RT-THREAD MBEDTLS 用户手册

RT-THREAD 文档中心

WWW.RT-THREAD.ORG

Wednesday 30th January, 2019

目录

第 1 章

版本和修订

Date	Version	Author	Note
2018-08-01	v0.1	MurphyZhao	初始版本
2018-08-14	v0.2	MurphyZhao	更新证书添加方式，增加资源占用优化手册

第 2 章

软件包介绍

mbedtls 软件包是 **RT-Thread** 基于 ARMmbed/mbedtls 开源库的移植。

mbedTLS（前身 PolarSSL）是一个由 ARM 公司开源和维护的 SSL/TLS 算法库。其使用 C 编程语言以最小的编码占用空间实现了 SSL/TLS 功能及各种加密算法，易于理解、使用、集成和扩展，方便开发人员轻松地在嵌入式产品中使用 SSL/TLS 功能。

mbedTLS 软件包提供了如下的能力:

- 完整的 **SSL v3**、**TLS v1.0**、**TLS v1.1** 和 **TLS v1.2** 协议实现
- **X.509** 证书处理
- 基于 TCP 的 TLS 传输加密
- 基于 UDP 的 DTLS（Datagram TLS）传输加密
- 其它加解密库实现

有关 mbedTLS 的更多信息，请参阅 https://tls.mbed.org。

2.1 软件框架图

mbedTLS 软件包提供了一组可以单独使用和编译的加密组件，各组件及其可能的依赖关系如下图所示：

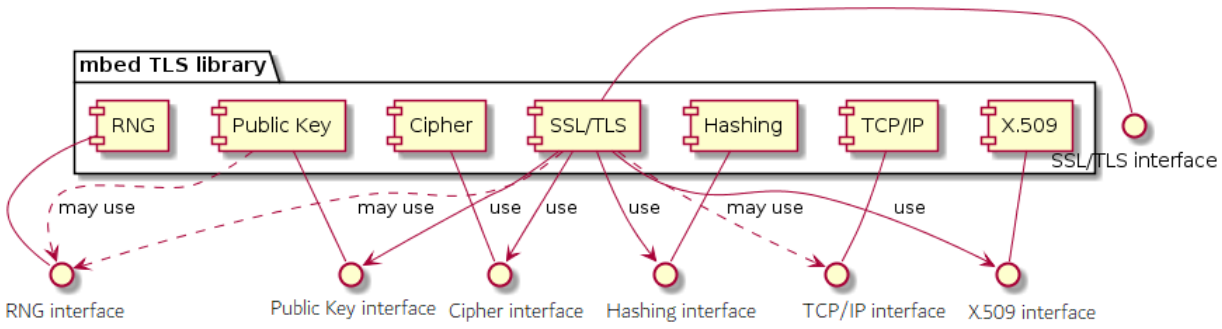

图 2.1: *mbedtls 软件框架图*

2.2 软件包目录结构

ports 目录是 RT-Thread 移植 mbedtls 软件包时所涉及到的移植文件，使用 scons 进行重新构建。

```
mbedtls
|   LICENSE                          // 软件包许可协议
|   README.md                        // 软件包使用说明
|   SConscript                       // RT-Thread 默认的构建脚本
+---certs                            // certs 根目录存放用户 CA 证书
|   +---default                      // default 目录保存着预置的 CA 证书
+---docs
|   +---figures                      // 文档使用图片
|   |   api.md                       // API 使用说明
|   |   introduction.md              // 软件包详细介绍
|   |   LICENSE                      // 许可证文件
|   |   principle.md                 // 实现原理
|   |   footprint-optimization-guide.md // 资源占用优化参考指南
|   |   README.md                    // 文档结构说明
|   |   samples.md                   // 软件包示例
|   |   user-guide.md                // 使用说明
|   +---version.md                   // 版本说明
+---ports                            // 移植文件
|   +---inc
|   +---src
+---samples                          // 示例程序
+---mbedtls                          // ARM mbedtls 源码
```

第 3 章

示例程序

该示例程序提供了一个简单的 TLS client，与测试网站建立 TLS 连接并获取加密数据。

示例文件

示例程序路径	说明
samples/tls_app_test.c	TLS 测试例程

3.1 例程工作流程

本例程使用了 RT-Thread 官方 TLS 测试网站 `www.rt-thread.org`，使用 `mbedtls_client_write` 函数发送 HTTP 测试请求，成功后，该网站会返回文本数据，测试例程将解析后的数据输出到控制台。

- 例程使用的 HTTP 请求数据如下所示

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

- mbedTLS 测试例程的基本工作流程如下所示
  - client 连接测试网站 `www.rt-thread.org`
  - client 和 server 握手成功
  - client 发送请求
  - server 回应请求
  - TLS 测试成功/失败

3.2 准备工作

3.2.1 获取软件包

- menuconfig 配置软件包

  打开 RT-Thread 提供的 ENV 工具，使用 **menuconfig** 配置软件包。

  启用 mbedtls 软件包，并配置使能测试例程（Enable a mbedtls client example），如下所示：

```
RT-Thread online packages --->
  security packages  --->
        Select Root Certificate  --->      # 选择证书文件
    [*] mbedtls: An portable and flexible SSL/TLS library # 打开 mbedtls 软件包
    [*]   Store the AES tables in ROM      # 将 AES 表存储在 ROM 中
    (2)   Maximum window size used         # 用于点乘的最大“窗口”大小（2-7）
    (3584) Maxium fragment length in bytes # 配置数据帧大小
    [*]   Enable a mbedtls client example  # 开启 mbedtls 测试例程
    [ ]   Enable Debug log output          # 开启调试 log 输出
          version (latest)  --->           # 选择软件包版本，默认为最新版本
```

- 使用 `pkgs --update` 命令下载软件包
- 编译下载

3.2.2 同步设备时间

SSL/TLS 服务器进行证书校验的过程中，会对当前发起校验请求的时间进行认证，如果时间不满足服务器的要求，就会校验证书失败。因此，我们需要为设备同步本地时间。

- 方式一：使用 **date** 命令

  未同步过时间的设备输入 **date** 命令后如下所示：

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

使用 **date** 设置当前时间，如下所示：

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

- 方式二：使用 NTP 同步网络时间

  该方式需要依赖 NTP 工具包，使用 `menuconfig` 配置获取，如下所示：

```
RT-Thread online packages --->
    IoT - internet of things  --->
        -*- netutils: Networking utilities for RT-Thread  --->
```

```
            -*-   Enable NTP(Network Time Protocol) client
            (8)     Timezone for calculate local time
            (cn.ntp.org.cn) NTP server name
```

使用命令 **ntp_sync** 同步网络时间

```
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 启动例程

在 MSH 中使用命令 **tls_test** 执行示例程序，成功建立 TLS 连接后，设备会从服务器拿到一组密码套件，设备 log 如下所示：

