USING_RTC` macro is enabled, and calibrate the device time. It is recommended to use NTP to synchronize the local time.

### 5.12.3 Certificate CN Error

verification info: ! The certificate Common Name (CN) does not match with the expected CN

- reason

When testing other TLS websites, if the domain name entered does not match the Common Name (CN) of the certificate, CN verification fails.

- Solution

Check whether the input domain name matches the CN in the certificate or input IP address

## 5.12.4 0x7200 Error

- reason

This is partly because mbedTLS receives packets larger than the buffer size.

- Solution

`menuconfig` configuration increases the data frame size ( Maximum fragment length in bytes )

```
RT-Thread online packages --->
  security packages --->
    Select Root Certificate --->                                #Select the certificate file
    [*] mbedtls: An portable and flexible SSL/TLS library ---
    [*] Store the AES tables in ROM
    (2) Maximum window size used
    (6144) Maximum fragment length in bytes #Configure data frame size (0x7200
        If errors occur, try increasing the size.)
    [*] Enable a mbedtls client example version (latest) ---
    >
```

## 5.13 References

- mbedTLS official website: <https://tls.mbed.org/>
- ARMmbed [GitHub: mbedtls](#)

## Chapter 6

# MbedTLS RAM and ROM resource usage optimization guide

The **mbedtls** software package adopts a modular design, and the `config.h` file can be used to configure and select functional modules.

The default `config.h` file provided by **mbedtls** is a general, full-featured configuration file that occupies a large amount of RAM and ROM space, but guarantees the speed, stability, protocol compatibility, and data transmission efficiency of SSL handshake and communication. However, embedded devices are limited by their limited RAM and ROM space, so we have to sacrifice speed to save RAM space and cut unnecessary functional modules to reduce ROM occupancy.

This optimization guide optimizes RAM while ensuring that the SSL/TLS client can establish a secure and stable connection with the server. And ROM occupancy for optimization statistics.

Notice:

The optimization of **mbedtls** client is targeted optimization, which is optimized for specific SSL/TLS servers. Different SSL/TLS servers have different configurations, and the configuration parameters used for optimization are also different.

Therefore, before optimizing SSL/TLS, developers should first use the default configuration to adjust the SSL/TLS handshake connection and encrypted communication if MCU resources allow, and then optimize each item according to the specific configuration of the SSL/TLS server.

Of course, in most cases you don't know the specific parameter configuration of the server, so you can only optimize it tentatively. Descriptions of each configuration are provided to facilitate developers to perform targeted optimization.

### 6.1 Optimization Description

- RAM resource usage statistics

First, ensure that the SSL handshake connection is normal and the encrypted data communication is normal. Run the `tls_test` test routine to perform a RAM optimization test. The test routine runs in a separate thread and determines the RAM used during the handshake communication process by comparing the memory occupied before and after the SSL handshake is successful. This test method can only roughly estimate the SSL client



The RAM size required by the client to successfully complete the handshake connection. This data includes the additional RAM space required to ensure handshake communication.

- ROM resource usage statistics

The ROM size occupied by **mbedtls** is calculated by comparing the files involved in the link before and after starting the **mbedtls** functional component.

- Test platform: iMXRT1052
- Test IDE: MDK5
- Optimization level: o2
- Test routine: [samples/tls\\_app\\_test.c](#)
- SSL server used for testing : [www.rt-thread.org](http://www.rt-thread.org)
- Test server root certificate signature algorithm: sha1RSA
- Test server root certificate signature hash algorithm: sha1
- Test server root certificate public key: RSA 2048 [bits](#)
- Test server root certificate fingerprint algorithm: sha1
- SSL client-specified cipher suites

```
#define MBEDTLS_SSL_CIPHERSUITES \
    MBEDTLS_TLS_RSA_WITH_AES_256_CBC_SHA256
```

- SSL client specifies the frame size as `##define MBEDTLS_SSL_MAX_CONTENT_LEN 3584` • Configuration used in the test (see the end of the article for details)

## 6.2 Summary of resource usage after optimization

- Default `tls_config.h` configuration resource usage

The default configuration file of **mbedtls** is `mbedtls/include/mbedtls/config.h`, while the configuration file used by **RT-Thread** is `ports/inc/tls_config.h`. Users also use the `ports/inc/tls_config.h` file when optimizing the configuration .

```
RO(CODE + RO)      : 159828 bytes (156.08K) 720 bytes
RW(RW + ZI)        :
ROM (CODE + RO + RW) : 159972 bytes (156.22K) Dynamic
memory usage : 26849 bytes (26.22K ) (including 1K test buffer)
```

- Optimized configuration resource usage

```

RO(CODE + RO)          : 71893 bytes (70.21K)
RW(RW + ZI)            :      82 bytes
ROM(CODE + RO + RW) : 71975 bytes (70.29K)
Dynamic memory usage : 23344 bytes (22.79K ) (including 1K test buffer)

```

### 6.3 Preparation before optimization

1. First you need to have an SSL server ready to access (make sure it works properly)
2. Prepare the PEM root certificate file for accessing the SSL server (stored in the `certs` directory of the mbedtls package)
  - Delete other unnecessary certificates)
3. Successfully connect to the SSL server using the default mbedtls configuration file (it is more difficult to locate the cause of failure after optimization)
4. Optimize mbedtls configuration item by item and test repeatedly

Notice:

If your MCU resources are limited and the default `tls_config.h` configuration file cannot be used, developers can choose to use QEMU virtual machine for development and debugging and mbedtls optimization. Optimize mbedtls resource usage to an appropriate time. Then use the MCU you need for verification testing.

### 6.4 Optimization Configuration Overview

#### 6.4.1 Commonly used optimization configurations

By modifying the configuration in the following list, you can greatly reduce the RAM and ROM usage of mbedtls.

use.

When optimizing, developers are advised to optimize the configurations in the following list first. If they cannot meet the requirements, Optimize other configurations one by one.

Configuration	rely	illustrate	Optimization suggestions
const char mbedtls_root_certificate[]	none	A constant array for storing root certificates. When compiling, PEM certificates will be added to this array. It is recommended to only store the required root certificate files in the <a href="#">certs</a> directory, otherwise it will take up a lot of RAM and ROM space.	Only store the required certificate files
MBEDTLS_SSL_CIPHERSUITES	none	By specifying a cipher suite, you can save several hundred bytes of ROM and RAM. Note that you need to specify the cipher suite that the server supports, enable the relevant functional components for the cipher suite, and disable other functional components. If you only connect to one SSL server, you usually only need to define support for one cipher suite.	Specify only the cipher suites required by the root certificate
MBEDTLS_AES_ROM_TABLES	none	Store the AES table in ROM to save RAM usage (greatly reduces RAM usage)	Recommended to enable
MBEDTLS_SSL_MAX_CONTENT_LENGTH	none	The default is 16384. RFC defines the default size of SSL/TLS messages. If you change the value here, other clients/servers may no longer be able to communicate with you. Unless you can determine the frame size on the server side, make appropriate changes based on the maximum frame size sent by the server.	Reduce appropriately (please increase this configuration if 0x7200 error occurs)

