Mbedtls流程与问题分析

博通集成电路(上海)有限公司

上海市浦东新区张江高科技园区张东路1387号41幢

电话: 86-21-51086811

[www.bekencorp.com](http://www.bekencorp.com)

# Mbedtls相关结构体

typedef struct MbedTLSSession

{

char\* host;

char\* port;

unsigned char \*buffer;

size\_t buffer\_len;

mbedtls\_ssl\_context ssl;

mbedtls\_ssl\_config conf;

mbedtls\_entropy\_context entropy;

mbedtls\_ctr\_drbg\_context ctr\_drbg;

mbedtls\_net\_context server\_fd;

#if USE\_CA\_CERTIFICATE\_EN

mbedtls\_x509\_crt cacert;

#endif

}MbedTLSSession;

# Mbedtls相关初始化函数

mbedtls\_net\_init(&session->server\_fd);//初始化fd

mbedtls\_ssl\_init(&session->ssl); //初始化ssl结构体

mbedtls\_ssl\_config\_init(&session->conf);//初始化mbedtls\_ssl\_config结构体

mbedtls\_ctr\_drbg\_init(&session->ctr\_drbg);//初始化随机字节生成器（CRT\_DRBG：使用分组密码算法的计数器模式作为随机数生成的基础算法）

mbedtls\_entropy\_init(&session->entropy);//熵源结构体的初始化（伪随机数所使用的种子）

#if USE\_CA\_CERTIFICATE\_EN

mbedtls\_x509\_crt\_init(&session->cacert);//x509的证书链结构体初始化，根据实际情况，可不使用根证书

#endif

#if CFG\_OUT\_PUT\_MBEDTLS\_DEBUG\_INFO

mbedtls\_ssl\_conf\_dbg( &session->conf, my\_debug, 0 );//设置打印调试

#endif

mbedtls\_ctr\_drbg\_seed(&session->ctr\_drbg, mbedtls\_entropy\_func, &session->entropy, (unsigned char \*)entropy, entropyLen));//设置熵源，方便产生子种子。

# 配置mbedtls\_client\_context

（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）:

mbedtls\_x509\_crt\_parse(&session->cacert, (const unsigned char \*)mbedtls\_root\_certificate, mbedtls\_root\_certificate\_len);//根证书解析，可根据实际情况，不使用根证书

mbedtls\_ssl\_set\_hostname(&session->ssl, session->host));//设置主机名

mbedtls\_ssl\_config\_defaults(&session->conf, MBEDTLS\_SSL\_IS\_CLIENT, MBEDTLS\_SSL\_TRANSPORT\_STREAM,

MBEDTLS\_SSL\_PRESET\_DEFAULT));//设置默认配置

mbedtls\_ssl\_conf\_max\_frag\_len( &session->conf, mfl\_code ) );//设置desired fragment length，默认1024（此处比较重要，是客户端与服务器在握手时去协商发送的最大片段，产生的结果是当数据量过大的时候会分包发送）

#if CFG\_USE\_CA\_CERTIFICATE\_VERIFY

mbedtls\_ssl\_conf\_authmode(&session->conf, MBEDTLS\_SSL\_VERIFY\_REQUIRED);//设置根证书验证模式默认值，MBEDTLS\_SSL\_VERIFY\_NONE：无验证

MBEDTLS\_SSL\_VERIFY\_REQUIRED：必须验证

MBEDTLS\_SSL\_VERIFY\_OPTIONAL：证书验证失败也可继续通讯

#endif

#if CFG\_USE\_CA\_CERTIFICATE

mbedtls\_ssl\_conf\_ca\_chain(&session->conf, &session->cacert, NULL);// **设置验证对等证书所需的数据,** 将受信的证书链配置到 SSL 配置结构体对象中

#endif

mbedtls\_ssl\_conf\_rng(&session->conf, mbedtls\_ctr\_drbg\_random, &session->ctr\_drbg);// 设置随机数生成器回调

mbedtls\_ssl\_setup(&session->ssl, &session->conf));//设置ssl上下文，将SSL 配置结构体对象设置到 SSL 上下文

# Mbedtls连接

mbedtls\_net\_connect(&session->server\_fd, session->host,

session->port, MBEDTLS\_NET\_PROTO\_TCP));与指定的host、port 及 proto 协议建立网络连接

mbedtls\_ssl\_set\_bio(&session->ssl, &session->server\_fd, mbedtls\_net\_send, mbedtls\_net\_recv, NULL);// **设置网络层读写接口，如** mbedtls\_ssl\_read 和 mbedtls\_ssl\_write

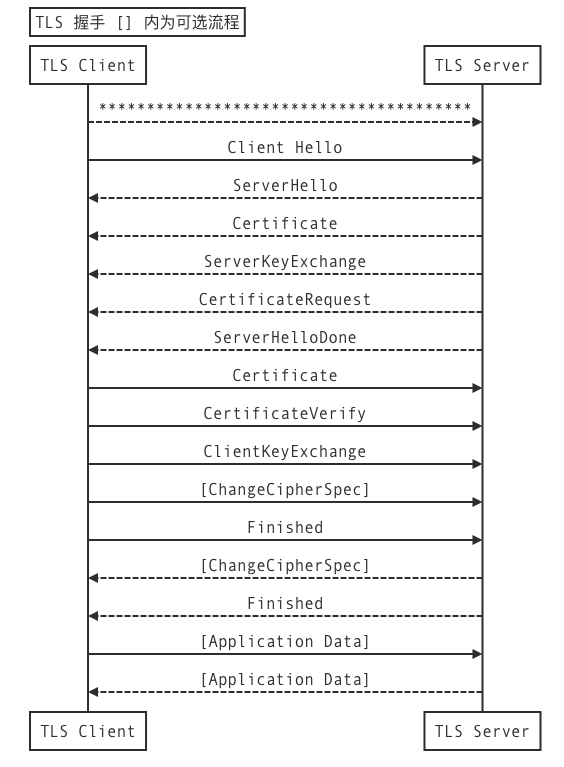
mbedtls\_ssl\_handshake(&session->ssl);//mbedtls握手

mbedtls\_ssl\_get\_verify\_result(&session->ssl));//获取根证书认证结果

mbedtls\_x509\_crt\_verify\_info((char \*)session->buffer, session->buffer\_len, " ! ", ret);//如果根证书认证结果失败，则通过此函数得到错误信息

# Mbedtls握手流程

流程图如下图所示



Mbedtls握手流程图

## Client Hello

Source:ssl\_cli.c

Func:ssl\_write\_client\_hello

Wireshark协议包：

Handshake Protocol: Client Hello

Handshake Type: Client Hello (1)

Length: 295

Version: TLS 1.2 (0x0303)

Random: 5f2933702428c5b2819c30feb9fc52b3483feedd7faf64c2…

GMT Unix Time: Aug 4, 2020 18:07:44.000000000 CST

Random Bytes: 2428c5b2819c30feb9fc52b3483feedd7faf64c22162e198…

Session ID Length: 0

Cipher Suites Length: 160

Cipher Suites (80 suites)

Compression Methods Length: 1

Compression Methods (1 method)

Extensions Length: 94

Extension: server\_name (len=18)

Extension: signature\_algorithms (len=22)

Extension: supported\_groups (len=24)

Extension: ec\_point\_formats (len=2)

Extension: encrypt\_then\_mac (len=0)

Extension: extended\_master\_secret (len=0)

Extension: session\_ticket (len=0)

数据分析：

Version: TLS 版本信息;

Random: 前四位：时间戳，后28位：随机数;

Cipher Suites: 客户端支持的组件;

## Server Hello

Source:ssl\_cli.c

Func: ssl\_parse\_server\_hello

Wireshark协议包：

Handshake Protocol: Server Hello

Handshake Type: Server Hello (2)

Length: 81

Version: TLS 1.2 (0x0303)

Random: 5f29336af1104faf18544577413080cb7f8932fd5c4d2249…

GMT Unix Time: Aug 4, 2020 18:07:38.000000000 CST

Random Bytes: f1104faf18544577413080cb7f8932fd5c4d2249fb8db872…

Session ID Length: 32

Session ID: 5f29336a02f78561926108084594d22cd6804819da2d9bd9…

Cipher Suite: TLS-RSA-WITH-AES-256-CBC-SHA256 (0x9c)