```
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.
```

第 4 章

工作原理

mbedtls 软件包是对 SSL/TLS 协议的实现。SSL（安全套接层）和 TLS（传输安全层）均是为了保证传输过程中信息的安全，是在明文传输基础上进行的加密，然后以密文的形式传输数据。

mbedTLS 建立安全通信连接需要经过以下几个步骤：

- 初始化 SSL/TLS 上下文
- 建立 SSL/TLS 握手
- 发送、接收数据
- 交互完成，关闭连接

其中，最关键的步骤就是 **SSL/TLS 握手**连接的建立，这里需要进行证书校验。

4.1 SSL/TLS 握手流程

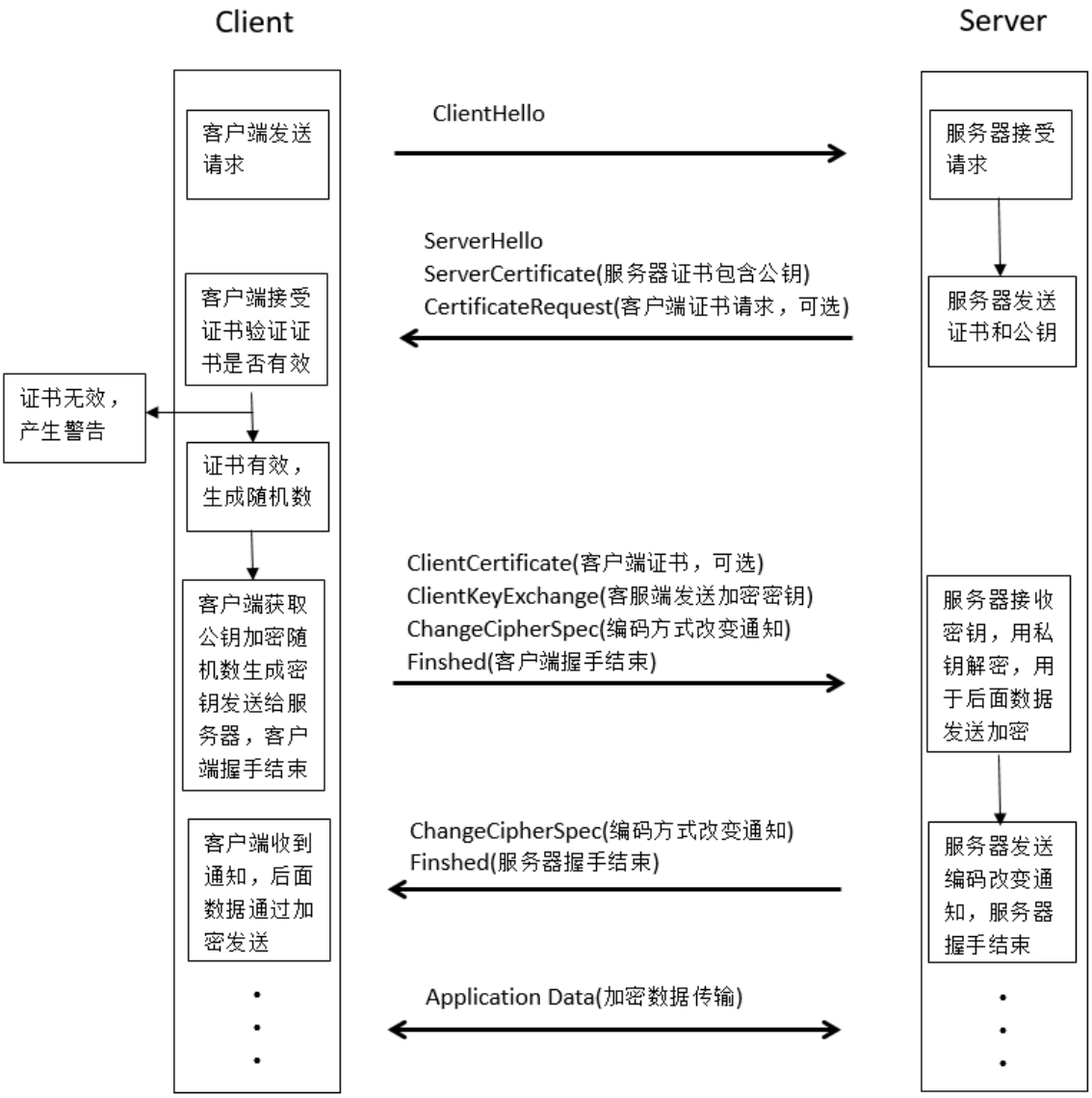

图 4.1: *SSL/TLS 握手交互流程*

4.2 DTLS 握手流程

为了避免拒绝服务攻击，DTLS 采用和 IKE 一样的无状态 cookie 技术。当客户端发送 client hello 消息后，服务器发送 HelloVerifyRequest 消息，这个消息包含了无状态 cookie。客户端收到之后必须重传添加上了 cookie 的 clienthello。

DTLS 握手流程如下图所示：

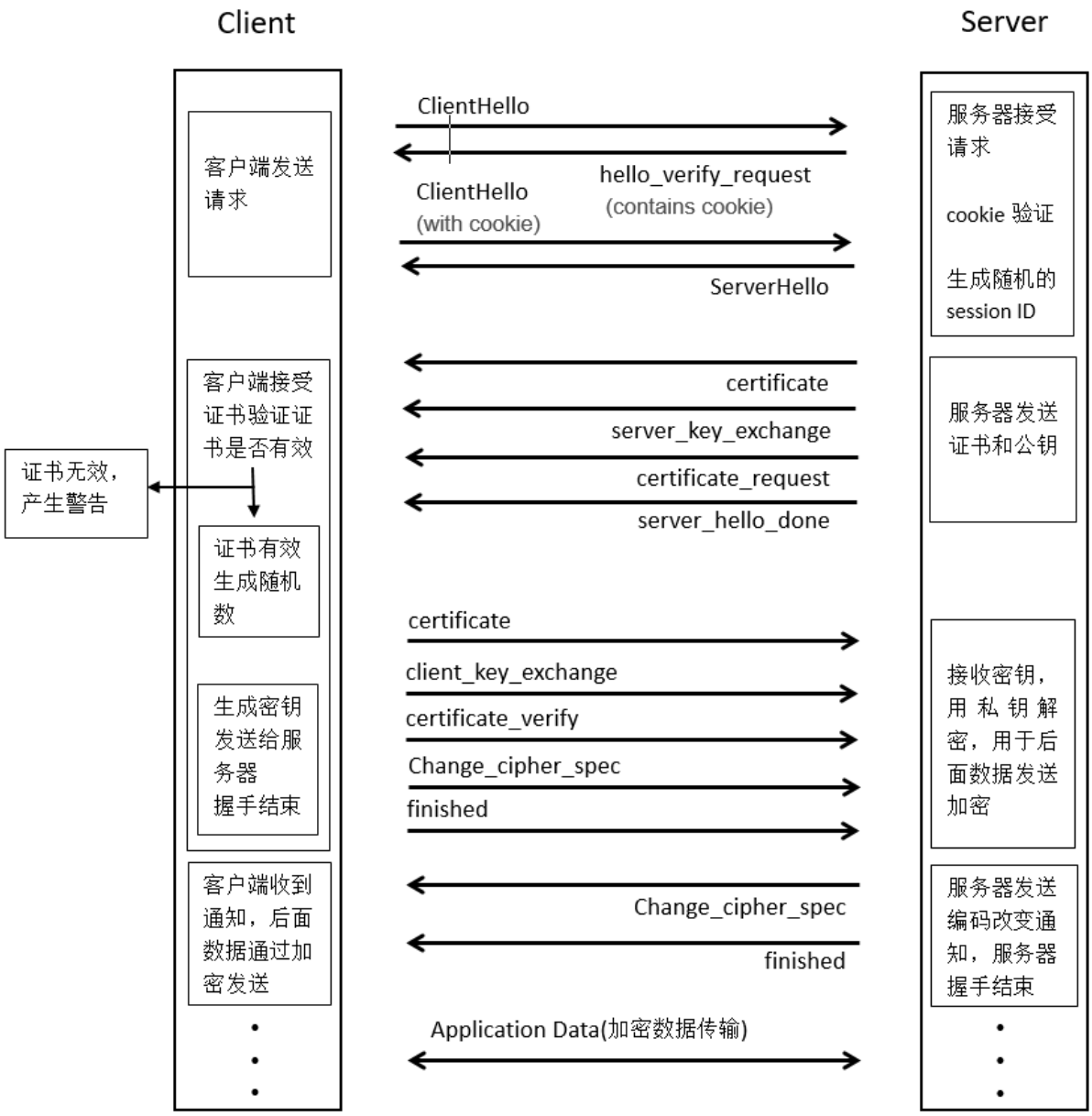

图 4.2: *DTLS 握手流程*

第 5 章

使用指南

这里主要介绍 mbedtls 程序的基本使用流程，并针对使用过程中经常涉及到的结构体和重要 API 进行简要说明。

mbedtls 的基本工作流程如下所示：

- 初始化 SSL/TLS 上下文
- 建立 SSL/TLS 握手
- 发送、接收数据
- 交互完成，关闭连接

5.1 menuconfig 配置说明

获取 mbedtls 软件包或者修改用户配置都需要使用 `menuconfig`。需要用户打开 ENV 工具，并将目录切换到您所用的 BSP 目录，使用 `menuconfig` 命令打开配置界面。

在 `RT-Thread online packages` → `security packages` 中选择 **mbedtls** 软件包，操作界面如下图所示：

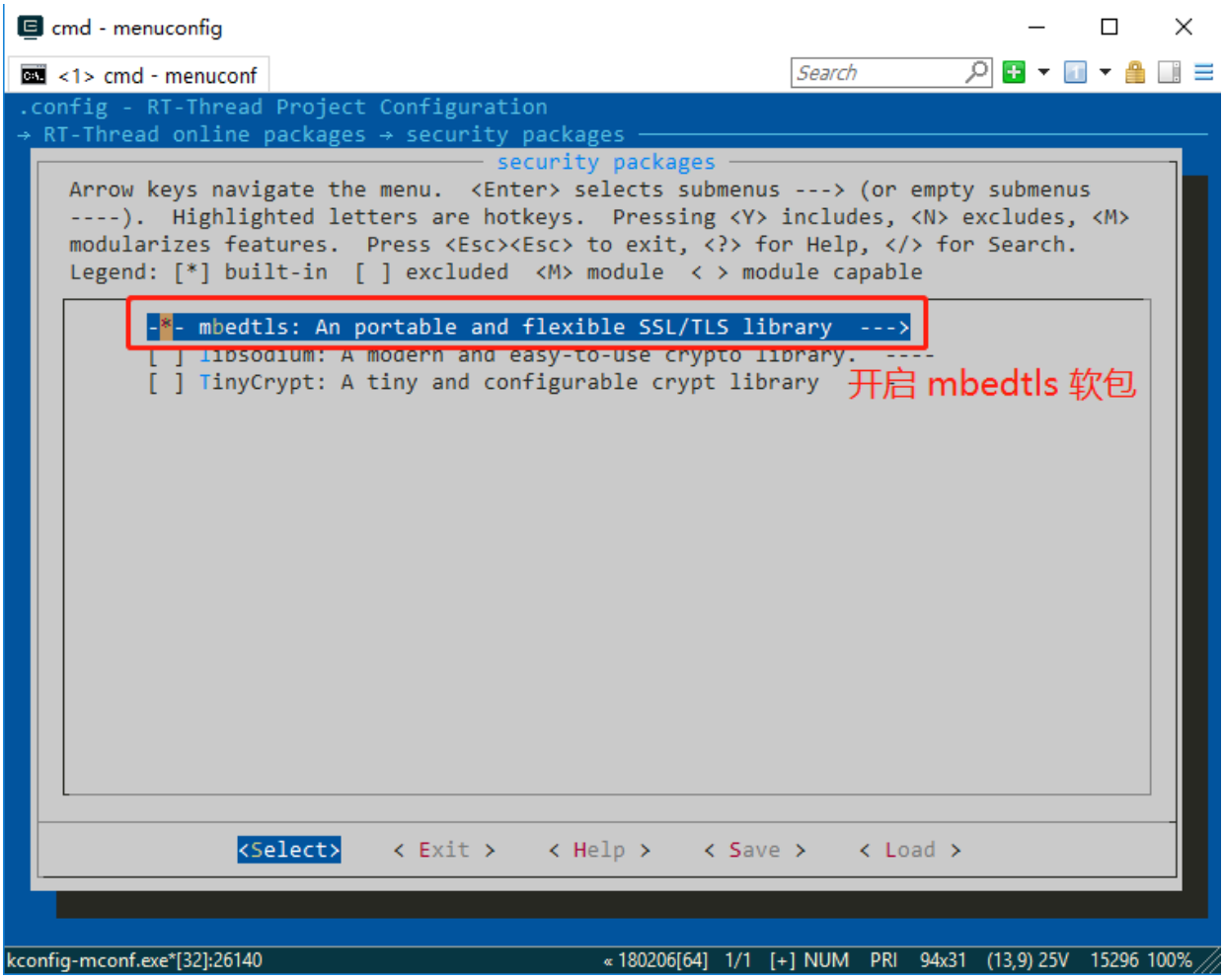

图 5.1: *打开 mbedtls 软件包*

详细的配置介绍如下所示：

```
RT-Thread online packages --->
  security packages  --->
        Select Root Certificate  --->      # 选择证书文件
    [*] mbedtls: An portable and flexible SSL/TLS library # 打开 mbedtls 软件包
    [*]   Store the AES tables in ROM      # 将 AES 表存储在 ROM 中
    (2)   Maximum window size used         # 用于点乘的最大“窗口”大小（2-7）
    (3584) Maxium fragment length in bytes # 配置数据帧大小
    [*]   Enable a mbedtls client example  # 开启 mbedtls 测试例程
    [ ]   Enable Debug log output          # 开启调试 log 输出
          version (latest)  --->           # 选择软件包版本，默认为最新版本
```

- `Using all default CA` 配置选项会将 `certs/default` 目录下的所有预置证书加入编译，将占用很大的内存
- `Using user CA` 配置选项允许用户将自己需要的证书文件加入编译，需要用户将证书文件拷贝到 `certs` 根目录

选择合适的配置项后，使用 `pkgs --update` 命令下载软件包并更新用户配置。

5.2 功能配置说明

mbedtls 功能模块的开启与关闭定义在 mbedtls/config.h 和 ports/inc/tls_config.h 文件中

`mbedtls/config.h` 是 mbedtls 源码里提供的配置文件，`ports/inc/tls_config.h` 是 RT-Thread 基于 mbedtls 源码中的配置文件进行的裁剪和适配。

最终，用户使用的是 RT-Thread 提供的配置文件 **`ports/inc/tls_config.h`**。

用户可以通过文件中的宏来使能或失能部分不需要使用的功能模块，从而将 mbedtls 配置到合适的尺寸。

5.3 证书配置说明

- 预置的 CA 证书文件存放在 `certs/default` 目录中
- 用户增加的 CA 证书文件存放在 `certs` 根目录中

`certs/default` 目录中已经包含了大多数 CA 根证书，如果您使用的根证书不在该文件夹内，需要用户将自己的 **PEM 格式**的 CA 证书拷贝 `certs` 根目录下。（仅支持 **PEM 格式**证书，不支持 **DER 格式**证书）。

该证书文件中已经包含了大多数 CA 根证书,，参考后边的 添加新证书 章节。

5.4 初始化 TLS 会话