Configuration	rely	illustrate	Optimization suggestions
MBEDTLS_MPI_MAX_SIZE None		Maximum MPI word size available  The default value is 1024.  Can be adjusted appropriately	Appropriately reduce
MBEDTLS_MPI_WINDOW_SIZE	none	MPI for modular exponentiation  The maximum number of windows, by default  Consider 6, and select the value range:  1-6, can be adjusted down appropriately	Appropriately reduce
MBEDTLS_ECP_MAX_BITS None		GF(p) Elliptic Curve  Large bit, default is 521	Appropriately reduce
MBEDTLS_ECP_WINDOW_SIZE	none	Maximum window for dot product  The default value is 6.  Value range: 2-7, optional  To appropriately reduce  Will affect speed	Appropriately reduce
MBEDTLS_ECP_FIXED_POINT_OPTIM No speed. When enabled, it will speed up		Default is 1, enabling fixed-point addition  Fast dot multiplication operation is about 3  to 4 times, the cost is peak  The memory usage of the value increases by about  2 times. Can be configured as  0, sacrifice speed to save  RAM usage	Can be optimized to 0
MBEDTLS_ECP_NIST_OPTIM	none	Enabled for each NIST  Specific examples,  Fast operation on the curve  4 to 8 times, the disadvantage is  The ROM space is large.  Selective Optimization	Can be disabled
MBEDTLS_ENTROPY_MAX_SOURCES minimum is 2, used by default		The maximum number of entropy sources,    <a href="#">mbedtls_platform_entropy_poll</a>  Source: RT-Thread  Using minimal configuration 2	Can be optimized to 2

### 6.4.2 System related configuration

This part of the configuration is related to the specific system and compiler. The following table lists the configuration required on **RT-Thread**.

Configuration	rely	illustrate	Optimization suggestions
MBEDTLS_HAVE_ASM None		The compiler needs to be able to handle Assembly code	Enable
MBEDTLS_HAVE_TIME None		If your system does not have time.h and time() function Please comment this configuration	Enable
MBEDTLS_HAVE_TIME_DATE none		If your system does not have time.h time.h time() or None Correct clock, please note Explain the configuration	Enable
MBEDTLS_DEBUG_C None		Define this configuration to start Debug log output	If you need to debug the log, enable Otherwise, please disable
MBEDTLS_NET_C None		This configuration only supports POSIX/Unix and Windows system, in On RT-Thread system Need to be closed	Disable
MBEDTLS_NO_PLATFORM_ENTROPY None	If your platform does not support	or Windows CryptoAPI and other standards, You need to enable this configuration. Required on RT-Thread Enable	Enable
MBEDTLS_TIMING_C None		If annotated, required Users implement related Function. Enabled by default	Enable
MBEDTLS_TIMING_ALT None		If annotated, required Users implement related Function. Enabled by default	Enable
MBEDTLS_ENTROPY_HARDWARE_ALT None	Not to use sample related	Function. Enabled by default	Enable
MBEDTLS_ENTROPY_C MBEDTLS_SHA256_C or MBEDTLS_SHA512_C		Enable platform-specific Entropy code. Need to be enabled	Enable
MBEDTLS_PADLOCK_C MBEDTLS_HAVE_ASM Enable VIA on x86		Padlock Support	Disable

Configuration	rely	illustrate	Optimization suggestions
MBEDTLS_AESNI_CMBEDTLS_HAVE_ASM is enabled on x86-64			Disable
AES-NI support			
MBEDTLS_PLATFORM_C None		Enables the platform abstraction layer, To redefine the implementation Functions such as free and printf	Enable

### 6.4.3 Functional component related configuration

Users can choose which one to enable based on the SSL/TLS server features to be accessed and the signature algorithm used by the root certificate. When enabling or disabling a functional component, please be sure to enable or disable related dependencies.

Configuration	rely	illustrate	Optimization suggestions
MBEDTLS_ASN1_PARSE_C None		Enable generic ASN1 Parser. ASN1: A method for describing digital objects Methods and standards, need Enable	Enable
MBEDTLS_ASN1_WRITE_C None		Enable Generic ASN1 Encoding Writer	Enable
MBEDTLS_BIGNUM_C None		Enable the big integer library (multi-precision integer library	Enable
MBEDTLS_CIPHER_C None		Enable Common Password Layer Enable	
MBEDTLS_AES_C None		Enable AES encryption. PEM_PARSE Usage AES to decrypt encrypted By enabling AES to support <a href="#">*_WITH_AES_*</a> Types Cipher suite	Enable
MBEDTLS_CTR_DRBG_C	MBEDTLS_AES_C enables	CTR_DRBG AES-256 random generation Success	Enable
MBEDTLS_MD_C None		Enable Generic Message Digest layer, needs to be enabled	Enable

Configuration	rely	illustrate	Optimization suggestions
MBEDTLS_OID_C None		Enable OID database. This module converts between OID and internal values and needs to be enabled.	Enable
MBEDTLS_PK_C MBEDTLS_RSA_C, symmetric) key layer, needs to be MBEDTLS_ECP_C		Enable common public (non enabled	Enable
MBEDTLS_PK_PARSE_C MBEDTLS_PK_C	Enables generic public (asymmetric) key parser, needs to be enabled		Enable
MBEDTLS_SHA256_C None		Enable SHA-224 and SHA-256 cryptographic hash algorithm, selected based on <a href="#">the signature hash algorithm</a> in the root certificate details	Select as needed
MBEDTLS_SHA512_C None		Enable SHA-384 and SHA-512 cryptographic hash algorithm, selected based on <a href="#">the signature hash algorithm</a> in the root certificate details	Select as needed
MBEDTLS_SSL_CLI_C MBEDTLS_SSL_TLS_C	enables SSL client code. It does not need to be enabled when used as an SSL server.		Enable
MBEDTLS_SSL_SRV_C MBEDTLS_SSL_TLS_C	enables the SSL server code. It does not need to be enabled when used as an SSL client.		Disable
MBEDTLS_SSL_TLS_C MBEDTLS_CIPHER_C, MBEDTLS_MD_C and define at least one MBEDTLS_SSL_PROTO_XXX		Enable SSL/TLS code enablement	
MBEDTLS_X509_CRT_PARSE_C MBEDTLS_X509_USE_C	Enable X509 certificate decryption		Enable

Analysis

Configuration	rely	illustrate	Optimization suggestions
	MBEDTLS_X509_USE_C MBEDTLS_ASN1_PARSE_C, MBEDTLS_BIGNUM_C, MBEDTLS_OID_C, MBEDTLS_PK_PARSE_C	Enable the X.509 core to Using Certificates	Enable
	MBEDTLS_BASE64_C None	Enable the base64 component, PEM certificate parsing requires use	Enable
	MBEDTLS_CERTS_C None	This module is used for testing SSL Clients and Services You can choose to disable	Can be disabled
	MBEDTLS_PEM_PARSE_C depends on MBEDTLS_BASE64_C enables support for PEM files parsing support		Enable
	MBEDTLS_RSA_C MBEDTLS_BIGNUM_C, MBEDTLS_OID_C	Enable RSA public key cryptography System. RSA, DHE-RSA, ECDHE-RSA, RSA-PSK encryption Key exchange requires the use of	Enable
	MBEDTLS_SHA1_C None	Enable SHA1 encryption TLS 1.1/1.2 Required	Enable
	MBEDTLS_MD5_C None	Enable MD5 hashing PEM parsing requires use	Enable
	MBEDTLS_PK_PARSE_EC_EXTENDED and MBEDTLS_EC_C	ECDSA and ECDSA SEC1 changes not allowed EC readout Key support	Can be disabled
	MBEDTLS_ERROR_STRERROR_NOMIME is not available,		Can be disabled
		MBEDTLS_ERROR_C It is easier to use third party Use in library mbedtls_strerror ( ) (Enable MBEDTLS_ERROR_C Invalid at this time)	