Compression Method: null (0)

Extensions Length: 9

Extension: renegotiation\_info (len=1)

Type: renegotiation\_info (65281)

Length: 1

Renegotiation Info extension

Extension: extended\_master\_secret (len=0)

Type: extended\_master\_secret (23)

Length: 0

数据分析：

Version: TLS 版本信息;

Random: 随机数序列，前四位：时间戳，后28位：随机数;

Cipher Suite: 服务端与客户端协商之后的密码套件;如TLS-RSA-WITH-AES-256-CBC-SHA256 (0x9c)

## Certificate

Source:ssl\_cli.c

Func: mbedtls\_ssl\_parse\_certificate

Wireshark协议包：

Handshake Protocol: Certificate

Handshake Type: Certificate (11)

Length: 484

Certificates Length: 481

Certificates (481 bytes)

Certificate Length: 478

Certificate: 308201da30820180021475bda3ed4a9e4fcba43686b12069… (id-at-commonName=svr.example.iwall.com,id-at-organizationalUnitName=SERVER,id-at-organizationName=IWall,id-at-localityName=HaiDian,id-at-stateOrProvinceName=BeiJing,id-at-co

signedCertificate

algorithmIdentifier (ecdsa-with-SHA256)

Padding: 0

encrypted: 30450221009499d98c1ea5963eaba85216b98de2ff45f5af…

数据分析：

Version: TLS 版本信息;

Certificates Length: 证书长度

Certificates: 证书信息;

## Server Key Exchange

Source:ssl\_cli.c

Func: ssl\_parse\_server\_key\_exchange

Wireshark协议包：

Handshake Protocol: Server Key Exchange

Handshake Type: Server Key Exchange (12)

Length: 211

EC Diffie-Hellman Server Params

Curve Type: named\_curve (0x03)

Named Curve: secp521r1 (0x0019)

Pubkey Length: 133

Pubkey: 04010a9d9275a1702a5eb822432c2f728e18613768c4818a…

Signature Algorithm: ecdsa\_secp521r1\_sha512 (0x0603)

Signature Hash Algorithm Hash: SHA512 (6)

Signature Hash Algorithm Signature: ECDSA (3)

Signature Length: 70

Signature: 3044022031c64f7e6fe642e6729bdf1822e1345e24858e13…

数据分析：

EC Diffie-Hellman Server Params

Pubkey: 若使用 ECDH，客户端使用证书中的服务器公钥，服务端不发送 Server Key Exchange。

若使用 ECDHE，服务器将通过 Server Key Exchange 消息告之客户端临时 ECDH 公钥，并使用服务器私钥，对该临时公钥进行签名。

Signature Algorithm: 签名算法

Signature: 服务器私钥签名

## Certificate Request

Source:ssl\_cli.c

Func: ssl\_parse\_certificate\_request

Wireshark协议包：

Handshake Protocol: Certificate Request

Handshake Type: Certificate Request (13)

Length: 141

Certificate types count: 3

Certificate types (3 types)

Certificate type: RSA Sign (1)

Certificate type: DSS Sign (2)

Certificate type: ECDSA Sign (64)

Signature Hash Algorithms Length: 20

Signature Hash Algorithms (10 algorithms)

Signature Algorithm: ecdsa\_secp521r1\_sha512 (0x0603)

Signature Algorithm: rsa\_pkcs1\_sha512 (0x0601)

Signature Algorithm: ecdsa\_secp384r1\_sha384 (0x0503)

Signature Algorithm: rsa\_pkcs1\_sha384 (0x0501)

Signature Algorithm: ecdsa\_secp256r1\_sha256 (0x0403)

Signature Algorithm: rsa\_pkcs1\_sha256 (0x0401)

Signature Algorithm: SHA256 DSA (0x0402)

Signature Algorithm: ecdsa\_sha1 (0x0203)

Signature Algorithm: rsa\_pkcs1\_sha1 (0x0201)

Signature Algorithm: SHA1 DSA (0x0202)

Distinguished Names Length: 113

Distinguished Names (113 bytes)

Distinguished Name Length: 111

Distinguished Name: (id-at-commonName=ca.example.iwall.com,id-at-organizationalUnitName=CA,id-at-organizationName=IWall,id-at-localityName=HaiDian,id-at-stateOrProvinceName=BeiJing,id-at-countryName=CN)

## Server Hello Done

Source:ssl\_cli.c

Func: ssl\_parse\_server\_hello\_done

Wireshark协议包：

Transport Layer Security

TLSv1.2 Record Layer: Handshake Protocol: Server Hello Done

Content Type: Handshake (22)

Version: TLS 1.2 (0x0303)

Length: 4

Handshake Protocol: Server Hello Done

Handshake Type: Server Hello Done (14)

Length: 0

## Certificate

Source:ssl\_cli.c

Func: mbedtls\_ssl\_write\_certificate

Wireshark协议包：

TLSv1.2 Record Layer: Handshake Protocol: Certificate

Content Type: Handshake (22)

Version: TLS 1.2 (0x0303)

Length: 475

Handshake Protocol: Certificate

Handshake Type: Certificate (11)

Length: 471

Certificates Length: 468

Certificates (468 bytes)

Certificate Length: 465

Certificate: 308201cd30820172021475bda3ed4a9e4fcba43686b12069… (id-at-commonName=Android,id-at-organizationalUnitName=CLIENT,id-at-organizationName=IWall,id-at-localityName=HaiDian,id-at-stateOrProvinceName=BeiJing,id-at-countryName=CN)

signedCertificate

algorithmIdentifier (ecdsa-with-SHA256)

Padding: 0

encrypted: 3046022100d2ff654bad8f8c917bce6e61c7c0767c661cbf…

## Client Key Exchange

Source:ssl\_cli.c

Func: ssl\_write\_client\_key\_exchange

Wireshark协议包：

TLSv1.2 Record Layer: Handshake Protocol: Client Key Exchange

Content Type: Handshake (22)

Version: TLS 1.2 (0x0303)

Length: 70

Handshake Protocol: Client Key Exchange

Handshake Type: Client Key Exchange (16)

Length: 66

EC Diffie-Hellman Client Params

Pubkey Length: 65

Pubkey: 047b18a25c26f22f7cea0307b89dcb483167a9d3bdb21929…

## Certificate Verify

Source:ssl\_cli.c

Func: ssl\_write\_certificate\_verify

Wireshark协议包：

TLSv1.2 Record Layer: Handshake Protocol: Certificate Verify

Content Type: Handshake (22)

Version: TLS 1.2 (0x0303)

Length: 79

Handshake Protocol: Certificate Verify

Handshake Type: Certificate Verify (15)

Length: 75

Signature Algorithm: ecdsa\_secp384r1\_sha384 (0x0503)

Signature Hash Algorithm Hash: SHA384 (5)

Signature Hash Algorithm Signature: ECDSA (3)

Signature length: 71

Signature: 3045022100c6a116675a6b42fef04d673fe305a6f961ff1e…

数据分析：

Signature: 签名，使用客户端私钥对协议内容签名。

## Change Cipher Spec

Source:ssl\_cli.c

Func: mbedtls\_ssl\_write\_change\_cipher\_spec

Wireshark协议包：

Transport Layer Security

TLSv1.2 Record Layer: Change Cipher Spec Protocol: Change Cipher Spec

Content Type: Change Cipher Spec (20)

Version: TLS 1.2 (0x0303)

Length: 1

Change Cipher Spec Message

数据分析：

Change Cipher Spec Message: 客户端发送 ChangeCipherSpec 消息，表示客户端期望变更密码套件。

TLS1.3 内已经移除。

## Encrypted Handshake Message（Finished）

Source:ssl\_cli.c

Func: mbedtls\_ssl\_write\_finished

Wireshark协议包：

Transport Layer Security

TLSv1.2 Record Layer: Handshake Protocol: Encrypted Handshake Message

Content Type: Handshake (22)

Version: TLS 1.2 (0x0303)

Length: 40

Handshake Protocol: Encrypted Handshake Message

数据分析：

客户端发送 Finished 消息，表示握手过程已经完成。

## Change Cipher Spec

Source:ssl\_cli.c

Func: mbedtls\_ssl\_parse\_change\_cipher\_spec

Wireshark协议包：

TLSv1.2 Record Layer: Change Cipher Spec Protocol: Change Cipher Spec

Content Type: Change Cipher Spec (20)