```
typedef struct MbedTLSSession
{
    char* host;
    char* port;

    unsigned char *buffer;              // 公用数据缓冲区
    size_t buffer_len;                  // 缓冲区大小

    mbedtls_ssl_context ssl;            // 保存 ssl 基本数据
    mbedtls_ssl_config conf;            // 保存 ssl 配置信息
    mbedtls_entropy_context entropy;    // 保存 ssl 熵配置
    mbedtls_ctr_drbg_context ctr_drbg;  // 保存随机字节发生器配置
    mbedtls_net_context server_fd;      // 保存文件描述符
    mbedtls_x509_crt cacert;            // 保存认证信息
} MbedTLSSession;
```

`MbedTLSSession` 用于保存建立 TLS 会话连接时的配置信息，在 TLS 上下文中传递使用。用户在使用建立 TLS 会话前，必须定义一个存储会话内容的结构体，如下所示：

```
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);
```

这里需要设置 SSL/TLS 服务器的 host 和 port，以及数据接收 buffer 等配置。

5.5 初始化 SSL/TLS 客户端

应用程序使用 mbedtls_client_init 函数初始化 TLS 客户端。

初始化阶段按照 API 参数定义传入相关参数即可，主要用来初始化网络接口、证书、SSL 会话配置等 SSL 交互必须的一些配置，以及设置相关的回调函数。

示例代码如下所示：

```
char *pers = "hello_world"; // 设置随机字符串种子
if((ret = mbedtls_client_init(tls_session, (void *)pers, strlen(pers))) != 0)
{
    rt_kprintf("MbedTLSClientInit err return : -0x%x\n", -ret);
    goto __exit;
}
```

实际调用的 mbedtls 库函数如下所示：

5.6 初始化 SSL/TLS 客户端上下文

应用程序使用 mbedtls_client_context 函数配置客户端上下文信息，包括证书解析、设置主机名、设置默认 SSL 配置、设置认证模式（默认 MBEDTLS_SSL_VERIFY_OPTIONAL）等。

示例代码如下所示：

```
if((ret = mbedtls_client_context(tls_session)) < 0)
{
    rt_kprintf("MbedTLSCLientContext err return : -0x%x\n", -ret);
    goto __exit;
}
```

5.7 建立 SSL/TLS 连接

使用 mbedtls_client_connect 函数为 SSL/TLS 连接建立通道。这里包含整个的握手连接过程，以及证书校验结果。

示例代码如下所示：

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

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

5.8 读写数据

向 SSL/TLS 中写入数据

示例代码如下所示：

```
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 rtt\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;
    }
}
```

从 SSL/TLS 中读取数据

示例代码如下所示：

```
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;
}
```

注意，如果读写接口返回了一个错误，必须关闭连接。

5.9 关闭 SSL/TLS 客户端连接

客户端主动关闭连接或者因为异常错误关闭连接，都需要使用 `mbedtls_client_close` 关闭连接并释放资源。

示例代码如下所示：

```
mbedtls_client_close(tls_session);
```

5.10 mbedtls 使用范式

参考示例程序 `samples/tls_app_test.c`。

5.11 添加新证书

CA 证书有两种常用格式 **PEM 格式**和 **DER** 格式，目前 RT-Thread mbedtls 仅支持 **PEM 格式**的证书文件。

- PEM 格式证书

  PEM 格式证书通常是以 **.pem** 和 **.cer** 后缀名结尾的文件。

  使用文本编辑器打开后，文件内容以 `-----BEGIN CERTIFICATE-----` 开头，以 `-----END CERTIFICATE-----` 结尾。

- DER 格式证书

  DER 格式证书是二进制文件类型。

5.11.1 根证书样式

双击 **.cer** 后缀名结尾的 CA 文件（Windows 系统）可以看到证书的签发机构和有效期，如下图所示：

图 5.2: *根证书信息*

PEM 格式格式的证书文件内容内容样式如下所示：

```
-----BEGIN CERTIFICATE-----
MIIDdTCCAl2gAwIBAgILBAAAAAABFUtaw5QwDQYJKoZIhvcNAQEFBQAwVzELMAkG
A1UEBhMCQkUxGTAXBgNVBAoTEEdsb2JhbFNpZ24gbnYtc2ExEDAOBgNVBAsTB1Jv
b3QgQ0ExGzAZBgNVBAMTEkdsb2JhbFNpZ24gUm9vdCBDQTAeFw05ODA5MDExMjAw
MDBaFw0yODAxMjgxMjAwMDBaMFcxCzAJBgNVBAYTAkJFMRkwFwYDVQQKExBHbG9i
YWxTaWduIG52LXNhMRAwDgYDVQQLEwdSb290IENBMRswGQYDVQQDExJHbG9iYWxT
aWduIFJvb3QgQ0EwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQDaDuaZ
jc6j40+Kfvvxi4Mla+pIH/EqsLmVEQS98GPR4mdmzxzdzxtIK+6NiY6arymAZavp
xy0Sy6scTHAHoT0KMM0VjU/43dSMUBUc71DuxC73/OlS8pF94G3VNTCOXkNz8kHp
1Wrjsok6Vjk4bwY8iGlbKk3Fp1S4bInMm/k8yuX9ifUSPJJ4ltbcdG6TRGHRjcdG
snUOhugZitVtbNV4FpWi6cgKOOvyJBNPc1STE4U6G7weNLWLBYy5d4ux2x8gkasJ
U26Qzns3dLlwR5EiUWMWea6xrkEmCMgZK9FGqkjWZCrXgzT/LCrBbBlDSgeF59N8
9iFo7+ryUp9/k5DPAgMBAAGjQjBAMA4GA1UdDwEB/wQEAwIBBjAPBgNVHRMBAf8E
BTADAQH/MB0GA1UdDgQWBBRge2YaRQ2XyolQL30EzTSo//z9SzANBgkqhkiG9w0B
AQUFAAOCAQEA1nPnfE920I2/7LqivjTFKDK1fPxsnCwrvQmeU79rXqoRSLblCKOz
yj1hTdNGCbM+w6DjY1Ub8rrvrTnhQ7k4o+YviiY776BQVvnGCv04zcQLcFGUl5gE
38NflNUVyRRBnMRddWQVDf9VMOyGj/8N7yy5Y0b2qvzfvGn9LhJIZJrglfCm7ymP
AbEVtQwdpf5pLGkkeB6zpxxxYu7KyJesF12KwvhHhm4qxFYxldBniYUr+WymXUad
DKqC5JlR3XC321Y9YeRq4VzW9v493kHMB65jUr9TU/Qr6cf9tveCX4XSQRjbgbME
HMUfpIBvFSDJ3gyICh3WZlXi/EjJKSZp4A==
```