Configuration	rely	illustrate	Optimization suggestions
MBEDTLS_GENPRIME MBEDTLS_BIGNUM_C Enable prime number generation code can be disabled			
MBEDTLS_FS_IO None		Enable file system interaction related functions	Can be disabled
MBEDTLS_PKCS5_C MBEDTLS_MD_C		This module adds support for PKCS#5 functionality.	Select as needed
		Required for AES algorithm data padding scheme.	
		Choose whether to disable as needed	
MBEDTLS_PKCS12_C MBEDTLS_ASN1_PARSE_C, MBEDTLS_CIPHER_C, key algorithm MBEDTLS_MD_C		Add for parsing PKCS #8 Encrypted Private Key	Can be disabled
MBEDTLS_PKCS1_V15 MBEDTLS_RSA_C		is used to support PKCS#1 v1.5 operations, which is required for RSA key suites. If RSA key suites are used, you need to enable	Select as needed
MBEDTLS_PKCS1_V21 MBEDTLS_MD_C, MBEDTLS_RSA_C		Enables support for PKCS#1 v2.1 encoding, which allows RSAES-OAEP and RSASSA-PSS Operations	Can be disabled
MBEDTLS_PK_RSA_ALT_SUPPORT		Support external private RSA keys in the PK layer (e.g. from an HSM). Not required, disable	Disable
MBEDTLS_SELF_TEST None		Enable the inspection function. It is recommended to enable it when debugging is enabled, and disable it at other times.	Can be disabled
MBEDTLS_SSL_ALL_ALERT_MESSAGES		Enable alert message	Can be disabled
MBEDTLS_SSL_ENCRYPT_THEN_MAC		None. Does support MAC, RFC 7366. Used to enhance protection for CBC cipher suites, can be disabled	Can be disabled

Configuration	rely	illustrate	Optimization suggestions	
MBEDTLS_SSL_EXTENDED_MASTER_SECRET	enable	Enable support for extended master key	Can be disabled	
MBEDTLS_SSL_FALLBACK_SCSV	none	Comment this macro to disable the Client uses fallback strategy	Can be disabled	
MBEDTLS_SSL_CBC_RECORD_SPLITTING	enable	enable 1.0 is CBC mode	In SSLv3 and TLS	Can be disabled
		Enable 1/n-1 record splitting		
		Enable this macro to		
		Low BEAST attack		
		Risk, selectivity		
		Disable		
MBEDTLS_SSL_RENEGOTIATION	none	Disable this macro to TLS renegotiation support	Disable	
		Enabling this may cause		
		To safety risk, it is recommended		
		Disable		
MBEDTLS_SSL_MAX_FRAGMENT_LENGTH	enable	Enable support for SSL RFC 6066 Maximum Frame Length	Can be disabled	
		Degree extension support		
MBEDTLS_SSL_ALPN	None	Enable support for RFC 7301 Application layer protocol negotiation support	Can be disabled	
MBEDTLS_SSL_SESSION_TICKETS	none	Enable support for SSL RFC 5077 Sessions Tickets support, required	Can be disabled	
		The server needs to support it.		
		Often used to optimize handshake flow		
		Provision		
MBEDTLS_SSL_EXPORT_KEYS	none	Enable export of key blocks and master key support.	Can be disabled	
		This is useful for some TLS		
		User is required, for example		
		EAP-TLS		
MBEDTLS_SSL_SERVER_NAME_INDICATION	enable	Enable support for SSL RFC 6066 Server Name Support for the SNI	Can be disabled	
		hold		

Configuration	rely	illustrate	Optimization suggestions
MBEDTLS_SSL_TRUNCATED_HMACnone		Enable support for SSL RFC 6066 Truncation HMAC support	Can be disabled
MBEDTLS_VERSION_FEATURES	MBEDTLS_VERSION_C	Version feature information related to can be disabled	
MBEDTLS_VERSION_C None		This module provides the runtime Version Information	Can be disabled
MBEDTLS_X509_CHECK_KEY_USAGE_C	MBEDTLS_X509_CHECK_KEY_USAGE_C	Verify, Disable this feature Avoids incorrect publishing and/or misuse (intermediate) CA and leaf certificate issues Skip after comment keyUsage Check CA Certificate of leaf	Can be disabled
MBEDTLS_X509_CHECK_EXTENDED_KEY_USAGE_C	MBEDTLS_X509_CHECK_EXTENDED_KEY_USAGE_C	Enable Extension (leaf certificate) verification Disabling this feature can Avoid false postings and/or The problem of misused certificates	Can be disabled
MBEDTLS_X509_RSASSA_PSS_SUPPORT_C	MBEDTLS_X509_RSASSA_PSS_SUPPORT_C	Known as PKCS #1 v2.1) X.509 signed Certificates, CRLs and CSRS analysis and verification Choose according to your needs Disable	Select as needed
MBEDTLS_BLOWFISH_C	MBEDTLS_BLOWFISH_C	Enable Blowfish Grouping password	Can be disabled
MBEDTLS_ERROR_C	MBEDTLS_ERROR_C	Enable error codes to Wrong string conversion	Can be disabled
MBEDTLS_HMAC_DRBG_C	MBEDTLS_MD_C	Enable random byte generation Device	Can be disabled

Configuration	rely	illustrate	Optimization suggestions
MBEDTLS_PEM_WRITE_C	MBEDTLS_BASE64_C	This module adds support for Encode/write PEM file TLS Client unnecessary	Can be disabled
MBEDTLS_PK_WRITE_C function.	MBEDTLS_PK_C	Enable generic public key writing Systems Generally not needed, disable	Disable
MBEDTLS_RIPEMD160_C	None	RIPEMD (RACE Original integrity check message Message digest) is an encrypted Hash functions, generality Worse than SHA-1/2	Can be disabled
MBEDTLS_SSL_CACHE_C	None	Enable SSL caching can be disabled	
MBEDTLS_SSL_TICKET_C	MBEDTLS_CIPHER_C	server configuration	Disable
MBEDTLS_X509_CRL_PARSE_C	MBEDTLS_X509_USE_C	CRL: Certificate Revocation List Certificate Revocation List (CRL) Table Module	Can be disabled
MBEDTLS_X509_CSR_PARSE_C	MBEDTLS_X509_USE_C	Certificate Signing Request (CSR). Bookmark request parsing, For DER certificates	Can be disabled
MBEDTLS_X509_CREATE_C	MBEDTLS_BIGNUM_C, MBEDTLS_OID_C, MBEDTLS_PK_WRITE_C	Enable the X.509 core to Create a certificate, server need	Disable
MBEDTLS_X509_CRT_WRITE_C	MBEDTLS_X509_CREATE_C	Enables creation of certificates. Required by the server. Disable	
MBEDTLS_X509_CSR_WRITE_C	MBEDTLS_X509_CREATE_C	Enables creation of X.509 certificates Book Signing Request (CSR)	Can be disabled
MBEDTLS_XTEA_C	None	Enable XTEA group encryption code	Can be disabled
MBEDTLS_ECDSA_DETERMINISTIC	MBEDTLS_HMAC_DRBG_C	enables deterministic ECDSA (RFC 6979), preventing Lack of entropy during signing This may lead to the leakage of signing keys. Recommended to enable	Recommended to enable

6.4.4 Cipher suite related configuration

The cipher suite naming format in `mbedtls` is `MBEDTLS_TLS_PSK_WITH_AES_256_GCM_SHA384`.