Version: TLS 1.2 (0x0303)

Length: 1

Change Cipher Spec Message

数据分析：

服务器发送 Change Cipher Spec 消息，表示服务器期望切换密码套件。

## Encrypted Handshake Message（Finished）

Source:ssl\_cli.c

Func: mbedtls\_ssl\_parse\_finished

Wireshark协议包：

TLSv1.2 Record Layer: Handshake Protocol: Encrypted Handshake Message

Content Type: Handshake (22)

Version: TLS 1.2 (0x0303)

Length: 40

Handshake Protocol: Encrypted Handshake Message

数据分析：

服务器发送 Finish 消息，表示握手过程已经完成。

## Wrapup

Source:ssl\_cli.c

Func: mbedtls\_ssl\_handshake\_wrapup

结束握手，释放资源

# 写数据mbedtls\_client\_write

握手成功之后继续发送（GET/POST）数据给服务端，来获取设备端想要的数据，示例代码如下：

int ssl\_txdat\_sender(MbedTLSSession \*tls\_session,int len,char \*data)

{

int ret;

int offset = 0,sd\_len = 0,wr\_len = 0;

int max\_len = 512;

unsigned int cur\_time,p\_time;

if( data && len)

{

if( tls\_session)

{

#if defined(MBEDTLS\_SSL\_MAX\_FRAGMENT\_LENGTH)

max\_len = mbedtls\_ssl\_get\_max\_frag\_len( &tls\_session->ssl );

max\_len = min(1024,max\_len);

#endif

///bk\_printf("[AS]frag len:%d,mlen:%d\r\n",max\_len,len);

sd\_len = len;

p\_time = rtos\_get\_time();

do

{

if((sd\_len - offset) > max\_len)

{

wr\_len = max\_len;

}

else

{

wr\_len = (sd\_len - offset);

}

cur\_time = rtos\_get\_time();

///bk\_printf("[ssl]cur\_time:%d ms\r\n",cur\_time);

if((cur\_time - p\_time) > (18 \* 1000))

{

bk\_printf("[%d]ssl send timeout\r\n",\_\_LINE\_\_);

goto retu;

}

while((ret = mbedtls\_client\_write(tls\_session, (unsigned char\*)(data + offset), wr\_len)) <= 0)

{

if(MBEDTLS\_ERR\_NET\_SEND\_FAILED == ret)

{

bk\_printf("mbedtls\_client\_write fialed\r\n");

goto retu;

}

if(ret != MBEDTLS\_ERR\_SSL\_WANT\_READ && ret != MBEDTLS\_ERR\_SSL\_WANT\_WRITE)

{

bk\_printf("mbedtls\_ssl\_write returned -0x%x\r\n", -ret);

goto \_\_exit;

}

rtos\_delay\_milliseconds(10);

}

offset += (ret > 0)?ret:0;

}while(offset < len);

retu:

return offset;

}

}

return -1;

\_\_exit:

return -1;

}

# 读数据mbedtls\_client\_read

**收服务器返回过来的数据存到缓存buf里面**

int ssl\_read\_data(MbedTLSSession \*session,unsigned char \*msg,unsigned int mlen,unsigned int timeout\_ms)

{

int len;

int ret;

fd\_set readfds;

struct timeval timeout;

do

{

if(mbedtls\_ssl\_get\_bytes\_avail(&session->ssl) == 0)

{

#if 1

FD\_ZERO( &readfds );

FD\_SET( session->server\_fd.fd, &readfds );

if(timeout\_ms < 0xFFFFFFFFU)

{

if(timeout\_ms != 0)

{

timeout.tv\_sec = timeout\_ms/1000;

timeout.tv\_usec = (timeout\_ms%1000) \* 1000;

}

else

{

timeout.tv\_sec = 0;

timeout.tv\_usec = 1000;

}

}

else

{

timeout.tv\_sec = 0;

timeout.tv\_usec = 0;

}

ret = select( session->server\_fd.fd+1, &readfds, NULL, NULL, &timeout);

if( ret <= -1 )

{ ///-1 select faild

bk\_printf("[TLS]select error:%d\r\n",ret);

return -1;

}

else if( 0 == ret )

{

///bk\_printf("[TLS]read select timeout(%d)\r\n",timeout\_ms);

return 0;

}

if (!FD\_ISSET( session->server\_fd.fd, &readfds ) ) /\*one client has data\*/

{

return 0;

}

#endif

}

if(mlen == 0)

{

return 0;

}

len = (session->buffer\_len > mlen)?mlen:session->buffer\_len;

ret = mbedtls\_client\_read(session, (unsigned char \*)session->buffer, len);

if(ret == MBEDTLS\_ERR\_SSL\_WANT\_READ || ret == MBEDTLS\_ERR\_SSL\_WANT\_WRITE)

{

return 0;

}

if(ret == MBEDTLS\_ERR\_SSL\_PEER\_CLOSE\_NOTIFY)

{

bk\_printf("[AS]MBEDTLS\_ERR\_SSL\_PEER\_CLOSE\_NOTIFY\r\n");

return -1;

}

if(ret == 0)

{

bk\_printf("[AS]connection closed\n");

return -1;

}

if(ret < 0)

{

bk\_printf("[AS]mbedtls\_ssl\_read returned -0x%x\r\n", -ret);

return -1;

}

}while(0);

len = ret;

memcpy(msg,session->buffer,len);

return len;

}

# 问题相关分析FAQ

相关错误码在net\_socket.h,x509.h,ssl.h,pk.h文件中，当出现握手失败的情况，请找到对应的错误码，有相应的解释。

## 问题一、华为kitfwk认证的时候mbedtls握手出现-76（-0x4c）错误（kitfwk版本为315的阻塞版本）

查看-0x4c对应的错误码MBEDTLS\_ERR\_NET\_RECV\_FAILED

原因：接收不到服务器的数据。资源暂时不可用

该错误从非阻塞套接字上不能立即完成的操作返回，例如当没有数据排队从套接字中读取时的recv。这是一个非致命错误，应该稍后重试该操作。WSAEWOULDBLOCK被报告为非阻塞SOCK\_STREAM套接字上调用connect的结果是正常的，因为连接必须经过一段时间才能建立。

解决办法：在出错的函数mbedtls\_ssl\_fetch\_input里面做一个等待接收处理。如下所示

if( ssl->f\_recv\_timeout != NULL )

{

unsigned count = 0;

do

{

ret = ssl->f\_recv\_timeout( ssl->p\_bio,

ssl->in\_hdr + ssl->in\_left, len, ssl->conf->read\_timeout );

count++;

if(count>15000)

break;

if(ret == -0x4C)

{

osDelay(10);//此处要注意的是只有当出现-0x4c的时候才延时再去接收，如果正常情况也进行延时可能产生数据接收异常的问题

}

}while(ret == -0x4C);

}

else

{

unsigned count = 0;

do

{

ret = ssl->f\_recv( ssl->p\_bio,

ssl->in\_hdr + ssl->in\_left, len );

count++;

if(count>15000)

break;

if(ret == -0x4C)

{

osDelay(10);

}

}while(ret == -0x4C);

}

## 问题二、华为kitfwk认证mbedtls握手出现-0x6800错误

查看-0x6800对应的错误码MBEDTLS\_ERR\_SSL\_TIMEOUT

原因：数据读取超时，当初在调试kitfwk版本为315的非阻塞版本的时候把timeval做了下更改，变成了int型，现在kitfwk版本为308，华为传进来的时间是500ms，换算成us的时候超过了int的范围，导致这个值会变的很小。设置的超时时间根本来不及接收数据就退出了，引起一直打印-0x6800错误。

#define \_\_DEFINED\_struct\_timeval

struct timeval{

int tv\_sec;

int tv\_usec;

};

解决方法：更改数据类型，或者直接去掉，默认就是long

#define \_\_DEFINED\_struct\_timeval

struct timeval{

long tv\_sec;

long tv\_usec;

};