```
-----END CERTIFICATE-----
```

5.11.2 获取根证书

- 直接向服务商索取

  向服务商索取 base64 编码 X.509 编码的 **PEM 格式**证书文件。

- 从服务商网站导出

  – 浏览器打开服务商网站，以 https://www.rt-thread.org/ 为例
  – 点击浏览器地址栏的 安全，然后点击证书

图 5.3: *获取网站根证书*

  – 查看证书详细信息

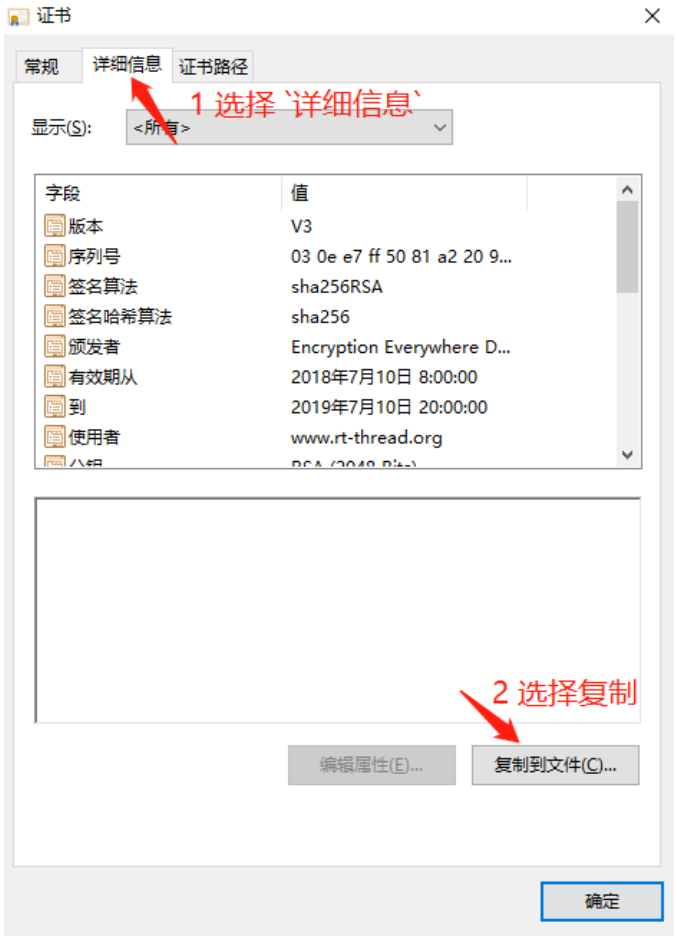

图 5.4: *查看证书详细信息*

– 根证书导出向导

图 5.5: *导出根证书向导*

– 选择导出 Base64 编码证书

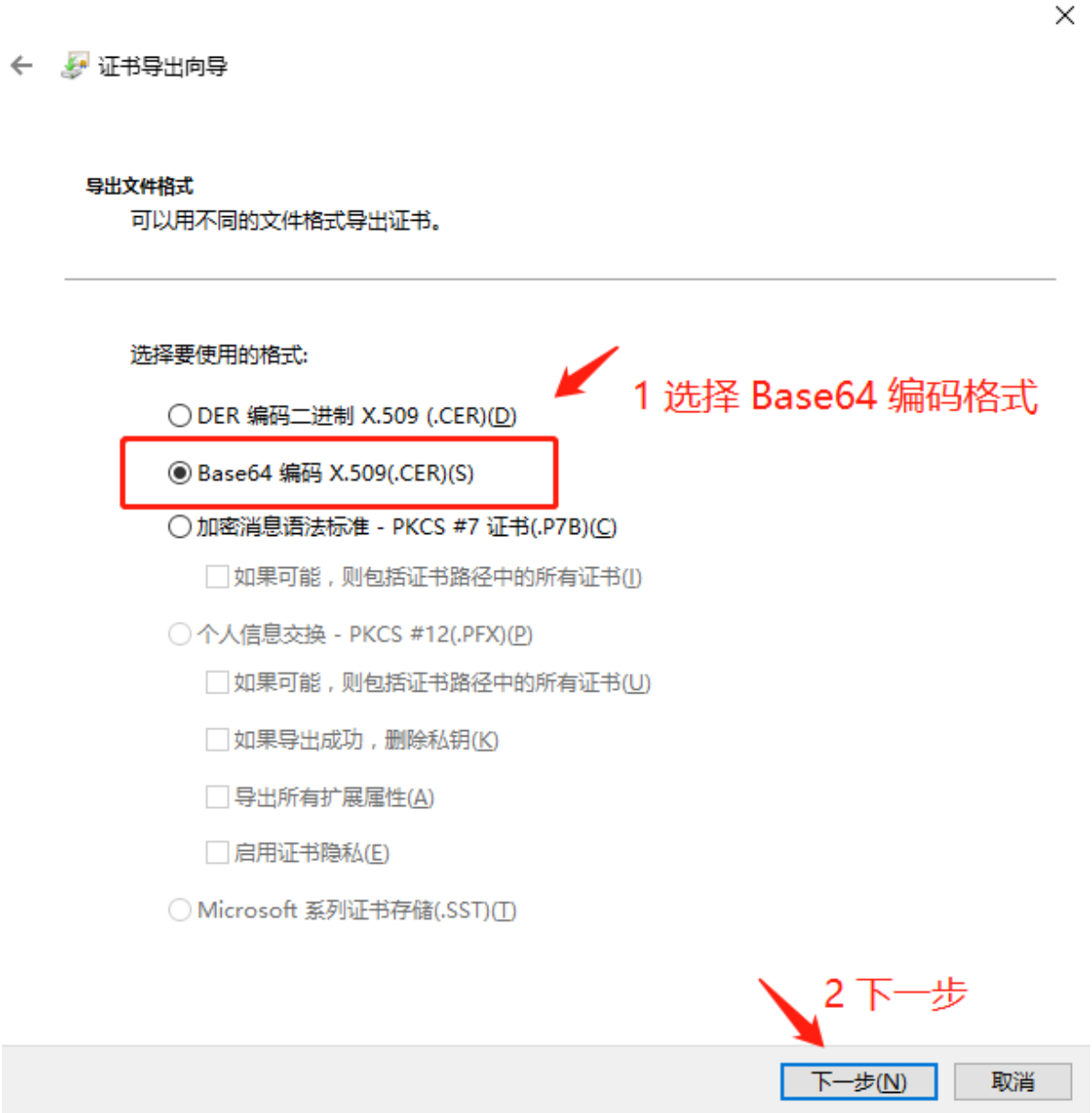

图 5.6: *选择根证书编码格式*

– 选择证书存储位置

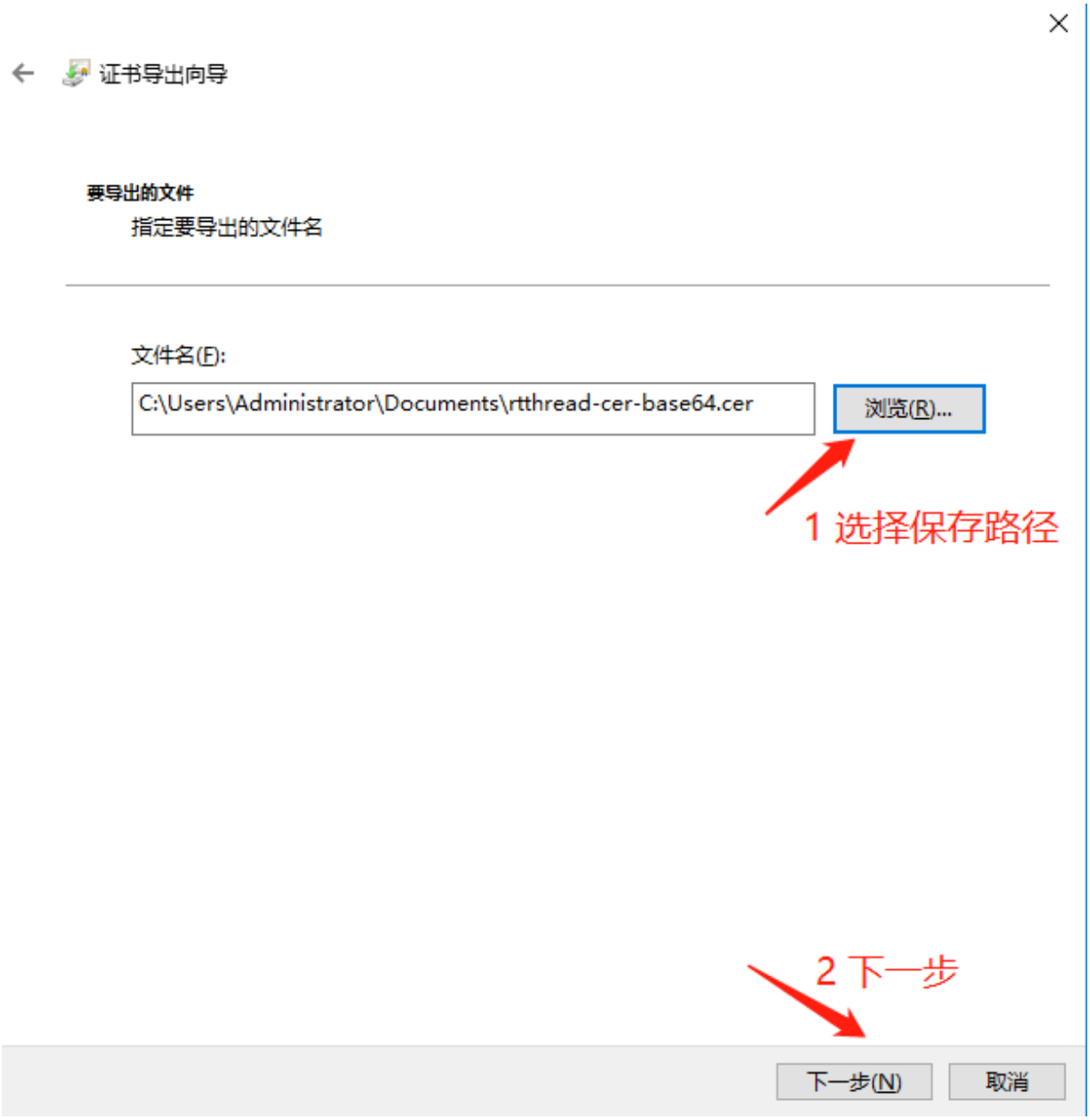

图 5.7: *选择根证书存储位置*

– 完成证书文件导出

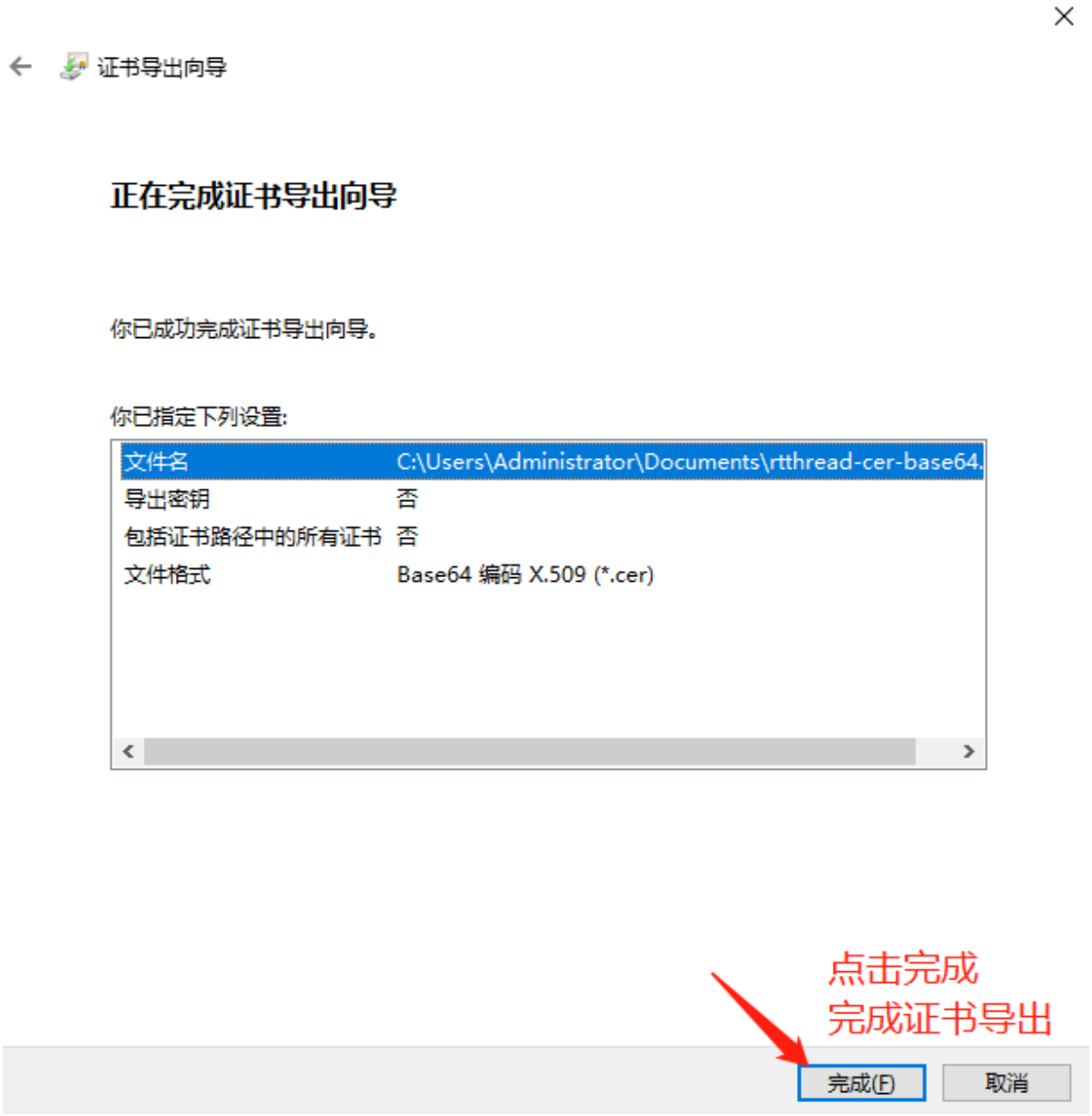

图 5.8: *完成根证书导出*

完成证书导出，假设证书文件名为 **USER_ROOT_CA.cer**。

5.11.3 导入证书

- 使用文本编辑器打开上个步骤导出的根证书文件 **USER_ROOT_CA.cer**
- 拷贝 **USER_ROOT_CA.cer** 文件到 `certs` 根目录
- 使用 `scons` 命令重新编译

注：

`scons` 命令编译后，会自动将证书文件拷贝到 `const char mbedtls_root_certificate[]` 数组中。

5.12 常见问题

5.12.1 证书验证失败

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

- 原因

  mbedtls 包中支持多种主流 CA 机构根证书，部分 CA 机构未支持

- 解决方法

  若测试其他 TLS 网站证书验证失败，手动获取测试网站根证书（Root Cerificate）添加到`mbedtls/tls_cerificate.c`文件中

5.12.2 证书时间错误

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

- 原因

  TLS 握手是证书验证需要时间的验证，本地时间获取有误导致

- 解决方式

  检查 RTC 设备是否支持，检查 `RT_USING_RTC` 宏是否打开，校准设备时间。建议使用 NTP 同步本地时间。

5.12.3 证书 CN 错误

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

- 原因

  测试其他 TLS 网站时，若输入域名不符合证书的 Common Name（CN）出现 CN 验证失败问题

- 解决方法

  检查输入域名和证书中 CN 是否匹配或输入 IP 地址

5.12.4 0x7200 错误

- 原因

  部分原因是因为 mbedTls 收到了大于缓冲区大小的数据包

- 解决方法

  menuconfig 配置增加数据帧大小（Maxium fragment length in bytes）