`mbedtls` defines all supported ciphersuites in the array `static const int ciphersuite_preference[]`. If all ciphersuites are enabled, a large amount of ROM space will be occupied. It is recommended that users use the `MBEDTLS_SSL_CIPHERSUITES` macro to specify which ciphersuite the client and server use. After specifying the ciphersuite, all unnecessary ciphersuites and dependent functional components are disabled, and unnecessary elliptic curves are disabled to save ROM space to the greatest extent.

Configuration	rely	illustrate	Optimization suggestions
MBEDTLS_SSL_CIPHERSUITES	none	By specifying a cipher suite, you can save ROM and several hundred bytes of RAM. Note that you need to specify the cipher suite that the server supports, enable the relevant functional components for the cipher suite, and disable other functional components. If you only connect to one SSL server, you usually only need to define support for one cipher suite.	Specify only the cipher suites required by the root certificate
MBEDTLS_AES_C	None	Support for <code>*_WITH_AES_*</code> cipher suites by enabling AES	Select as needed
MBEDTLS_GCM_C	MBEDTLS_AES_C, MBEDTLS_CAMELLIA_C	Enable this configuration to support <code>*_AES_GCM_*</code> , Cipher suites of type <code>*_CAMELLIA_GCM_*</code>	Select as needed
MBEDTLS_REMOVE_ARC4_CIPHERSUITES	None	None is enabled by default, in SSL/Disable RC4 cipher suites in TLS	Select as needed
MBEDTLS_ARC4_C	None	Enable RC4 encryption suite, <code>*_WITH_RC4*</code> type cipher suite. Choose whether to disable it as needed	Select as needed

Configuration	rely	illustrate	Optimization suggestions
MBEDTLS_CAMELLIA_C	None	Enable Camellia Grouping Password for support <a href="#">*_WITH_CAMELLIA_*</a> The type of cipher suite. Select whether to Disable	Select as needed
MBEDTLS_CIPHER_MODE_CBC	none	Enable symmetric ciphers Block Chaining Mode (CBC). If using CBC cipher suites Need to be enabled	Select as needed
MBEDTLS_CIPHER_MODE_CFB	none	Enable symmetric ciphers Code Feedback Mode (CFB). If CFB encryption is used Code Suite needs to be enabled	Select as needed
MBEDTLS_CIPHER_MODE_CTR	none	Enable symmetric encryption Counter Block Cipher Mode (CTR). If you use CTR cipher suite You need to enable	Select as needed
MBEDTLS_CIPHER_PADDING_XXX		Enable filling in password layer If you disable For all fill modes, Only complete blocks can be Use with CBC	Choose according to your needs. Disable all
MBEDTLS_CIPHER_PADDING_PKCS7	Can be disabled		Can be disabled
MBEDTLS_CIPHER_PADDING_ONE_AND_ZEROS	Can be disabled		Can be disabled
MBEDTLS_CIPHER_PADDING_ZEROS_AND_LEN	Can be disabled		Can be disabled
MBEDTLS_CIPHER_PADDING_ZEROS	Can be disabled		Can be disabled
MBEDTLS_CIPHER_PADDING_XXX			
MBEDTLS_KEY_EXCHANGE_ECDSA_ENABLED,	Enable <a href="#">*_ECDH_RSA_*</a> Type of cipher suite	Select as needed	
MBEDTLS_X509_CRT_PARSE_C			
MBEDTLS_KEY_EXCHANGE_ECDSA_ENABLED type	Enable <a href="#">*_ECDH_RSA_*</a> cipher suite	Select as needed	
MBEDTLS_RSA_C,			
MBEDTLS_PKCS1_V15,			
MBEDTLS_X509_CRT_PARSE_C			

Configuration	rely	illustrate	Optimization suggestions
MBEDTLS_KEY_EXCHANGE_ECDSA_ENABLED	Enable	Enable	Select as needed
MBEDTLS_ECDSA_C,		*_ECDHE_ECDSA_* classes	
MBEDTLS_X509_CRT_PARSE_C		Cipher suite	
MBEDTLS_KEY_EXCHANGE_ECDSA_ENABLED	Enable	Enable	Select as needed
MBEDTLS_X509_CRT_PARSE_C		*_ECDHE_ECDSA_* classes	
		Cipher suite	
MBEDTLS_KEY_EXCHANGE_RSA_ENABLED	Enable	Enable	Select as needed
MBEDTLS_RSA_C,		Cipher suite *_HE_RSA_*	
MBEDTLS_PKCS1_V15,			
MBEDTLS_X509_CRT_PARSE_C			
MBEDTLS_KEY_EXCHANGE_PSK_ENABLED	Enable	Enable	Select as needed
MBEDTLS_PSK_C		Cipher suite	
MBEDTLS_KEY_EXCHANGE_PSK_ENABLED	Enable	Enable	Select as needed
MBEDTLS_PSK_C		Cipher suite	
MBEDTLS_KEY_EXCHANGE_ECDSA_PSK_ENABLED	Enable	Enable	Select as needed
MBEDTLS_ECDSA_C,		*_ECDHE_PSK_*	
MBEDTLS_PSK_C		Type Cipher Suite	
MBEDTLS_KEY_EXCHANGE_RSA_PSK_ENABLED	Enable	Enable	Select as needed
MBEDTLS_RSA_C,		Cipher suite *_RSA_PSK_*	
MBEDTLS_PKCS1_V15,			
MBEDTLS_X509_CRT_PARSE_C			
MBEDTLS_KEY_EXCHANGE_RSA_C	Enable	Enable	Select as needed
MBEDTLS_RSA_C,		*_RSA_* types	
MBEDTLS_PKCS1_V15,			
MBEDTLS_X509_CRT_PARSE_C			
MBEDTLS_CCM_C MBEDTLS_AES_C	Enable	Enable	Select as needed
or		CBC-MAC (CCM)	
MBEDTLS_CAMELLIA_C		The mode counter is used to	
		128-bit block cipher, using	
		To support AES-CCM	
		Cipher suite.	
		To choose whether to disable	
MBEDTLS_DES_C None		Enable DES block cipher can be disabled	
MBEDTLS_DHM_C None		Enable Diffie- Hellman-Merkle module	Select as needed
		Blocks to support	
		DHE-RSA,	
		DHE-PSK Cipher Suite	
		Select according to your needs.	
		Disable	

6.4.5 Elliptic Curve Related Configuration

After the user successfully selects the matching encryption suite and verifies that the handshake connection and encrypted communication can be established normally, you can try Try disabling elliptic curves that are not required by the cipher suite.