## 问题三、聚芯项目mbedtls握手出现-0x7080错误

查看-0x7080对应的错误码MBEDTLS\_ERR\_SSL\_FEATURE\_UNAVAILABLE

原因：把debug打印开启来之后（第一个是debug的宏打开，另外一个是debug\_threshold变量设置成5），通过正常的日志和异常的日志对比，定位到的原因是recv收到的头信息数据里面的长度小于数据的总长，导致返回-0x7080错误，通过wireshark抓包发现，服务器在发证书数据的时候采取了分包的形式，咨询中移动服务器的人也确认数据量大的时候会分包。分包采用的是包头+数据的形式，包头里面带的是一包数据的长度 。而mbedtls不会采用这种私有协议，只会根据第一包数据的长度来接收数据，导致数据不完整出错。

解决办法一：采用拼包的形式，根据总长，来判断具体要拼包的次数，同时申请一个缓存，用来存剔除包含第二包之后的包头只存数据的空间。

解决办法二：确认为什么会分包，是由客户端和服务端进行协商了，通过mbedtls\_ssl\_conf\_max\_frag\_len函数进行协商，确定发送的最大数据长度，如果超过设置的长度，则会产生分包逻辑。去掉这里协商即可

## 问题四、华为kitfwk认证mbedtls握手出现-0x52错误

查看-0x52对应的错误码MBEDTLS\_ERR\_NET\_UNKNOWN\_HOST

原因：网络异常，解析不到host

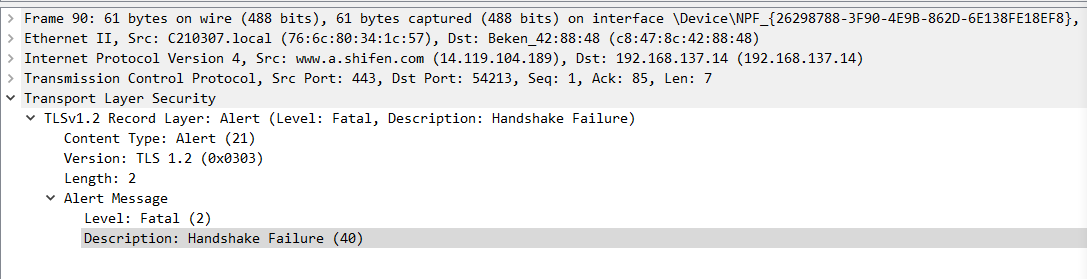
## 问题五、用百度的CA证书mbedtls握手出现-0x7780错误

查看-0x7780对应的错误码MBEDTLS\_ERR\_SSL\_FATAL\_ALERT\_MESSAGE

原因：服务端与客户端协商之后的密码套件与设备端支持的套件不匹配

解决办法：检测设备端支持的加密套件

通过抓包：如截图所示



## 问题六、mbedtls握手出现-0x7200错误

查看-0x7200对应的错误码MBEDTLS\_ERR\_SSL\_INVALID\_RECORD

原因：服务器下发的证书数据长度超过了MBEDTLS\_SSL\_IN\_CONTENT\_LEN设置的长度

解决办法：加大MBEDTLS\_SSL\_IN\_CONTENT\_LEN的长度

## 问题七、mbedtls握手出现-0x4290错误

未查到此对应的错误码

原因：打开了宏MBEDTLS\_SSL\_KEEP\_PEER\_CERTIFICATE，此宏是决定了在 SSL/TLS 握手结束后是否保留对端的证书，开启这个选项可能会消耗更多的内存，并且如果对端证书发生变化，你可能需要手动更新缓存,可禁用

## 问题八、mbedtls握手出现-0x7280错误

查看-0x7280对应的错误码是MBEDTLS\_ERR\_SSL\_CONN\_EOF

原因：打开了宏MBEDTLS\_KEY\_EXCHANGE\_ECDHE\_RSA\_ENABLED就有问题，应该是加密套件不支持这个

## 问题九、mbedtls握手出现-0x2700错误

查看-0x2700对应的错误码是MBEDTLS\_ERR\_X509\_CERT\_VERIFY\_FAILED

MBEDTLS\_KEY\_EXCHANGE\_DHE\_RSA\_ENABLEDMBEDTLS\_KEY\_EXCHANGE\_ECDH\_ECDSA\_ENABLEDMBEDTLS\_KEY\_EXCHANGE\_ECDH\_RSA\_ENABLED（加密套件可能不支持这些）MBEDTLS\_SSL\_MAX\_FRAGMENT\_LENGTH（此宏是客户端与服务端协商一包交互数据的最大长度，默认是1024,可去掉这个宏，交互数据服务器全部发过来，不去协商分包的大小）

## 问题十、魔方客户mbedtls握手在MBEDTLS\_SSL\_SERVER\_CERTIFICATE这一步解析服务器的证书出现-0x2700错误

日志如下：



原因：客户的证书是自己的CA证书，去解析可能会不认，导致报-0x2700错误MBEDTLS\_ERR\_X509\_CERT\_VERIFY\_FAILED

解决办法：mbedtls\_ssl\_conf\_authmode(&session->conf, MBEDTLS\_SSL\_VERIFY\_OPTIONAL);设置成证书验证失败也可继续通讯

## 问题十一、魔方客户mbedtls握手在MBEDTLS\_SSL\_CLIENT\_FINISHED这一步发送finished消息出现-0x50错误

查看0x50对应的错误码是MBEDTLS\_ERR\_NET\_CONN\_RESET，连接被服务端复位。

日志如下：



原因：客户采用的是双向认证，有客户端证书和PK，默认是没有去设置客户端证书和pk的，导致服务器不认client，报错。

解决办法：增加设置客户端证书和pk的逻辑



## 问题十二、魔方客户mbedtls握手出现-0x7780错误

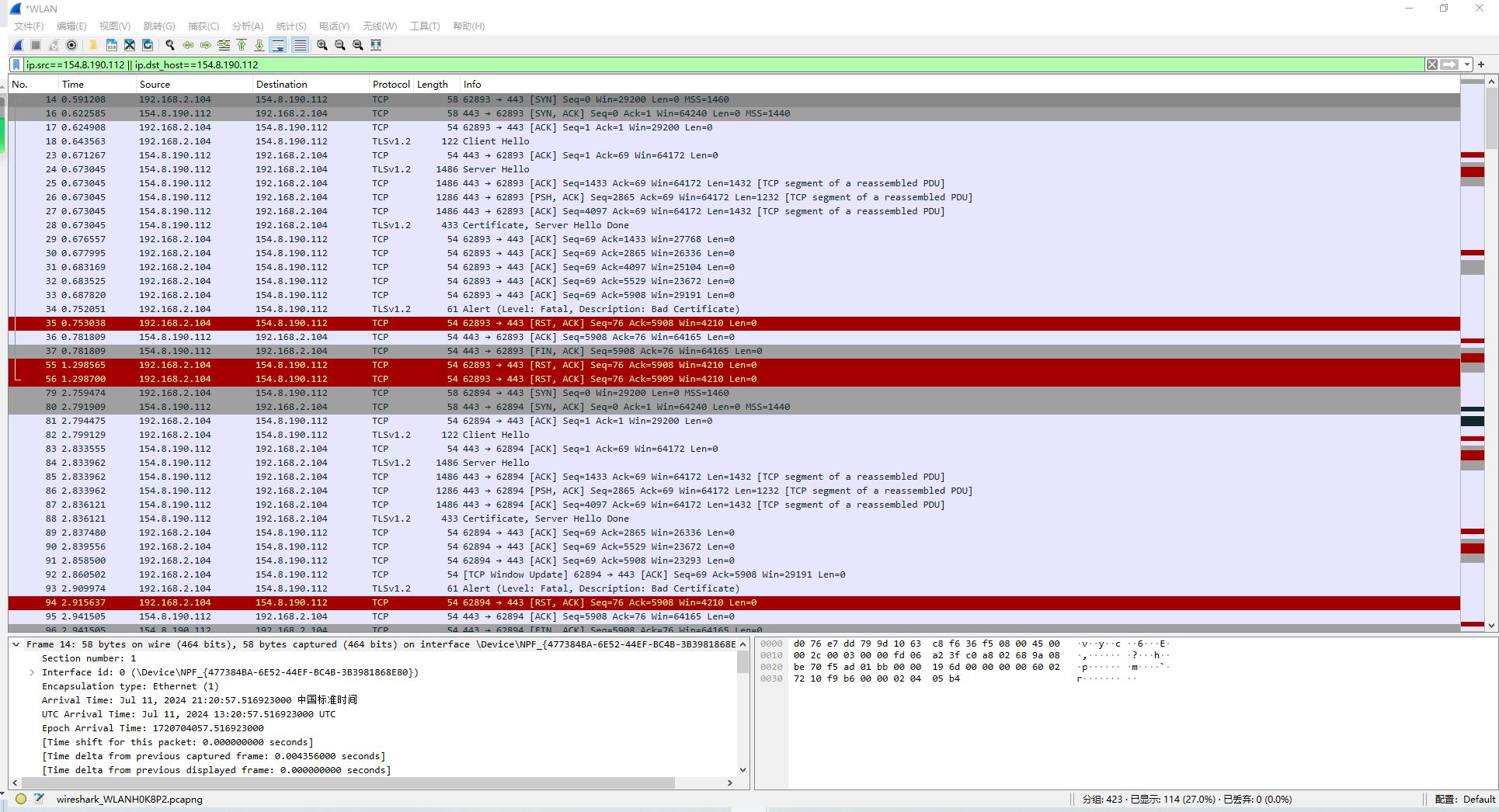
查看-0x7780对应的错误码MBEDTLS\_ERR\_SSL\_FATAL\_ALERT\_MESSAGE，收到服务器一条致命警告消息