```
RT-Thread online packages --->
    security packages  --->
            Select Root Certificate  --->      # 选择证书文件
        [*] mbedtls: An portable and flexible SSL/TLS library  ---
        [*]   Store the AES tables in ROM
        (2)   Maximum window size used
        (6144) Maxium fragment length in bytes # 配置数据帧大小（0x7200 错误可尝试增
            加该大小）
        [*]   Enable a mbedtls client example
              version (latest)  --->
```

5.13 参考

- mbedTLS 官方网站：https://tls.mbed.org/
- ARMmbed GitHub：mbedtls

第 6 章

MbedTLS RAM 和 ROM 资源占用优化指南

mbedtls 软件包采用了模块化的设计，可以使用 `config.h` 文件来进行功能模块的配置选择。

`mbedtls` 默认提供的 `config.h` 文件是一个通用的、全功能的配置，占用了非常大的 RAM 和 ROM 空间，但是保证了 SSL 握手和通讯的建立速度、稳定性、协议兼容性以及数据传输效率。但嵌入式设备受限于其有限的 RAM 和 ROM 空间，我们不得不牺牲速度来节省 RAM 空间，裁剪不需要的功能模块来降低 ROM 占用。

本优化指南，在保证 SSL/TLS 客户端能与服务器建立安全稳定连接的前提下，对 RAM 和 ROM 占用进行优化统计。

注意：

mbedtls 客户端的优化属于针对性优化，针对特定的 SSL/TLS 服务器进行的优化，不同的 SSL/TLS 服务器配置不同，优化所用到的配置参数也是不同的。

因此，开发者在进行 SSL/TLS 优化前，在 MCU 资源条件允许的情况下，请先使用默认的配置调通 SSL/TLS 握手连接和加密通讯，然后再根据 SSL/TLS 服务器具体的配置进行逐项优化。

当然，多数情况下您并不知道服务器的具体参数配置，因此也只能试探性优化，本文给出了各个配置的说明，来方便开发者进行针对性的优化。

6.1 优化说明

- RAM 资源占用统计说明

  首先保证 SSL 握手连接正常，加密数据通讯正常。运行 `tls_test` 测试例程，进行 RAM 优化测试。测试例程在单独的线程中运行，通过对比 SSL 握手成功前后所占用的内存来确定在握手通讯过程所使用的 RAM 情况。该测试方法只能粗略估计 SSL 客户端成功进行握手连接所需要的 RAM 大小，该数据包含了保证握手通讯所需要的额外的 RAM 空间。

- ROM 资源占用统计说明

  通过对比启动 **mbedtls** 功能组件前后参与链接的文件来统计 **mbedtls** 所占用 ROM 大小。

- 测试平台：iMXRT1052
- 测试 IDE：MDK5
- 优化级别：o2
- 测试例程：samples/tls_app_test.c
- 测试使用的 SSL 服务器：www.rt-thread.org
- 测试服务器根证书签名算法：sha1RSA
- 测试服务器根证书签名哈希算法：sha1
- 测试服务器根证书公钥：RSA 2048 bits
- 测试服务器根证书指纹算法：sha1
- SSL 客户端指定密码套件

```
#define MBEDTLS_SSL_CIPHERSUITES                        \
    MBEDTLS_TLS_RSA_WITH_AES_256_CBC_SHA256
```

- SSL 客户端指定帧大小为 ##define MBEDTLS_SSL_MAX_CONTENT_LEN 3584
- 测试使用的配置（**详见文末**）

6.2 优化后的资源占用汇总

- 默认的 tls_config.h 配置资源占用情况

  mbedtls 默认的配置文件为 mbedtls/include/mbedtls/config.h，而 **RT-Thread** 使用的配置文件为 ports/inc/tls_config.h。用户进行配置优化的时候也是使用的 ports/inc/tls_config.h 文件。

```
RO(CODE + RO)        : 159828 bytes（156.08K）
RW(RW + ZI)          :    720 bytes
ROM(CODE + RO + RW) : 159972 bytes（156.22K）
动态内存使用          :  26849 bytes（26.22K）（包含 1K 的测试 buffer）
```

- 优化后的配置资源占用情况