Configuration	rely	illustrate	Optimization suggestions
MBEDTLS_ECDH_C	Enable elliptic curve	Diffie-Hellman library. For support *_ECDHE_ECDSA_*, *_ECDHE_RSA_*, *_DHE_PSK_* Cipher suites	Select as needed
MBEDTLS_ECDSA_C	MBEDTLS_ASN1_WRITE_C, MBEDTLS_ASN1_PARSE_C	For support *_ECDHE_ECDSA_* classes Cipher suites of type	Select as needed
MBEDTLS_ECP_C	MBEDTLS_BIGNUM_C and at least one MBEDTLS_ECP_DP_XXX_ENABLED	Enable GF(p) elliptic curve Wire	Select as needed
MBEDTLS_ECP_DP_XXX_ENABLED	Enable all or a specific curve.	By default, all There are supported curves. Choose according to the actual situation Select a curve	Select as needed
MBEDTLS_ECP_DP_SECP192R1_ENABLED			Select as needed
MBEDTLS_ECP_DP_SECP224R1_ENABLED			Select as needed
MBEDTLS_ECP_DP_SECP256R1_ENABLED			Select as needed
MBEDTLS_ECP_DP_SECP384R1_ENABLED			Select as needed
MBEDTLS_ECP_DP_SECP521R1_ENABLED			Select as needed
MBEDTLS_ECP_DP_SECP192K1_ENABLED			Select as needed
MBEDTLS_ECP_DP_SECP224K1_ENABLED			Select as needed
MBEDTLS_ECP_DP_SECP256K1_ENABLED			Select as needed
MBEDTLS_ECP_DP_BP256R1_ENABLED			Select as needed
MBEDTLS_ECP_DP_BP384R1_ENABLED			Select as needed
MBEDTLS_ECP_DP_BP512R1_ENABLED			Select as needed
MBEDTLS_ECP_DP_CURVE25519_ENABLED			Select as needed



Configuration	rely	illustrate	Optimization suggestions
---------------	------	------------	--------------------------

<b>MBEDTLS_ECP_XXXX_ENABLED</b>			
---------------------------------	--	--	--

#### 6.4.6 TLS version selection related configuration

Usually SSL/TLS servers support multiple TLS protocol versions, but clients do not need to support all protocol versions. Therefore, after determining the TLS protocol version supported by the server, other versions of the protocol can be disabled.

Configuration	rely	illustrate	Optimization suggestions
---------------	------	------------	--------------------------

MBEDTLS_SSL_PROTO_TLS1 MBEDTLS_MD5_C, MBEDTLS_SHA1_C	Enable support for TLS 1.0 Support	Select as needed
---	---------------------------------------	------------------

MBEDTLS_SSL_PROTO_TLS1_1 MBEDTLS_MD5_C, MBEDTLS_SHA1_C	Enable support for TLS 1.1 Support	Select as needed
---	---------------------------------------	------------------

MBEDTLS_SSL_PROTO_TLS1_2 MBEDTLS_SHA1_C or MBEDTLS_SHA256_C or MBEDTLS_SHA512_C	Enable support for TLS 1.2 Support	Select as needed
---	---------------------------------------	------------------

#### 6.4.7 DTLS related configuration

DTLS is a secure encrypted connection based on UDP, which aims to ensure the data security of UDP communication. It does not support automatic retransmission and has packet loss issues, so it is slightly different from TLS when performing a handshake connection. The two share most of the same code. Therefore, you can optimize the DTLS encrypted connection by configuring the following table.

If DTLS is not required in your system, you can disable all the configurations in the following table.

Configuration	rely	illustrate	Optimization suggestions
---------------	------	------------	--------------------------

MBEDTLS_SSL_PROTO_DTLS MBEDTLS_SSL_PROTO_TLS1_2 or MBEDTLS_SSL_PROTO_TLS1_2	Enable DTLS feature, Used to perform UDP encryption	If DTLS is not required Encrypted connection, disable
---	---	--

MBEDTLS_SSL_DTLSANTI_REPLAY MBEDTLS_SSL_TLS_C, MBEDTLS_SSL_PROTO_DTLS	Enable support for DTLS in Anti-replay mechanism support	Can be disabled
--	---	-----------------

MBEDTLS_SSL_DTLS_HELLO_VERIFY MBEDTLS_SSL_PROTO_DTLS	Enable support for DTLS HelloVerifyRequest support	Need to be turned on
--	--	----------------------

Configuration	rely	illustrate	Optimization suggestions
MBEDTLS_SSL_DTLS_CLIENT_PORT_REUSE		Enable server-side support for clients that can reconnect from the same port, requires special support from the server	
MBEDTLS_SSL_DTLS_BADMAC_LIMIT		Enable support for MAC Error	Can be disabled
MBEDTLS_SSL_COOKIE_C	None	DTLS hello cookie	Can be disabled
		Supported. Can be disabled in non-DTLS environments	

6.5 References

- mbedTLS official website: <https://tls.mbed.org/>
- Configuration files used for testing

```
/* tls_config.h */ #ifndef MBEDTLS_CONFIG_H #define MBEDTLS_CONFIG_H #include <rtthread.h> #if defined(_MSC_VER) && !defined(_CRT_SECURE_NO_DEPRECATED) #define _CRT_SECURE_NO_DEPRECATED 1 #endif #define MBEDTLS_HAVE_ASM #define MBEDTLS_HAVE_TIME #define MBEDTLS_ASN1_PARSE_C #define MBEDTLS_ASN1_WRITE_C #define MBEDTLS_BIGNUM_C #define MBEDTLS_CIPHER_C #define MBEDTLS_AES_C #define MBEDTLS_CTR_DRBG_C // #define MBEDTLS_ECDH_C // #define MBEDTLS_ECDSA_C #define MBEDTLS_ECP_C // #define MBEDTLS_GCM_C #define MBEDTLS_MD_C // #define MBEDTLS_NET_C
```

```

#define MBEDTLS_OID_C
#define MBEDTLS_PK_C
#define MBEDTLS_PK_PARSE_C
#define MBEDTLS_SHA256_C //
#define MBEDTLS_SHA512_C #define
MBEDTLS_SSL_CLI_C // #define
MBEDTLS_SSL_SRV_C #define
MBEDTLS_SSL_TLS_C #define
MBEDTLS_X_509_CRT_PARSE_C #define
MBEDTLS_X509_USE_C #define
MBEDTLS_BASE64_C // #define
MBEDTLS_CERTS_C #define
MBEDTLS_PEM_PARSE_C #define
MBEDTLS_AES_ROM_TABLES #define
MBEDTLS_MPI_MAX_SIZE #define 384
MBEDTLS_MPI_WINDOW_SIZE #define 2
MBEDTLS_ECP_MAX_BITS #define 384
MBEDTLS_ECP_WINDOW_SIZE #define 2
MBEDTLS_ECP_FIXED_POINT_OPTIM 0 #define
MBEDTLS_ECP_NIST_OPTIM #define
MBEDTLS_ENTROPY_MAX_SOURCES 2 #define
MBEDTLS_SSL_CIPHERSUITES \
    MBEDTLS_TLS_RSA_WITH_AES_256_CBC_SHA256

// #define MBEDTLS_SSL_MAX_CONTENT_LEN #define 3584
MBEDTLS_NO_PLATFORM_ENTROPY
// #define MBEDTLS_TLS_DEFAULT_ALLOW_SHA1_IN_KEY_EXCHANGE #define
MBEDTLS_RSA_C #define
MBEDTLS_SHA1_C #define
MBEDTLS_TIMING_C #define
MBEDTLS_ENTROPY_HARDWARE_ALT #define
MBEDTLS_TIMING_ALT // #define
MBEDTLS_DEBUG_C #define
MBEDTLS_MD5_C
// #define MBEDTLS_HAVE_TIME_DATE #define
MBEDTLS_CIPHER_MODE_CBC // #define
MBEDTLS_CIPHER_MODE_CFB // #define
MBEDTLS_CIPHER_MODE_CTR
// #define MBEDTLS_CIPHER_PADDING_PKCS7 // #define
MBEDTLS_CIPHER_PADDING_ONE_AND_ZEROS // #define
MBEDTLS_CIPHER_PADDING_ZEROS_AND_LEN // #define
MBEDTLS_CIPHER_PADDING_ZEROS #define
MBEDTLS_REMOVE_ARC4_CIPHERSUITES // #define
MBEDTLS_ECP_DP_SECP192R1_ENABLED // #define
MBEDTLS_ECP_DP_SECP224R1_ENABLED #define
MBEDTLS_ECP_DP_SECP256R1_ENABLED