原因：客户采用的是双向认证，有客户端证书和PK，默认是没有去设置客户端证书和pk的，导致服务器不认client，报错。

解决办法：增加设置客户端证书和pk的逻辑

## 问题十三、作业帮客户mbedtls握手出现-0x3b00错误

#define MBEDTLS\_ERR\_PK\_INVALID\_PUBKEY -0x3B00 /\*\*< The pubkey tag or value is invalid (only RSA and EC are supported). \*/



原因：使用rsa的密钥很大，解析的rsa的密钥是pubkey组件的缓冲区，而这个缓冲区是依赖宏MBEDTLS\_MPI\_MAX\_SIZE的大小

解决办法：

将

#define MBEDTLS\_MPI\_MAX\_SIZE 384

#define MBEDTLS\_ECP\_MAX\_BITS 384

修改

#define MBEDTLS\_MPI\_MAX\_SIZE 1024

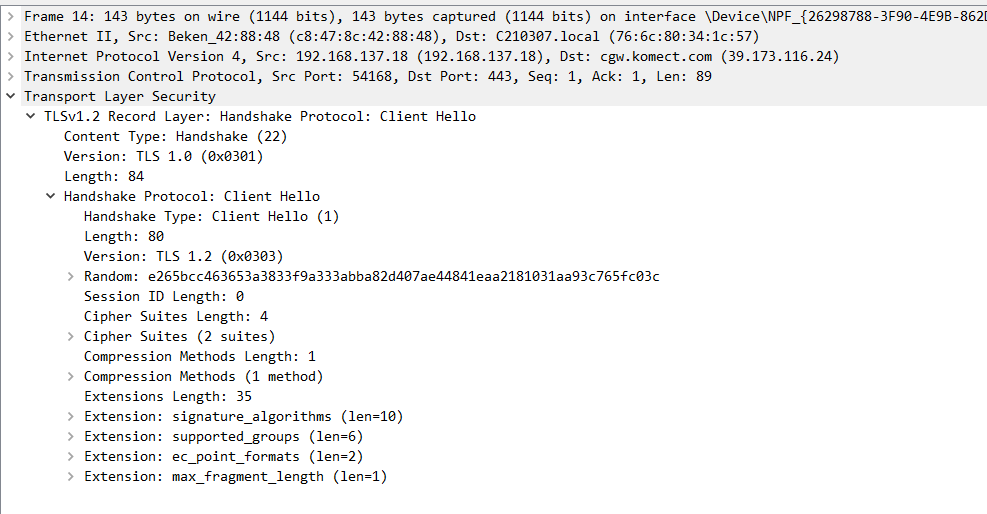
#define MBEDTLS\_ECP\_MAX\_BITS 521

# 示例聚芯wireshark抓包分析

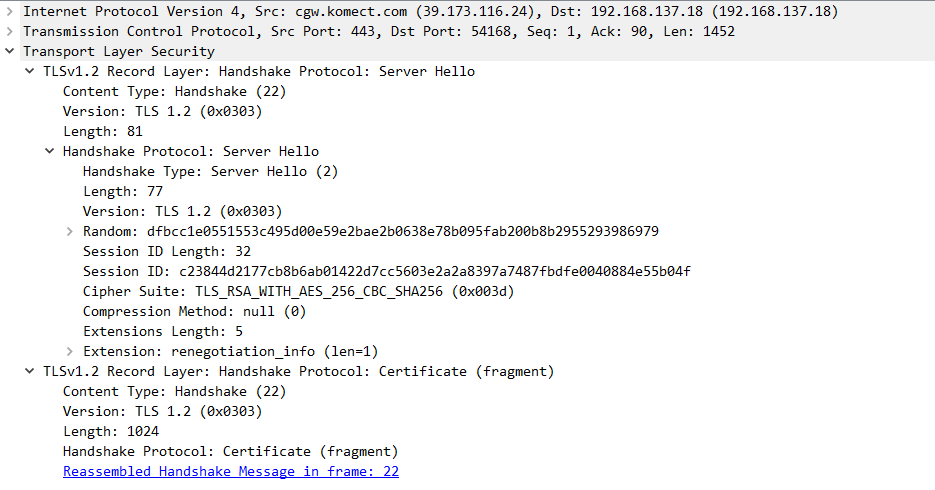
环境搭建：笔记本电脑连接一个无线，然后在开启一个移动热点，设备连接移动热点，打开wireshark工具，在捕获那里选择对应的过滤器。

## 1.未加拼包策略的抓包

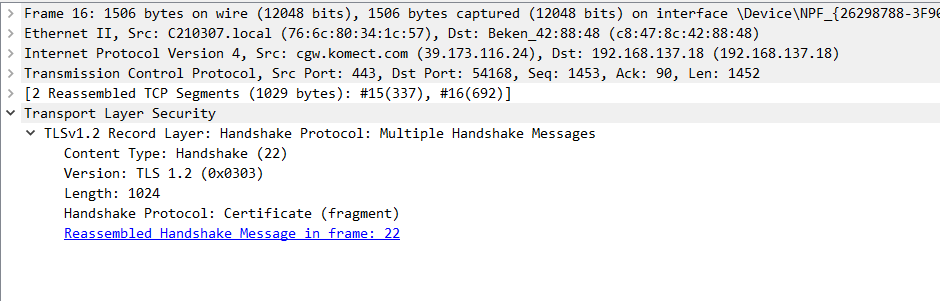
Client hello:如下图所示



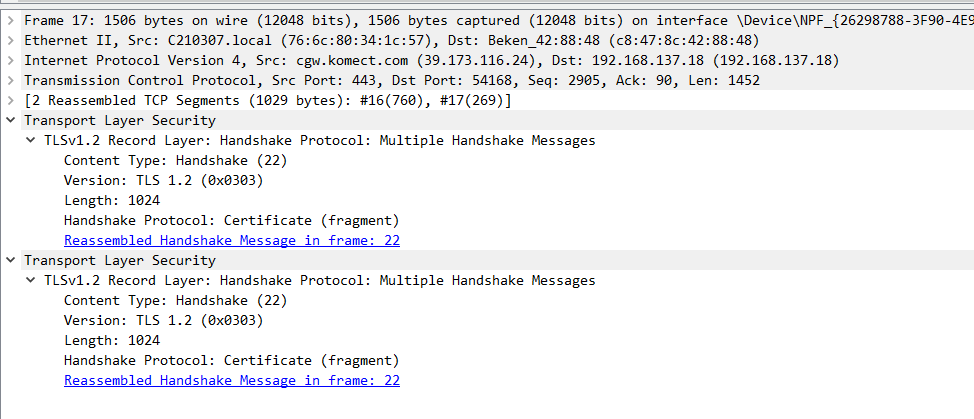
Server hello + first certificate fragment: 如下图所示此处可发现certificate证书数据进行了分包，一次只发了1024字节。这里设备接收数据之后，解析头信息里面的长度发现只有1024字节，然后跟总长比较，小于总长的5231字节，结果设备端认为异常就返回-0x7080错误，实际上服务器的数据还未完全发完，分包发送了。只是设备端mbedtls没有这个机制，导致后面的不去响应服务器，直接关闭链接了。