```
RO(CODE + RO)        : 71893 bytes（70.21K）
RW(RW + ZI)          :    82 bytes
ROM(CODE + RO + RW) : 71975 bytes（70.29K）
动态内存使用          : 23344 bytes（22.79K）（包含 1K 的测试 buffer）
```

6.3 优化前的准备

1. 首先你要有准备接入的 SSL 服务器（保证能正常工作）
2. 准备好接入 SSL 服务器的 PEM 格式根证书文件（存放到 mbedtls 软件包 `certs` 目录下，删除其他不需要的证书）
3. 使用默认的 mbedtls 配置文件成功接通 SSL 服务器（优化后更难定位失败原因）
4. 逐项优化 mbedtls 配置，反复测试

注意：

如果您的 MCU 资源比较小，无法使用默认的 `tls_config.h` 配置文件，开发者可以选择使用 `QEMU` 虚拟机进行开发调试以及 mbedtls 优化。将 mbedtls 资源占用优化到合适的时候，再使用您需要的 MCU 进行验证测试。

6.4 优化配置概述

6.4.1 常用优化配置

通过对下面列表中的配置进行修改，可以很大程度上降低 mbedtls RAM 和 ROM 的占用。

开发者在进行优化时，建议优先对下面列表中的配置进行优化，如果不能满足要求，再针对其他的配置进行逐项的优化。

配置	说明	优化建议
const char mbedtls_root_certificate[ ]	存储根证书的常量数组。编译的时候，会将 PEM 证书添加到该数组。建议只在 `certs` 证书目录存放需要的根证书文件，否则会占用非常大的 RAM 和 ROM 空间	只存放需要的证书文件
MBEDTLS_SSL_CIPHERSUITES	通过指定密码套件来节省几百字节的 ROM 和几百字节的 RAM。这里注意需要指定服务器支持的加密套件，并为该加密套件启用相关的功能组件，关闭其他功能组件。如果只接入一个 SSL 服务器，通常这里只需要定义支持一个加密套件即可	仅指定根证书需要的加密套件
MBEDTLS_AES_ROM_TABLES	将 AES 表存储在 ROM 中以节省 RAM 占用（很大程度上降低 RAM 占用）	建议启用

配置	说明	优化建议
MBEDTLS_SSL_MAX_CONTENT_LEN	默认为 16384。RFC 定义了 SSL/TLS 消息的默认大小，如果您在此处更改值，则其他客户端/服务器可能无法再与您通信。除非你能确定服务端的帧大小。根据服务器发送的最大帧大小做适当的修改	适当调小（出现 0x7200 错误时请增大该配置）
MBEDTLS_MPI_MAX_SIZE	可用的 MPI 最大字节数，默认为 1024，可以根据适当调小	适当调小
MBEDTLS_MPI_WINDOW_SIZE	MPI 用于模幂运算的最大窗口数量，默认为 6，选值范围：1-6，可适当调小	适当调小
MBEDTLS_ECP_MAX_BITS	GF(p) 椭圆曲线最大位，默认为 521	适当调小
MBEDTLS_ECP_WINDOW_SIZE	用于点乘的最大窗口大小，默认为 6，选值范围：2-7，可以适当减小，减小会影响速度	适当调小
MBEDTLS_ECP_FIXED_POINT_OPTIM	默认 1，启用定点加速。启用后，将加速点乘运算大约 3 到 4 倍，成本是峰值内存占用增加约 2 倍。可以配置为 0，牺牲速度来节省 RAM 占用	可优化配置为 0
MBEDTLS_ECP_NIST_OPTIM	为每个 NIST 启用特定的实例，使相应曲线上的操作快 4 到 8 倍，缺点是 ROM 占用大。可以选择性优化	可禁用
MBEDTLS_ENTROPY_MAX_SOURCES	最大的熵源数量，最小为 2，默认使用 `mbedtls_platform_entropy_poll` 源。RT-Thread 上使用最小配置 2	可优化配置为 2

6.4.2 系统相关配置

这部分配置跟具体的系统和编译器相关，下表列出了在 **RT-Thread** 上需要做的配置。

配置	说明	优化建议
MBEDTLS_HAVE_ASM	需要编译器可以处理汇编代码	启用
MBEDTLS_HAVE_TIME	如果您的系统没有 time.h 和 time() 函数，请注释该配置	启用

配置	说明	优化建议
MBEDTLS_HAVE_TIME_DATE	如果您的系统没有 time.h、time()、gmtime() 或者没有正确的时钟，请注释该配置	启用
MBEDTLS_DEBUG_C	定义该配置以启动调试 log 输出	如需调试 log，则启用，否则请禁用
MBEDTLS_NET_C	该配置仅支持 POSIX/Unix 和 windows 系统，在 RT-Thread 系统上需要关闭	禁用
MBEDTLS_NO_PLATFORM_ENTROPY	如果您的平台不支持 /dev/urandom 或 Windows CryptoAPI 等标准，则需要启用该配置。RT-Thread 上必须启用	启用
MBEDTLS_TIMING_C	如果注释，则需要用户自己实现相关的函数。默认启用	启用
MBEDTLS_TIMING_ALT	如果注释，则需要用户自己实现相关的函数。默认启用	启用
MBEDTLS_ENTROPY_HARDWARE_ALT	如果注释，则需要用户自己实现相关的函数。默认启用	启用
MBEDTLS_ENTROPY_C （依赖：MBEDTLS_SHA256_C 或 MBEDTLS_SHA512_C）	启用特定于平台的熵代码。需要启用	启用
MBEDTLS_PADLOCK_C （依赖：MBEDTLS_HAVE_ASM）	在 x86 上启用 VIA Padlock 支持	禁用
MBEDTLS_AESNI_C （依赖：MBEDTLS_HAVE_ASM）	在 x86-64 上启用 AES-NI 支持	禁用
MBEDTLS_PLATFORM_C	使能平台抽象层，用于重定义实现 free、printf 等函数	启用

6.4.3 功能组件相关配置

用户可以根据要接入的 SSL/TLS 服务器特性以及根证书使用的签名算法来选择启用哪部分的功能。启用或禁用功能组件时请注意将相关的依赖打开或禁用。

配置	说明	优化建议
MBEDTLS_ASN1_PARSE_C	使能通用的 ASN1 解析器。ASN1: 一种描述数字对象的方法和标准，需要启用	启用
MBEDTLS_ASN1_WRITE_C	启用通用 ASN1 编写器	启用
MBEDTLS_BIGNUM_C	启用大整数库（multi-precision integer library）	启用
MBEDTLS_CIPHER_C	启用通用密码层	启用
MBEDTLS_AES_C	启用 AES 加密。PEM_PARSE 使用 AES 来解密被加密的密钥。通过启用 AES 来支持 *_WITH_AES_* 类型的加密套件	启用
MBEDTLS_CTR_DRBG_C （依赖：MBEDTLS_AES_C）	启用基于 CTR_DRBG AES-256 的随机生成器	启用
MBEDTLS_MD_C	启用通用消息摘要层，需要启用	启用
MBEDTLS_OID_C	启用 OID 数据库，此模块在 OID 和内部值之间进行转换，需要启用	启用
MBEDTLS_PK_C （依赖：MBEDTLS_RSA_C、MBEDTLS_ECP_C）	启用通用公共（非对称）密钥层，需要启用	启用
MBEDTLS_PK_PARSE_C （依赖：MBEDTLS_PK_C）	启用通用公共（非对称）密钥解析器，需要启用	启用
MBEDTLS_SHA256_C	启用 SHA-224 和 SHA-256 加密哈希算法，根据根证书详细信息中的签名哈希算法进行选择	根据需要选择
MBEDTLS_SHA512_C	启用 SHA-384 和 SHA-512 加密哈希算法，根据根证书详细信息中的签名哈希算法进行选择	根据需要选择
MBEDTLS_SSL_CLI_C （依赖：MBEDTLS_SSL_TLS_C）	启用 SSL 客户端代码，作为 SSL 服务端的时候不需要启用	启用
MBEDTLS_SSL_SRV_C （依赖：MBEDTLS_SSL_TLS_C）	启用 SSL 服务端代码，作为 SSL 客户端的时候不需要启用	禁用

配置	说明	优化建议
MBEDTLS_SSL_TLS_C （依赖：MBEDTLS_CIPHER_C、MBEDTLS_MD_C 和至少定义一个 MBEDTLS_SSL_PROTO_XXX）	使能 SSL/TLS 代码	启用
MBEDTLS_X509_CRT_PARSE_C （依赖：MBEDTLS_X509_USE_C）	使能 X509 证书解析	启用
MBEDTLS_X509_USE_C （依赖：MBEDTLS_ASN1_PARSE_C、MBEDTLS_BIGNUM_C、MBEDTLS_OID_C、MBEDTLS_PK_PARSE_C）	启用 X.509 核心以使用证书	启用
MBEDTLS_BASE64_C	启用 base64 组件，PEM 证书解析需要使用	启用
MBEDTLS_CERTS_C	该模块用于测试 SSL 客户端和服务器，可以选择禁用	可禁用
MBEDTLS_PEM_PARSE_C（依赖：MBEDTLS_BASE64_C）	启用对 PEM 文件解码解析的支持	启用
MBEDTLS_RSA_C （依赖：MBEDTLS_BIGNUM_C、MBEDTLS_OID_C）	启用 RSA 公钥密码系统。RSA、DHE-RSA、ECDHE-RSA、RSA-PSK 方式的密钥交换需要使用	启用
MBEDTLS_SHA1_C	启用 SHA1 加密哈希算法。TLS 1.1/1.2 需要使用	启用
MBEDTLS_MD5_C	启用 MD5 哈希算法。PEM 解析需要使用	启用
MBEDTLS_PK_PARSE_EC_EXTENDED （依赖：）	该宏用以支持 RFC 5915 和 RFC 5480 不允许的 SEC1 变体增强对读取 EC 密钥的支持	可以禁用
MBEDTLS_ERROR_STRERROR_DUMMY	启用虚拟错误功能，以便在禁用 MBEDTLS_ERROR_C 时更容易在第三方库中使用 mbedtls_strerror ()（启用 MBEDTLS_ERROR_C 时无效）	可以禁用
MBEDTLS_GENPRIME （依赖：MBEDTLS_BIGNUM_C）	启用素数生成代码	可以禁用

配置	说明	优化建议
MBEDTLS_FS_IO	启用文件系统交互相关的功能函数	可以禁用
MBEDTLS_PKCS5_C （依赖：MBEDTLS_MD_C）	该模块增加了对 PKCS # 5 功能的支持。AES 算法数据填充方案的需要。根据需要选择是否禁用	根据需要选择
MBEDTLS_PKCS12_C （依赖：MBEDTLS_ASN1_PARSE_C、MBEDTLS_CIPHER_C、MBEDTLS_MD_C）	添加用于解析 PKCS # 8 加密私钥的算法	可以禁用
MBEDTLS_PKCS1_V15（依赖：MBEDTLS_RSA_C）	用于支持 PKCS # 1 v1.5 操作，RSA 密钥套件需要使用。如果使用了 RSA 密钥套件，则需要启用	根据需要选择
MBEDTLS_PKCS1_V21 （依赖：MBEDTLS_MD_C、MBEDTLS_RSA_C）	启用对 PKCS # 1 v2.1 编码的支持，这样可以支持 RSAES-OAEP 和 RSASSA-PSS 操作	可以禁用
MBEDTLS_PK_RSA_ALT_SUPPORT	支持 PK 层中的外部私有 RSA 密钥（例如，来自 HSM）。不需要启用，禁用	禁用
MBEDTLS_SELF_TEST	启用检查功能。建议在启用 debug 的时候启用，其他时候禁用	可以禁用
MBEDTLS_SSL_ALL_ALERT_MESSAGES	启用警报消息发送功能	可以禁用
MBEDTLS_SSL_ENCRYPT_THEN_MAC	启用对 Encrypt-then-MAC，RFC 7366 的支持。用于加强对 CBC 密码套件的保护，可以禁用	可以禁用
MBEDTLS_SSL_EXTENDED_MASTER_SECRET	启用对扩展主密钥的支持	可以禁用
MBEDTLS_SSL_FALLBACK_SCSV	注释此宏以禁用客户端使用回退策略	可以禁用
MBEDTLS_SSL_CBC_RECORD_SPLITTING	在 SSLv3 和 TLS 1.0 中为 CBC 模式启用 1/n-1 记录拆分。启用该宏以降低 BEAST 攻击的风险，可以选择性禁用	可以禁用

配置	说明	优化建议
MBEDTLS_SSL_RENEGOTIATION	屏蔽该宏禁用对 TLS 重新协商的支持。启用可能会带来安全风险，建议禁用	禁用
MBEDTLS_SSL_MAX_FRAGMENT_LENGTH	在 SSL 中启用对 RFC 6066 最大帧长度扩展的支持	可以禁用
MBEDTLS_SSL_ALPN	启用对 RFC 7301 应用层协议协商的支持	可以禁用
MBEDTLS_SSL_SESSION_TICKETS	在 SSL 中启用对 RFC 5077 会话 tickets 的支持，需要服务端支持。通常用来优化握手流程	可以禁用
MBEDTLS_SSL_EXPORT_KEYS	启用对导出密钥块和主密钥的支持。这对于 TLS 的某些用户是必需的，例如 EAP-TLS	可以禁用
MBEDTLS_SSL_SERVER_NAME_INDICATION （依赖：MBEDTLS_X509_CRT_PARSE_C）	在 SSL 中启用对 RFC 6066 服务器名称指示（SNI）的支持	可以禁用
MBEDTLS_SSL_TRUNCATED_HMAC	在 SSL 中启用对 RFC 6066 截断 HMAC 的支持	可以禁用
MBEDTLS_VERSION_FEATURES （依赖：MBEDTLS_VERSION_C）	版本功能信息相关	可以禁用
MBEDTLS_VERSION_C	该模块提供运行时版本信息	可以禁用
MBEDTLS_X509_CHECK_KEY_USAGE	启用 keyUsage 扩展（CA 和叶证书）的验证。禁用此功能可避免错误发布和/或误用（中间）CA 和叶证书的问题。注释后跳过 keyUsage 检查 CA 和叶证书	可以禁用