```

```

#define MBEDTLS_ECP_DP_SECP384R1_ENABLED //
#define MBEDTLS_ECP_DP_SECP521R1_ENABLED //
#define MBEDTLS_ECP_DP_SECP192K1_ENABLED //
#define MBEDTLS_ECP_DP_SECP224K1_ENABLED //
#define MBEDTLS_ECP_DP_SECP256K1_ENABLED //
#define MBEDTLS_ECP_DP_BP256R1_ENABLED //
#define MBEDTLS_ECP_DP_BP384R1_ENABLED //
#define MBEDTLS_ECP_DP_BP512R1_ENABLED //
#define MBEDTLS_ECP_DP_CURVE25519_ENABLED //
#define MBEDTLS_ECDSA_DETERMINISTIC // #define
MBEDTLS_KEY_EXCHANGE_ECDH_RSA_ENABLED // #define
MBEDTLS_KEY_EXCHANGE_ECDHE_RSA_ENABLED // #define
MBEDTLS_KEY_EXCHANGE_ECDHE_ECDSA_ENABLED // #define
MBEDTLS_KEY_EXCHANGE_ECDH_ECDSA_ENABLED // #define
MBEDTLS_KEY_EXCHANGE_DHE_RSA_ENABLED // #define
MBEDTLS_KEY_EXCHANGE_PSK_ENABLED // #define
MBEDTLS_KEY_EXCHANGE_DHE_PSK_ENABLED // #define
MBEDTLS_KEY_EXCHANGE_ECDHE_PSK_ENABLED // #define
MBEDTLS_KEY_EXCHANGE_RSA_PSK_ENABLED // #define
MBEDTLS_KEY_EXCHANGE_RSA_ENABLED // #define
MBEDTLS_PK_PARSE_EC_EXTENDED // #define
MBEDTLS_ERROR_STRERROR_DUMMY // #define
MBEDTLS_GENPRIME // #define
MBEDTLS_FS_IO // #define
MBEDTLS_PK_RSA_ALT_SUPPORT // #define
MBEDTLS_PKCS12_C // #define
MBEDTLS_PKCS1_V15 // #define
MBEDTLS_PKCS1_V21 // #define
MBEDTLS_SELF_TEST // #define
MBEDTLS_SSL_ALL_ALERT_MESSAGES
// #define MBEDTLS_SSL_ENCRYPT_THEN_MAC //
#define MBEDTLS_SSL_EXTENDED_MASTER_SECRET //
#define MBEDTLS_SSL_FALLBACK_SCSV //
#define MBEDTLS_SSL_CBC_RECORD_SPLITTING
// #define MBEDTLS_SSL_RENEGOTIATION //
#define MBEDTLS_SSL_MAX_FRAGMENT_LENGTH //
#define MBEDTLS_SSL_PROTO_TLS1 //
#define MBEDTLS_SSL_PROTO_TLS1_1 // #define
MBEDTLS_SSL_PROTO_TLS1_2 // #define
MBEDTLS_SSL_ALPN // #define
MBEDTLS_SSL_PROTO_DTLS // #define
MBEDTLS_SSL_DTLSANTI_REPLAY
// #define MBEDTLS_SSL_DTLS_HELLO_VERIFY //
#define MBEDTLS_SSL_DTLS_CLIENT_PORT_REUSE // #define
MBEDTLS_SSL_DTLS_BADMAC_LIMIT // #define
MBEDTLS_SSL_SESSION_TICKETS

```

```

// #define MBEDTLS_SSL_EXPORT_KEYS //
#define MBEDTLS_SSL_SERVER_NAME_INDICATION // #define
MBEDTLS_SSL_TRUNCATED_HMAC // #define
MBEDTLS_VERSION_FEATURES // #define
MBEDTLS_X509_CHECK_KEY_USAGE // #define
MBEDTLS_X509_CHECK_EXTENDED_KEY_USAGE // #define
MBEDTLS_X509_RSASSA_PSS_SUPPORT // #define
MBEDTLS_AESNI_C // #define
MBEDTLS_ARC4_C // #define
MBEDTLS_BLOWFISH_C // #define
MBEDTLS_CAMELLIA_C // #define
MBEDTLS_CCM_C // #define
MBEDTLS_DES_C // #define
MBEDTLS_DHM_C #define
MBEDTLS_ENTROPY_C // #define
MBEDTLS_ERROR_C // #define
MBEDTLS_HMAC_DRBG_C // #define
MBEDTLS_PADLOCK_C // #define
MBEDTLS_PEM_WRITE_C // #define
MBEDTLS_PK_WRITE_C // #define
MBEDTLS_PKCS5_C #define
MBEDTLS_PLATFORM_C // #define
MBEDTLS_RIPEMD160_C // #define
MBEDTLS_SSL_CACHE_C // #define
MBEDTLS_SSL_COOKIE_C // #define
MBEDTLS_SSL_TICKET_C
// #define MBEDTLS_VERSION_C //
#define MBEDTLS_X509_CRL_PARSE_C //
#define MBEDTLS_X509_CSR_PARSE_C //
#define MBEDTLS_X509_CREATE_C
// #define MBEDTLS_X509_CRT_WRITE_C //
#define MBEDTLS_X509_CSR_WRITE_C //
#define MBEDTLS_XTEA_C

#if defined(YOTTA_CFG_MBEDTLS_USER_CONFIG_FILE)
#include YOTTA_CFG_MBEDTLS_USER_CONFIG_FILE
#elif defined(MBEDTLS_USER_CONFIG_FILE) #include
MBEDTLS_USER_CONFIG_FILE
#endif

#include "mbedtls/check_config.h"

#define tls_malloc rt_malloc #define
tls_free rt_free #define tls_realloc
rt_realloc #define tls_calloc rt_calloc

```

```
#endif /* MBEDTLS_CONFIG_H */
```

Chapter 7

API Description

For the convenience of users, commonly used APIs are listed here and related instructions are given.