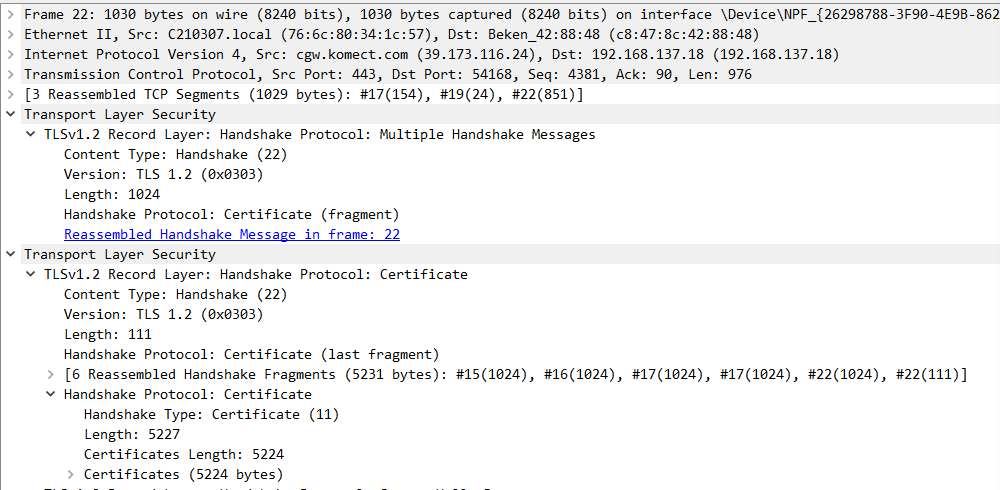
Second certificate fragment: 如下图所示



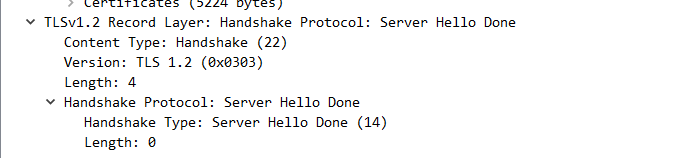
third and fourth certificate fragment: 如下图所示



Fifth and sixth certificate fragment：如下图所示



Server Hello Done:如下图所示

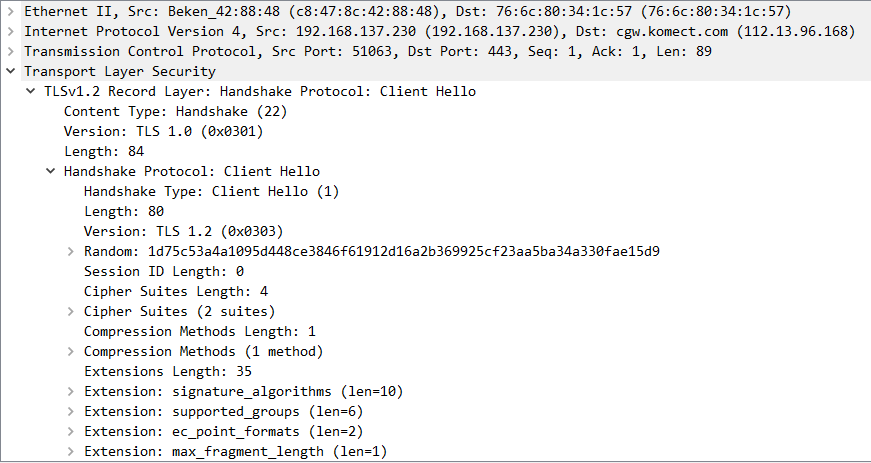


设备端因证书数据异常，直接返回错误，关闭连接了。

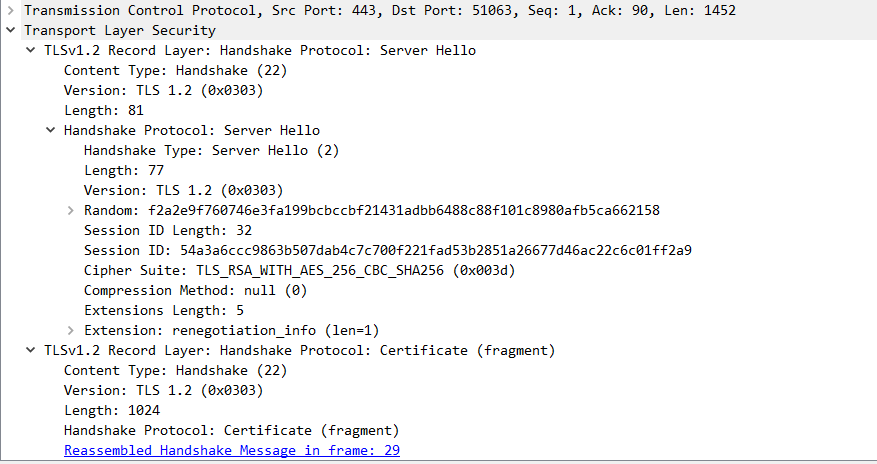


## 2.加拼包策略后的抓包

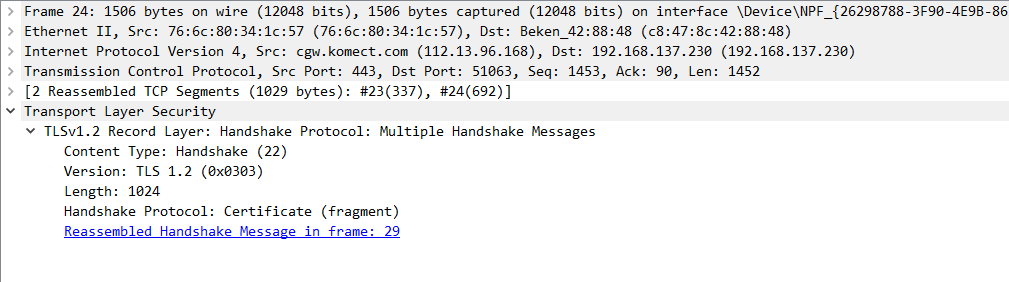
Client hello:如下图所示



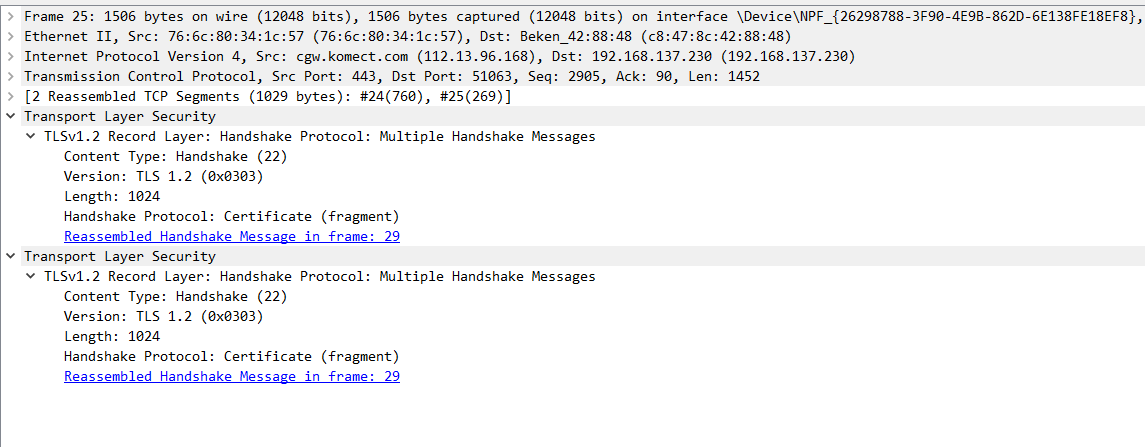
Server hello + first certificate fragment: 如下图所示(此处就发现服务器发下来的数据进行了分包，包的长度为1024字节，如果设备端不做拼包处理，那么就只会接收这一包，然后就去解析数据，数据长度和实际总长不匹配，返回-0x7080)