MBEDTLS_X509_CHECK_EXTENDED_KEY_USAGE	启用 extendedKeyUsage 扩展（叶证书）的验证。禁用此功能可避免错误发布和/或误用证书的问题	可以禁用

配置	说明	优化建议
MBEDTLS_X509_RSASSA_PSS_SUPPORT	启用使用 RSASSA-PSS（也称为 PKCS # 1 v2.1）签名的 X.509 证书，CRL 和 CSRS 的解析和验证。根据需要选择是否禁用	根据需要选择
MBEDTLS_BLOWFISH_C	启用 Blowfish 分组密码	可以禁用
MBEDTLS_ERROR_C	启用错误代码到错误字符串的转换	可以禁用
MBEDTLS_HMAC_DRBG_C （依赖：MBEDTLS_MD_C）	启用随机字节发生器	可以禁用
MBEDTLS_PEM_WRITE_C （依赖：MBEDTLS_BASE64_C）	此模块添加了对编码/写入 PEM 文件的支持。TLS Client 不需要	可以禁用
MBEDTLS_PK_WRITE_C（依赖：MBEDTLS_PK_C）	启用通用公钥写入功能。嵌入式系统一般不需要，禁用	禁用
MBEDTLS_RIPEMD160_C	RIPEMD (RACE 原始完整性校验讯息摘要) 是一种加密哈希函数，通用性差于 SHA-1/2	可以禁用
MBEDTLS_SSL_CACHE_C	启用 SSL 缓存	可以禁用
MBEDTLS_SSL_TICKET_C（依赖：MBEDTLS_CIPHER_C）	服务端的配置	禁用
MBEDTLS_X509_CRL_PARSE_C（依赖：MBEDTLS_X509_USE_C）	CRL: Certidicate Revocation List (CRL) 证书吊销列表模块	可以禁用
MBEDTLS_X509_CSR_PARSE_C （依赖：MBEDTLS_X509_USE_C）	Certificate Signing Request (CSR). 证书签名请求解析，用于 DER 证书	可以禁用
MBEDTLS_X509_CREATE_C （依赖：MBEDTLS_BIGNUM_C、MBEDTLS_OID_C、MBEDTLS_PK_WRITE_C）	启用 X.509 核心以创建证书，服务器需要	禁用
MBEDTLS_X509_CRT_WRITE_C （依赖：MBEDTLS_X509_CREATE_C）	启用创建证书，服务器需要。禁用	
MBEDTLS_X509_CSR_WRITE_C （依赖：MBEDTLS_X509_CREATE_C）	启用创建 X.509 证书签名请求（CSR）	可以禁用
MBEDTLS_XTEA_C	启用 XTEA 分组密码	可以禁用

配置	说明	优化建议
MBEDTLS_ECDSA_DETERMINISTIC （依赖：MBEDTLS_HMAC_DRBG_C）	启用确定性 ECDSA（RFC 6979），防止签名时缺少熵而导致签名密钥泄露。建议启用	建议启用

6.4.4 密码套件相关配置

mbedtls 中密码套件命名形式为 `MBEDTLS_TLS_PSK_WITH_AES_256_GCM_SHA384`。

mbedtls 在数组 `static const int ciphersuite_preference[]` 中定义了所有支持的密码套件，如果开启支持所有的密码套件将会占用非常大的 ROM 空间。这里建议用户通过 `MBEDTLS_SSL_CIPHERSUITES` 宏来指定客户端与服务器使用具体哪种加密套件。指定加密套件后，将不需要的加密套件和依赖的功能组件全部禁用，同时禁用不需要的椭圆曲线，来最大程度上节省 ROM 空间。

配置	说明	优化建议
MBEDTLS_SSL_CIPHERSUITES	通过指定密码套件来节省 ROM 和几百字节的 RAM。这里注意需要指定服务器支持的加密套件，并为该加密套件启用相关的功能组件，关闭其他功能组件。如果只接入一个 SSL 服务器，通常这里只需要定义支持一个加密套件即可	仅指定根证书需要的加密套件
MBEDTLS_AES_C	通过启用 AES 来支持 `*_WITH_AES_*` 类型的密码套件	根据需要选择
MBEDTLS_GCM_C （依赖：MBEDTLS_AES_C、MBEDTLS_CAMELLIA_C）	启用该配置来支持 `*_AES_GCM_*`、`*_CAMELLIA_GCM_*` 类型的密码套件	根据需要选择
MBEDTLS_REMOVE_ARC4_CIPHERSUITES	默认启用，在 SSL/TLS 中禁用 RC4 密码套件	根据需要选择

配置	说明	优化建议
MBEDTLS_ARC4_C	启用 RC4 加密套件，*_WITH_RC4* 类型密码套件。根据需要选择是否禁用	根据需要选择
MBEDTLS_CAMELLIA_C	启用 Camellia 分组密码，用于支持 * _WITH_CAMELLIA_ * 类型的密码套件。根据需要选择是否禁用	根据需要选择
MBEDTLS_CIPHER_MODE_CBC	为对称密码启用密码块链接模式（CBC）。如果使用了 CBC 密码套件则需要启用	根据需要选择
MBEDTLS_CIPHER_MODE_CFB	为对称密码启用密码反馈模式（CFB）。如果使用了 CFB 密码套件则需要启用	根据需要选择
MBEDTLS_CIPHER_MODE_CTR	启用对称密码的计数器分组密码模式（CTR）。如果使用了 CTR 密码套件则需要启用	根据需要选择
MBEDTLS_CIPHER_PADDING_XXX	在密码层中启用填充模式。如果禁用所有填充模式，则只有完整块可以与 CBC 一起使用	根据需要选择，可以全禁用
MBEDTLS_CIPHER_PADDING_PKCS7		可以禁用
MBEDTLS_CIPHER_PADDING_ONE_AND_ZEROS		可以禁用
MBEDTLS_CIPHER_PADDING_ZEROS_AND_LEN		可以禁用
MBEDTLS_CIPHER_PADDING_ZEROS		可以禁用
MBEDTLS_CIPHER_PADDING_XXX		

配置	说明	优化建议
MBEDTLS_KEY_EXCHANGE_ECDH_RSA_ENABLED （依赖：MBEDTLS_ECDH_C、MBEDTLS_X509_CRT_PARSE_C）	启用 *_ECDH_RSA_* 类型的密码套件	根据需要选择
MBEDTLS_KEY_EXCHANGE_ECDHE_RSA_ENABLED（依赖：MBEDTLS_ECDH_C、MBEDTLS_RSA_C、MBEDTLS_PKCS1_V15、MBEDTLS_X509_CRT_PARSE_C）	启用 *_ECDHE_RSA_* 类型密码套件	根据需要选择
MBEDTLS_KEY_EXCHANGE_ECDHE_ECDSA_ENABLED（依赖：MBEDTLS_ECDH_C、MBEDTLS_ECDSA_C、MBEDTLS_X509_CRT_PARSE_C）	启用 *_ECDHE_ECDSA_* 类型密码套件	根据需要选择
MBEDTLS_KEY_EXCHANGE_ECDH_ECDSA_ENABLED（依赖：MBEDTLS_ECDH_C、MBEDTLS_X509_CRT_PARSE_C）	启用 *_ECDHE_ECDSA_* 类型密码套件	根据需要选择
MBEDTLS_KEY_EXCHANGE_DHE_RSA_ENABLED （依赖：MBEDTLS_DHM_C、MBEDTLS_RSA_C、MBEDTLS_PKCS1_V15、MBEDTLS_X509_CRT_PARSE_C）	启用 *_DHE_RSA_* 类型密码套件	根据需要选择
MBEDTLS_KEY_EXCHANGE_PSK_ENABLED	启用 *_PSK_* 类型密码套件	根据需要选择
MBEDTLS_KEY_EXCHANGE_DHE_PSK_ENABLED（依赖：MBEDTLS_DHM_C）	启用 *_DHE_PSK_* 类型密码套件	根据需要选择
MBEDTLS_KEY_EXCHANGE_ECDHE_PSK_ENABLED（依赖：MBEDTLS_ECDH_C）	启用 *_ECDHE_PSK_* 类型密码套件	根据需要选择
MBEDTLS_KEY_EXCHANGE_RSA_PSK_ENABLED （依赖：MBEDTLS_RSA_C、MBEDTLS_PKCS1_V15、MBEDTLS_X509_CRT_PARSE_C）	启用 *_RSA_PSK_* 类型密码套件	根据需要选择
MBEDTLS_KEY_EXCHANGE_RSA_ENABLED （依赖：MBEDTLS_RSA_C、MBEDTLS_PKCS1_V15、MBEDTLS_X509_CRT_PARSE_C）	启用 *_RSA_* 类型密码套件	根据需要选择
MBEDTLS_CCM_C （依赖：MBEDTLS_AES_C 或 MBEDTLS_CAMELLIA_C）	启用具有 CBC-MAC（CCM）模式的计数器用于 128 位分组密码，用于支持 AES-CCM 密码套件。根据需要选择是否禁用	根据需要选择
MBEDTLS_DES_C	启用 DES 块密码	可以禁用

配置	说明	优化建议
MBEDTLS_DHM_C	启用 Diffie-Hellman-Merkle 模块，用于支持 DHE-RSA, DHE-PSK 密码套件。根据需要选择是否禁用	根据需要选择

6.4.5 椭圆曲线相关配置

用户成功选择了匹配的加密套件，并验证可以正常建立握手连接和加密通讯后，可以尝试将加密套件不需要的椭圆曲线禁用。

配置	说明	优化建议
MBEDTLS_ECDH_C （依赖：MBEDTLS_ECP_C）	启用椭圆曲线 Diffie-Hellman 库。用于支持 *_ECDHE_ECDSA_*、*_ECDHE_RSA_*、*_DHE_PSK_* 类型的密码套件	根据需要选择
MBEDTLS_ECDSA_C（依赖：MBEDTLS_ECP_C、MBEDTLS_ASN1_WRITE_C、MBEDTLS_ASN1_PARSE_C）	用于支持 *_ECDHE_ECDSA_* 类型的密码套件	根据需要选择
MBEDTLS_ECP_C（依赖：MBEDTLS_BIGNUM_C 和至少一个 MBEDTLS_ECP_DP_XXX_ENABLED）	启用 GF(p) 椭圆曲线	根据需要选择
MBEDTLS_ECP_XXXX_ENABLED	在椭圆曲线模块中启用特定的曲线。默认情况下启用所有支持的曲线。可以根据实际情况选择一个曲线即可	根据需要选择
MBEDTLS_ECP_DP_SECP192R1_ENABLED		根据需要选择
MBEDTLS_ECP_DP_SECP224R1_ENABLED		根据需要选择
MBEDTLS_ECP_DP_SECP256R1_ENABLED		根据需要选择
MBEDTLS_ECP_DP_SECP384R1_ENABLED		根据需要选择
MBEDTLS_ECP_DP_SECP521R1_ENABLED		根据需要选择

配置	说明	优化建议
MBEDTLS_ECP_DP_SECP192K1_ENABLED		根据需要选择
MBEDTLS_ECP_DP_SECP224K1_ENABLED		根据需要选择
MBEDTLS_ECP_DP_SECP256K1_ENABLED		根据需要选择
MBEDTLS_ECP_DP_BP256R1_ENABLED		根据需要选择
MBEDTLS_ECP_DP_BP384R1_ENABLED		根据需要选择
MBEDTLS_ECP_DP_BP512R1_ENABLED		根据需要选择
MBEDTLS_ECP_DP_CURVE25519_ENABLED		根据需要选择
MBEDTLS_ECP_XXXX_ENABLED		

6.4.6 TLS 版本选择相关配置

通常 SSL/TLS 服务器支持多种 TLS 协议版本，客户端则不需要支持所有的协议版本。因此在确定服务器支持的 TLS 协议版本后，可以禁用其他版本的协议。

配置	说明	优化建议
MBEDTLS_SSL_PROTO_TLS1 （依赖：MBEDTLS_MD5_C、MBEDTLS_SHA1_C）	启用对 TLS 1.0 版本的支持	根据需要选择
MBEDTLS_SSL_PROTO_TLS1_1（依赖：MBEDTLS_MD5_C、MBEDTLS_SHA1_C）	启用对 TLS 1.1 版本的支持	根据需要选择
MBEDTLS_SSL_PROTO_TLS1_2（依赖：MBEDTLS_SHA1_C 或者 MBEDTLS_SHA256_C 或者 MBEDTLS_SHA512_C）	启用对 TLS 1.2 版本的支持	根据需要选择

6.4.7 DTLS 相关配置

DTLS 是基于 UDP 的安全加密连接，目的是保障 UDP 通讯的数据安全。由于 UDP 本身不支持自动重传，且存在丢包问题，所以在进行握手连接的时候与 TLS 有些许不同，但两者重复了大部分的代码。因此可以通过下表中的配置来优化 DTLS 加密连接。

如果用户的系统中，不需要使用 DTLS，则可以将下表中的所有配置禁用。

配置	说明	优化建议
MBEDTLS_SSL_PROTO_DTLS（依赖：MBEDTLS_SSL_PROTO_TLS1_1 或 MBEDTLS_SSL_PROTO_TLS1_2）	启用 DTLS 功能，用于对 UDP 进行加密	如果不需要 DTLS 加密连接，则禁用
MBEDTLS_SSL_DTLS_ANTI_REPLAY（依赖：MBEDTLS_SSL_TLS_C、MBEDTLS_SSL_PROTO_DTLS）	启用对 DTLS 中的反重放机制的支持	可以禁用
MBEDTLS_SSL_DTLS_HELLO_VERIFY（依赖：MBEDTLS_SSL_PROTO_DTLS）	启用对 DTLS HelloVerifyRequest 的支持	需要开启
MBEDTLS_SSL_DTLS_CLIENT_PORT_REUSE（依赖：MBEDTLS_SSL_DTLS_HELLO_VERIFY）	为从同一端口重新连接的客户端启用服务器端支持，需要服务器特殊支持	可以禁用
MBEDTLS_SSL_DTLS_BADMAC_LIMIT（依赖：MBEDTLS_SSL_PROTO_DTLS）	启用支持 MAC 错误的记录限制	可以禁用
MBEDTLS_SSL_COOKIE_C	DTLS hello cookie 支持。非 DTLS 下可以禁用	可以禁用

6.5 参考

- mbedTLS 官方网站：https://tls.mbed.org/
- 测试时用的配置文件