Note: For more detailed API content, please refer to the [ARM mbedtls API manual](#).

7.1 Application Layer API

The application layer API is provided to users for direct use in the App. This part of the API shields the mbedtls internal. It simplifies user use by eliminating specific operation steps.

7.1.1 mbedtls initialization

```
int mbedtls_client_init(MbedtlsSession session, void entropy, size_t entropy-
Len);
```

mbedtls client initialization function, used to initialize the underlying network interface, set up certificates, set up SSL sessions, etc.

parameter	describe
session	Input parameter, mbedtls session object MbedtlsSession
entropy	Input parameter, mbedtls entropy string
entropyLen	Input parameter, mbedtls entropy string length
return	describe
= 0	success
!0	fail

### 7.1.2 Configuring mbedtls context

```
int mbedtls_client_context(MbedtlsSession *session);
```

SSL layer configuration, the application uses the `mbedtls_client_context` function to configure the client context information, including certificate parsing, setting the host name, setting the default SSL configuration, setting the authentication mode (default MBEDTLS\_SSL\_VERIFY\_OPTIONAL) etc.

parameter	describe
session	Input parameter, mbedtls session object MbedtlsSession
return	describe
= 0	success
!0	fail

### 7.1.3 Establishing an SSL/TLS connection

```
int mbedtls_client_connect(MbedtlsSession *session);
```

Use the `mbedtls_client_connect` function to establish a channel for the SSL/TLS connection. This includes the entire handshake Connection process and certificate verification results.

parameter	describe
session	Input parameter, mbedtls session object MbedtlsSession
return	describe
= 0	success
!0	fail

### 7.1.4 Reading Data

- Writing data to an encrypted connection

```
int mbedtls_client_write(MbedtlsSession session, const unsigned char buf size_t len);
```

parameter	describe
session	Input parameter, mbedtls session object MbedtlsSession



parameter	describe
buf	Input parameter, data buffer to be written
len	Input parameter, length of data to be written
return	describe
= 0	success
!0	fail

- Read data from an encrypted connection

```
int mbedtls_client_read(MbedtlsSession session, unsigned char buf len);, size_t
```

parameter	describe
session	Input parameter, mbedtls session object MbedtlsSession
buf	Input parameter, mbedtls reads the content buffer
len	Input parameter, length of the content to be read by mbedtls
return	describe
= 0	success
!0	fail

## 7.1.5 Close mbedtls client

```
int mbedtls_client_close(MbedtlsSession *session);
```

The client actively closes the connection or closes the connection due to an abnormal error. You need to use [mbedtls\\_client\\_close](#)

Closes the connection and releases resources.

parameter	describe
session	Input parameter, mbedtls session object MbedtlsSession
return	describe
= 0	success
!0	fail

## 7.2 mbedtls related API

### 7.2.1 Setting the debug level

```
void mbedtls_debug_set_threshold( int threshold );
```

If MBEDTLS\_DEBUG\_C is enabled , you can use this function to set the debug level to control different levels of debugging.

Test log output.

parameter	describe
threshold	Input parameter, Debug level, default is 0, no debug log
return	describe
none	none

mbedtls defines five debugging levels, as follows:

Debug Level	describe
0	No debug
1	Error
2	State change
3	Informational
4	Verbose

### 7.2.2 Initialization phase related API

- Network context initialization

```
void mbedtls_net_init(mbedtls_net_context *ctx);
```

Initialize TLS network context, currently only has fd descriptor.

parameter	describe
ctx	Input parameter, network context object
return	describe
none	none

- **SSL context initialization**

```
void mbedtls_ssl_init(mbedtls_ssl_context *ssl);
```

SSL context initialization mainly clears the SSL context object to prepare for SSL connection.

parameter	describe
ssl	Input parameter, SSL context object
return	describe
none	none

- Initialize **SSL** configuration

```
void mbedtls_ssl_config_init( mbedtls_ssl_config *conf );
```

SSL configuration initialization mainly clears the SSL configuration structure object to prepare for SSL connection.

parameter	describe
conf	Input parameter, SSL configuration structure object
return	describe
none	none

- Initialize **the SSL** random byte generator

```
void mbedtls_ctr_drbg_init( mbedtls_ctr_drbg_context *ctx );
```

Clear the CTR\_DRBG (SSL random byte generator) context structure object, which is [mbedtls\\_ctr\\_drbg\\_seed](#) prepare for.

parameter	describe
ctx	Input parameter, CTR_DRBG structure object
return	describe
none	none

- Initialize **SSL** entropy

```
void mbedtls_entropy_init( mbedtls_entropy_context *ctx );
```

Initializes the SSL entropy structure object.

parameter	describe
ctx	Input parameter, entropy structure object
return	describe
none	none

- Set up **SSL/TLS** entropy source

```
int mbedtIs_ctr_drbg_seed( mbedtIs_ctr_drbg_context *ctx,
                           int (*f_entropy)(void *, unsigned char *, size_t),
                           void *p_entropy,
                           const unsigned char *custom,
                           size_t len );
```

Sets the entropy source for SSL/TLS entropy, to facilitate the generation of sub-seeds.

parameter	describe
ctx	Input parameter, CTR_DRBG structure object
f_entropy	Input parameter, entropy callback
p_entropy	Input parameter, entropy structure (mbedtIs_entropy_context) object
custom	Input parameter, personalized data (device specific identifier), can be Thought it was empty
len	Personalized data length
return	describe
none	none

- Set up a root certificate list

void mbedtIs\_x509\_init(mbedtIs\_x509\_cert \*crt);

Initialize the root certificate chain list.

parameter	describe
crt	Input parameter, x509 certificate structure object

parameter	describe
return	describe
none	none

- Parsing root certificates

```
int mbedtls_x509_crt_parse( mbedtls_x509_crt chain, const unsigned char buf,
size_t buflen );
```

Parse interpretively. Parse one or more certificates in buf and add them to the root certificate chain list. If possible

If a certificate is parsed, the result is the number of failed certificates it encountered. If it did not complete correctly, the first error is returned.

The root certificate is located in the `mbedtls_root_certificate` array in the `ports/src/tls_certificate.c` file .

parameter	describe
chain	Input parameter, x509 certificate structure object
buf	Input parameter, buffer for storing root certificates, <a href="#">mbedtls_root_certificate</a> array
Buflen	Input parameter, buffer size for storing root certificates
return	describe
none	none

- Set the hostname

```
int mbedtls_ssl_set_hostname( mbedtls_ssl_context ssl, const char hostname
);
```

Note that the `hostname` set here must correspond to the common name in the server certificate , that is, the CN field.