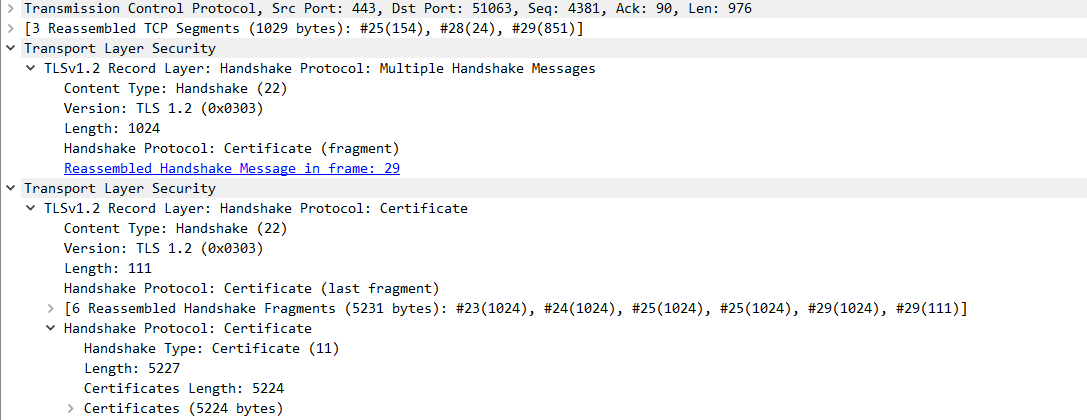
Second certificate fragment: 如下图所示



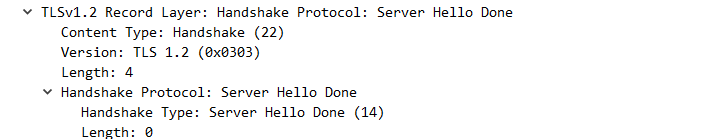
third and fourth certificate fragment: 如下图所示



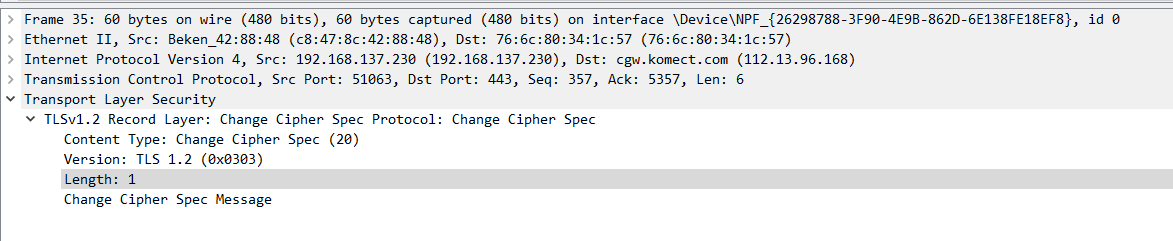
Fifth and sixth certificate fragment：如下图所示



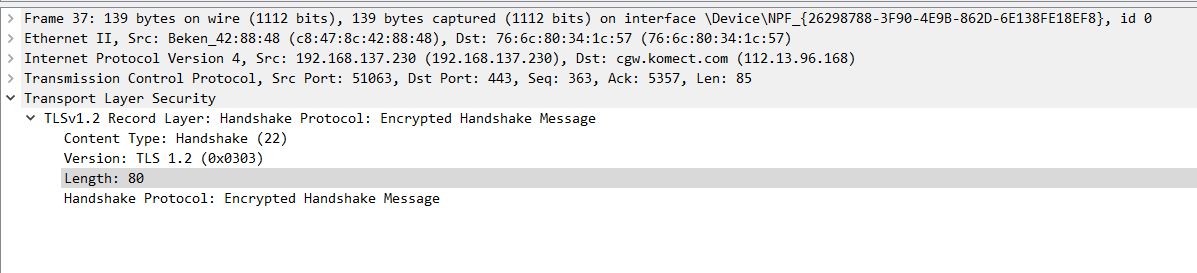
Server hello done：如下图所示



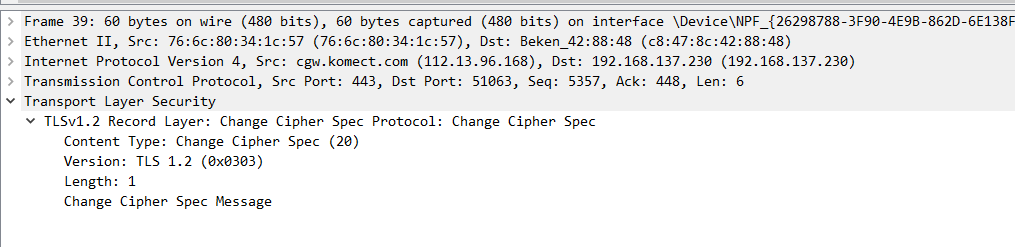
Change cipher spec(Client)：如下图所示



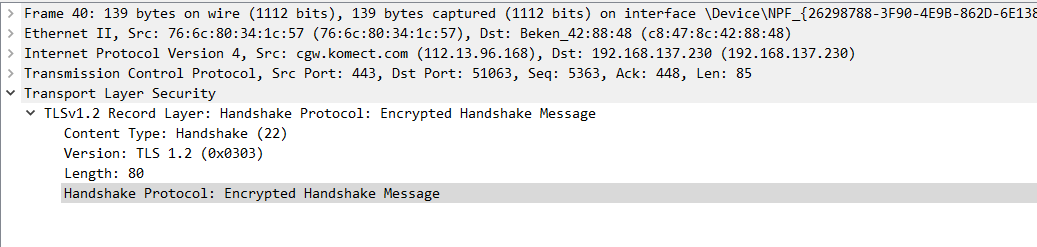
Encrypted Handshake Message(Client)：如下图所示



Change cipher spec(Server)：如下图所示



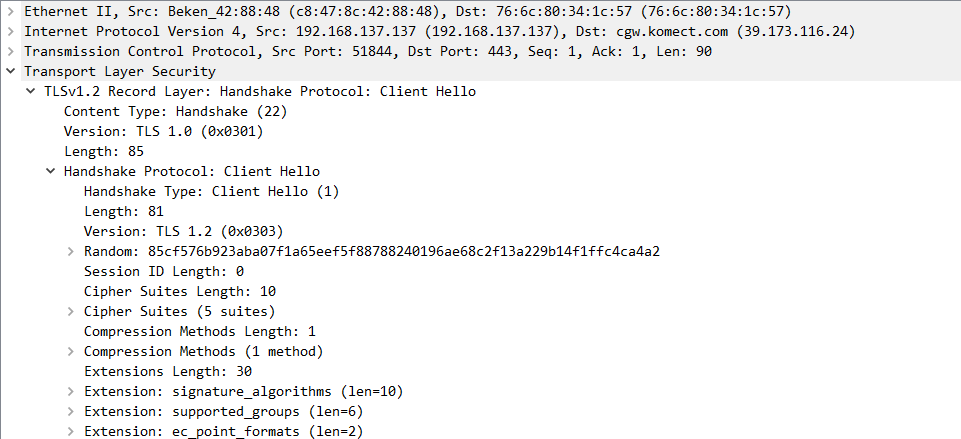
Encrypted Handshake Message(Server)：如下图所示