```
/* tls_config.h*/
#ifndef MBEDTLS_CONFIG_H
#define MBEDTLS_CONFIG_H

#include <rtthread.h>

#if defined(_MSC_VER) && !defined(_CRT_SECURE_NO_DEPRECATE)
#define _CRT_SECURE_NO_DEPRECATE 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_X509_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         384
#define MBEDTLS_MPI_WINDOW_SIZE        2
#define MBEDTLS_ECP_MAX_BITS         384
#define MBEDTLS_ECP_WINDOW_SIZE        2
#define 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             3584
#define 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_DTLS_ANTI_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 */
```

第 7 章

API 说明

为了方便用户使用，这里列出了常用的 API，并给出了相关的使用说明。

注：更多详细 API 内容请参阅 **ARM mbedtls API 手册**。

7.1 应用层 API

应用层 API 是提供给用户在 App 中直接使用的 API，这部分 API 屏蔽了 mbedtls 内部具体的操作步骤，简化了用户使用。

7.1.1 mbedtls 初始化

```
int mbedtls_client_init(MbedTLSSession *session, void *entropy, size_t entropyLen);
```

mbedtls 客户端初始化函数，用于初始化底层网络接口、设置证书、设置 SSL 会话等。

参数	描述
session	入参，mbedtls 会话对象 MbedTLSSession
entropy	入参，mbedtls 熵字符串
entropyLen	入参，mbedtls 熵字符串长度
返回	**描述**
`= 0`	成功
`!0`	失败

7.1.2 配置 mbedtls 上下文

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

SSL 层配置，应用程序使用 `mbedtls_client_context` 函数配置客户端上下文信息，包括证书解析、设置主机名、设置默认 SSL 配置、设置认证模式（默认 MBEDTLS_SSL_VERIFY_OPTIONAL）等。

参数	描述
session	入参，mbedtls 会话对象 MbedTLSSession
返回	**描述**
= 0	成功
!0	失败

7.1.3 建立 SSL/TLS 连接

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

使用 `mbedtls_client_connect` 函数为 SSL/TLS 连接建立通道。这里包含整个的握手连接过程，以及证书校验结果。

参数	描述
session	入参，mbedtls 会话对象 MbedTLSSession
返回	**描述**
= 0	成功
!0	失败

7.1.4 读取数据

- 向加密连接写入数据

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

参数	描述
session	入参，mbedtls 会话对象 MbedTLSSession
buf	入参，待写入的数据缓冲区
len	入参，待写入的数据长度
返回	**描述**
= 0	成功
!0	失败

- 从加密连接读取数据

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

参数	描述
session	入参，mbedtls 会话对象 MbedTLSSession
buf	入参，mbedtls 读取内容的缓冲区
len	入参，mbedtls 待读取内容长度
返回	**描述**
= 0	成功
!0	失败

7.1.5 关闭 mbedtls 客户端

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

客户端主动关闭连接或者因为异常错误关闭连接，都需要使用 mbedtls_client_close 关闭连接并释放资源。

参数	描述
session	入参，mbedtls 会话对象 MbedTLSSession
返回	**描述**
= 0	成功
!0	失败

7.2 mbedtls 相关 API

7.2.1 设置调试级别

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

如果开启了 MBEDTLS_DEBUG_C，可以使用该函数设置调试级别，用于控制不同级别的调试日志输出。

参数	描述
threshold	入参，Debug 级别，默认为 0 没有调试日志
返回	**描述**
无	无

参数	描述

mbedtls 定义了 5 种调试级别，如下所示：

调试级别	描述
0	No debug
1	Error
2	State change
3	Informational
4	Verbose

7.2.2 初始化阶段相关 API

- **网络上下文初始化**

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

初始化 TLS 网络上下文，目前只有 fd 描述符。

参数	描述
ctx	入参，网络上下文对象
返回	**描述**
无	无

- **SSL 上下文初始化**

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

SSL 上下文初始化，主要是清空 SSL 上下文对象，为 SSL 连接做准备。

参数	描述
ssl	入参，SSL 上下文对象
返回	**描述**
无	无

- **初始化 SSL 配置**

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

SSL 配置初始化，主要是清空 SSL 配置结构体对象，为 SSL 连接做准备。

参数	描述
conf	入参，SSL 配置结构体对象
返回	**描述**
无	无

- **初始化 SSL 随机字节发生器**

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

清空 CTR_DRBG（SSL 随机字节发生器）上下文结构体对象，为 `mbedtls_ctr_drbg_seed` 做准备。

参数	描述
ctx	入参，CTR_DRBG 结构体对象
返回	**描述**
无	无

- **初始化 SSL 熵**

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

初始化 SSL 熵结构体对象。

参数	描述
ctx	入参，熵结构体对象
返回	**描述**
无	无

- **设置 SSL/TLS 熵源**

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

为 SSL/TLS 熵设置熵源，方便产生子种子。

参数	描述
ctx	入参，CTR_DRBG 结构体对象
f_entropy	入参，熵回调
p_entropy	入参，熵结构体（mbedtls_entropy_context）对象
custom	入参，个性化数据（设备特定标识符），可以为空
len	个性化数据长度
返回	**描述**
无	无

- **设置根证书列表**

```
void mbedtls_x509_crt_init( mbedtls_x509_crt *crt );
```

初始化根证书链表。

参数	描述
crt	入参，x509 证书结构体对象
返回	**描述**
无	无

- **解析根证书**

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

解释性地解析。解析 buf 中一个或多个证书并将其添加到根证书链接列表中。如果可以解析某些证书，则结果是它遇到的失败证书的数量。如果没有正确完成，则返回第一个错误。

根证书位于 `ports/src/tls_certificate.c` 文件 `mbedtls_root_certificate` 数组中。

参数	描述
chain	入参，x509 证书结构体对象
buf	入参，存储根证书的 buffer，`mbedtls_root_certificate` 数组
buflen	入参，存储根证书的 buffer 大小
返回	**描述**

参数	描述
无	无

- **设置主机名**

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

注意，这里设置的 hostname 必须对应服务器证书中的 common name，即 CN 字段。

- **加载默认的 SSL 配置**

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

使用前，需要先调用 mbedtls_ssl_config_init 函数初始化 SSL 配置结构体对象。

参数	描述
conf	入参，SSL 配置结构体对象
endpoint	入参，MBEDTLS_SSL_IS_CLIENT 或者 MBEDTLS_SSL_IS_SERVER
transport	入参，TLS: MBEDTLS_SSL_TRANSPORT_STREAM; DTLS: MBEDTLS_SSL_TRANSPORT_DATAGRAM
preset	入参，预定义的 MBEDTLS_SSL_PRESET_XXX 类型值，默认使用 MBEDTLS_SSL_PRESET_DEFAULT
返回	**描述**
无	无

- **设置证书验证模式**

```
void mbedtls_ssl_conf_authmode( mbedtls_ssl_config *conf, int authmode );
```

设置证书验证模式默认值:服务器上为 MBEDTLS_SSL_VERIFY_NONE,客户端上为 MBEDTLS_SSL_VERIFY_REQUIRED 或者 MBEDTLS_SSL_VERIFY_OPTIONAL（默认使用）。

MBEDTLS_SSL_VERIFY_OPTIONAL 表示证书验证失败也可以继续通讯。

参数	描述
conf	入参，SSL 配置结构体对象

参数	描述
authmode	入参，证书验证模式
返回	**描述**
无	无

- **设置验证对等证书所需的数据**

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

将受信的证书链配置到 SSL 配置结构体对象中。

参数	描述
conf	入参，SSL 配置结构体对象
ca_chain	入参，受信的 CA 证书链，存储在 MbedTLSSession 的成员对象 cacert 中
ca_crl	入参，受信的 CA CRLs，可为空
返回	**描述**
无	无

- **设置随机数生成器回调**

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

参数	描述
conf	入参，SSL 配置结构体对象
f_rng	入参，随机数生成器函数
p_rng	入参，随机数生成器函数参数
返回	**描述**
无	无

- **设置 SSL 上下文**

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

将 SSL 配置结构体对象设置到 SSL 上下文中。

参数	描述
ssl	入参，SSL 上下文结构体对象
conf	入参，SSL 配置结构体对象
返回	**描述**
= 0	成功
- 0x7F00	内存分配失败

7.2.3 连接阶段相关 API

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

与给定的 host、port 及 proto 协议建立网络连接。

参数	描述
ctx	入参，NET 网络配置结构体对象
host	入参，指定的待连接主机名
port	入参，指定的主机端口号
proto	入参，指定的协议类型， MBEDTLS_NET_PROTO_TCP 或者 MBEDTLS_NET_PROTO_UDP
返回	**描述**
= 0	成功
- 0x0042	socket 创建失败
- 0x0052	未知的主机名，DNS 解析失败
- 0x0044	网络连接失败

- 设置网络层读写接口

```
void mbedtls_ssl_set_bio( mbedtls_ssl_context *ssl,
                          void *p_bio,
                          mbedtls_ssl_send_t *f_send,
                          mbedtls_ssl_recv_t *f_recv,
                          mbedtls_ssl_recv_timeout_t *f_recv_timeout );
```

为网络层设置读写函数，被 `mbedtls_ssl_read` 和 `mbedtls_ssl_write` 函数调用。

- 对于 TLS，用户提供 f_recv 和 f_recv_timeout 其中之一即可，如果都有提供，默认使用 `f_recv_timeout` 回调
- 对于 DTLS，用户需要提供 `f_recv_timeout` 回调函数

参数	描述
ssl	入参，SSL 上下文结构体对象
p_bio	入参，socket 描述符
f_send	入参，网络层写回调函数
f_recv	入参，网络层读回调函数
f_recv_timeout	入参，网络层非阻塞带超时读回调函数
返回	**描述**
无	无

- **SSL/TLS 握手接口**

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

执行 SSL/TLS 握手操作。

参数	描述
ssl	入参，SSL 上下文结构体对象
返回	**描述**
`= 0`	成功
`- 0x6900`	SSL 客户端需要读取调用
`- 0x6880`	SSL 客户端需要写入调用
`- 0x6A80`	DTLS 客户端必须重试才能进行 hello 验证
其它	其它 SSL 指定的错误码

注意，如果您使用的是 DTLS，你需要单独处理 `- 0x6A80` 错误，因为它是预期的返回值而不是实际错误。

- **获取证书验证结果**

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

参数	描述
ssl	入参，SSL 上下文结构体对象
返回	**描述**
= 0	成功
- 1	返回结果不可用
其它	BADCERT_xxx 和 BADCRL_xxx 标志的组合，请参阅 x509.h

或者证书验证结果的 API 接口，具体的错误信息需要使用 mbedtls_x509_crt_verify_info 接口获取。

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

使用 mbedtls_x509_crt_verify_info 函数获取有关证书验证状态的信息字符串，存储在 MbedTLSSession 的对象的 buffer 成员中。

参数	描述
buf	入参，存储验证状态信息字符串的缓冲区
size	入参，缓冲区大小
prefix	入参，行前缀
flags	入参，由 mbedtls_x509_crt_verify_info 函数返回的值
返回	**描述**
整数	写入的字符串的长度（不包括结束符）或负的错误代码

7.2.4 读写 API

SSL/TLS 写函数

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

从 SLL/TLS 读取数据，最多读取 'len' 字节长度数据字节。

参数	描述
ssl	入参，SSL 上下文结构体对象
buf	入参，接收读取数据的缓冲区
len	入参，要读取的数据长度

参数	描述
返回	**描述**
> 0	读取到的数据长度
= 0	读取到结束符
- 0x6900	SSL 客户端需要读取调用
- 0x6880	SSL 客户端需要写入调用
- 0x6780	SSL 客户端需要重连
其它	其它 SSL 指定的错误码

SSL/TLS 读函数

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

向 SSL/TLS 写入数据，最多写入 'len' 字节长度数据。

参数	描述
ssl	入参，SSL 上下文结构体对象
buf	入参，待写入数据的缓冲区
len	入参，待写入数据的长度
返回	**描述**
> 0	实际写入的数据长度
= 0	读取到结束符
- 0x6900	SSL 客户端需要读取调用
- 0x6880	SSL 客户端需要写入调用
其它	其它 SSL 指定的错误码