- Load the default **SSL** configuration

```
int mbedtls_ssl_config_defaults( mbedtls_ssl_config *conf,
int endpoint, int transport, int preset
);
```

Before use, you need to call the `mbedtls_ssl_config_init` function to initialize the SSL configuration structure object.

parameter	describe
conf	Input parameter, SSL configuration structure object
endpoint	Input parameter, MBEDTLS_SSL_IS_CLIENT or MBEDTLS_SSL_IS_SERVER
transport	Input parameters, TLS: MBEDTLS_SSL_TRANSPORT_STREAM; DTLS: MBEDTLS_SSL_TRANSPORT_DATAGRAM
preset	Input parameters, predefined MBEDTLS_SSL_PRESET_XXX types Value, used by default MBEDTLS_SSL_PRESET_DEFAULT
return	describe
none	none

- Set the certificate verification mode > void mbedtls\_ssl\_conf\_authmode( mbedtls\_ssl\_config \*conf,  
int authmode );

Set the certificate verification mode default value: [MBEDTLS\\_SSL\\_VERIFY\\_NONE](#) on the server and  
[MBEDTLS\\_SSL\\_VERIFY\\_REQUIRED](#) or [MBEDTLS\\_SSL\\_VERIFY\\_OPTIONAL](#) (used by default).

[MBEDTLS\\_SSL\\_VERIFY\\_OPTIONAL](#) means that communication can continue even if certificate verification fails.

parameter	describe
conf	Input parameter, SSL configuration structure object
authmode	Input parameter, certificate verification mode
return	describe
none	none

- Set the data required to verify the peer certificate

```
void mbedtls_ssl_conf_ca_chain( mbedtls_ssl_config *conf,
                               mbedtls_x509_crt *ca_chain,
                               mbedtls_x509_crl *ca_crl );
```

Configure the trusted certificate chain into an SSL configuration structure object.

parameter	describe
conf	Input parameter, SSL configuration structure object
ca_chain	Input parameter, trusted CA certificate chain, stored in In the member object cacert of MbedTLSSession
ca_crl	Input parameter, trusted CA CRLs, can be empty
return	describe
none	none

- Set up the random number generator callback

```
void mbedtls_ssl_conf_rng( mbedtls_ssl_config *conf,
                          int (*f_rng)(void *, unsigned char *, size_t),
                          void *p_rng );
```

parameter	describe
conf	Input parameter, SSL configuration structure object
f_rng	Input parameter, random number generator function
p_rng	Input parameters, random number generator function parameters
return	describe
none	none

- Setting up the SSL context

```
int mbedtls_ssl_setup( mbedtls_ssl_context *ssl,
                      const mbedtls_ssl_config *conf );
```

Set the SSL configuration structure object into the SSL context.

parameter	describe
ssl	Input parameter, SSL context structure object
conf	Input parameter, SSL configuration structure object
return	describe

parameter	describe
<code>= 0</code>	success
<code>- 0x7F00</code>	Memory allocation failed

### 7.2.3 Connection Phase Related APIs

```
int mbedtls_net_connect( mbedtls_net_context *ctx,
                        const char *host, const char *port,
                        int proto );
```

Establishes a network connection with the given host, port and proto protocol.

parameter	describe
<code>ctx</code>	Input parameter, NET network configuration structure object
<code>host</code>	Input parameter, the specified host name to be connected
<code>port</code>	Input parameter, the specified host port number
<code>proto</code>	Input parameter, the specified protocol type, MBEDTLS_NET_PROTO_TCP or MBEDTLS_NET_PROTO_UDP
<code>return</code>	describe
<code>= 0</code>	success
<code>- 0x0042</code>	Socket creation failed
<code>- 0x0052</code>	Unknown hostname, DNS resolution failed
<code>- 0x0044</code>	Network connection failed

- Set up network layer read and write interfaces

```
void mbedtls_ssl_set_bio( mbedtls_ssl_context *ssl,
                        void *p_bio,
                        mbedtls_ssl_send_t *f_send,
                        mbedtls_ssl_rcv_t *f_rcv,
                        mbedtls_ssl_rcv_timeout_t *f_rcv_timeout );
```

Sets read and write functions for the network layer, which are called by `mbedtls_ssl_read` and `mbedtls_ssl_write` functions.



- For TLS, the user only needs to provide either `f_rcv` or `f_rcv_timeout`. If both are provided, the default is

Using `f_rcv_timeout` callback

- For DTLS, users need to provide `f_rcv_timeout` callback function

parameter	describe
ssl	Input parameter, SSL context structure object
p_bio	Input parameter, socket descriptor
f_send	Input parameter, network layer write callback function
f_rcv	Input parameter, network layer read callback function
f_rcv_timeout	Input parameter, network layer non-blocking read callback function with timeout
return	describe
none	none

- **SSL/TLS** handshake interface

```
int mbedtls_ssl_handshake(mbedtls_ssl_context *ssl);
```

Performs an SSL/TLS handshake operation.

parameter	describe
ssl	Input parameter, SSL context structure object
return	describe
= 0	success
- 0x6900	SSL clients need to read the call
-0x6880	SSL clients need to write calls
- 0x6A80	DTLS client must retry for hello verification
other	Other SSL-specific error codes

Note that if you are using DTLS, you need to handle the [0x6A80](#) error separately , as it is the expected return value instead of the actual error.

- Get certificate verification results

```
uint32_t mbedtls_ssl_get_verify_result( const mbedtls_ssl_context *ssl);
```

parameter	describe
ssl	Input parameter, SSL context structure object
return	describe
= 0	success
- 1	The returned result is not available
other	BADCERT_xxx and BADCRL_xxx Flags For combinations of

Or the API interface of the certificate verification result. For specific error information, you need to use `mbedtls_x509_crt_verify_info` Interface acquisition.

```
int mbedtls_x509_crt_verify_info( char *buf, size_t size,
                                const char *prefix,
                                uint32_t flags );
```

Use the `mbedtls_x509_crt_verify_info` function to get an information string about the certificate verification status and store it in the buffer member of the `MbedtlsSession` object.

parameter	describe
buf	Input parameter, buffer for storing verification status information string
size	Input parameter, buffer size
prefix	Input parameter, line prefix
flags	Input parameter, from <code>mbedtls_x509_crt_verify_info</code> The value returned by the function
return	describe
Integer	The length of the string written (excluding the terminator) or Negative error codes

### 7.2.4 Read and Write APIs

#### SSL/TLS write function

```
int mbedtls_ssl_read(mbedtls_ssl_context ssl, unsigned char buf, size_t len);
```

Read data from SLL/TLS, read at most 'len' bytes of data.

parameter	describe
ssl	Input parameter, SSL context structure object
buf	Input parameter, buffer for receiving read data
len	Input parameter, length of data to be read
return	describe
> 0	The length of the data read
= 0	Read to end character
- 0x6900	SSL clients need to read the call
-0x6880	SSL clients need to write calls
-0x6780	SSL client needs to reconnect
other	Other SSL-specific error codes

**SSL/TLS read function**

```
int mbedtls_ssl_write( mbedtls_ssl_context ssl, const unsigned char buf, size_t
len );
```

Writes data to SSL/TLS, up to 'len' bytes long.

parameter	describe
ssl	Input parameter, SSL context structure object
buf	Input parameter, buffer to be written data
len	Input parameter, length of the data to be written
return	describe
> 0	The actual length of the data written
= 0	Read to end character
- 0x6900	SSL clients need to read the call
-0x6880	SSL clients need to write calls
other	Other SSL-specific error codes