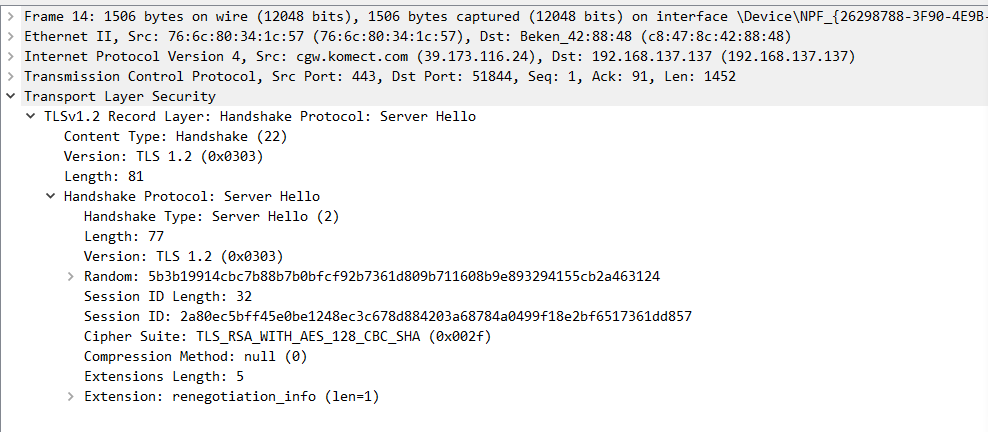


## 3.没有通过mbedtls\_ssl\_conf\_max\_frag\_len进行客户端和服务端协商数据包长度的抓包

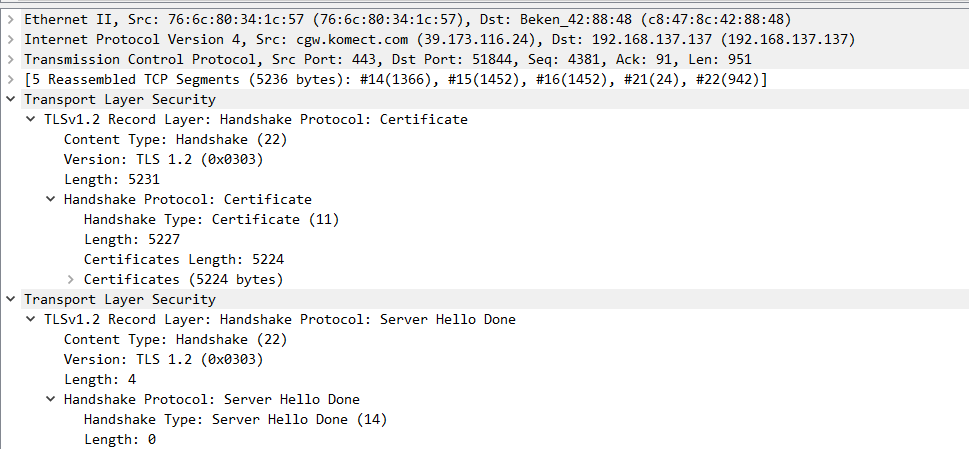
Client hello:如下图所示



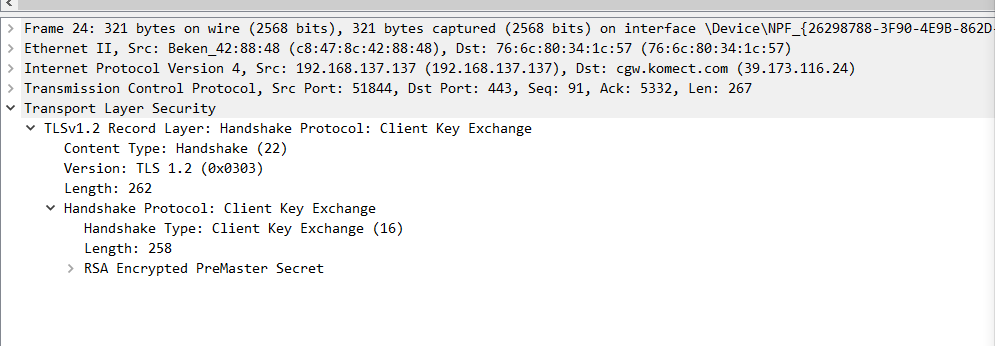
Server hello: 如下图所示



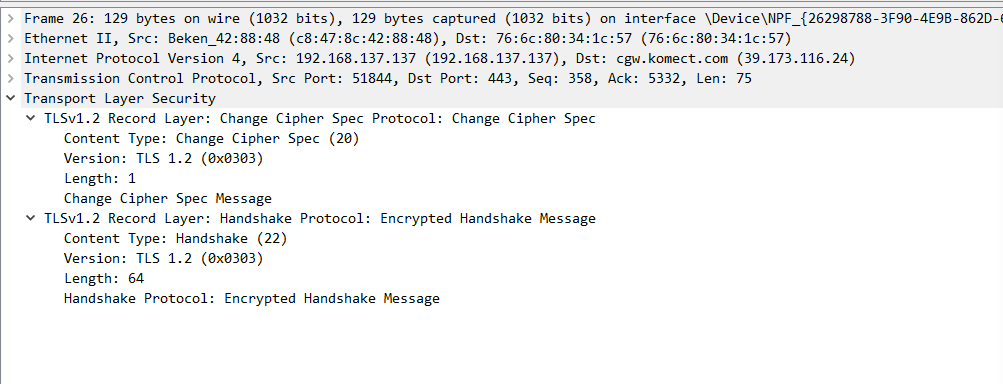
Certificate+Server Hello Done: 如下图所示



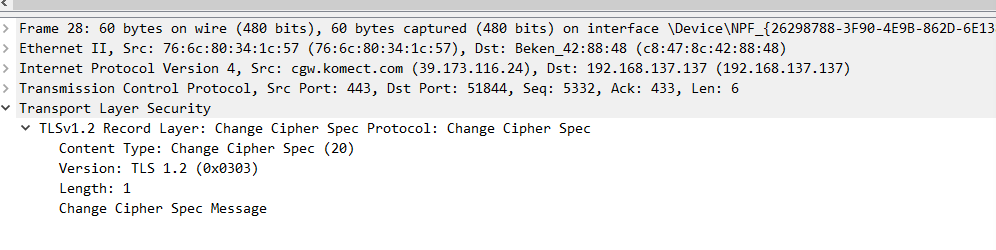
Client Key Exchange:如下图所示



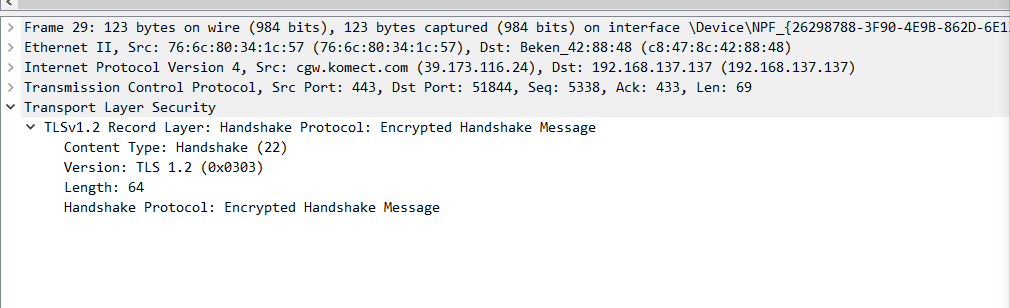
Change cipher spec(Client) + Encrypted Handshake Message(Client)：如下图所示



Change cipher spec(Server) ：如下图所示



Encrypted Handshake Message(Server)：如下图所示



# Mbedtls测试验证示例

## 示例一：串口命令

创建tls格式：tls create url port

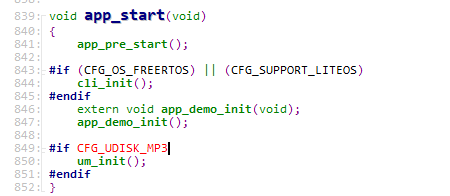
发送数据格式：tls sender data

聚芯测试命令如下：

tls create cgw.komect.com 443

注意：用串口命令测试前需要将app\_demo\_init()调用起来

如在：



## 示例二：curl

mbedtls下载链接：git clone <https://github.com/Mbed-TLS/mbedtls.git>

切换tag: mbedtls-2.4.2

cd mbedtls

git checkout mbedtls-2.4.2

make clean

make SHARED=1

make install

curl下载链接：<https://github.com/curl/curl.git>

1. 切换到tag: curl-7\_64\_1: git checkout curl-7\_64\_1
2. ./buildconf
3. ./configure --with-mbedtls --prefix=指定路径
   1. 如：./configure --with-mbedtls=/home/beken/wifi/mbedtls/mbedtls --prefix=/home/beken/wifi/curl
4. make && make install

这样会安装到指定路径，进行指定路径 测试

cd curl/bin

执行命令

./curl --location --request POST 'https://cgw.komect.com/device/inform/bootstrap' --header 'Content-Type: application/json' --header 'Accept-Charset: utf-8' --header 'User-Key: 42f05a0f0f2b4c62bb92ea7be67428f6' --data-raw '{"deviceMac":"C8478C428848","productToken":"8BZtnZw5yiPpWxSw","deviceType":"590384","timestamp":"1673186535843","protocolVersion":"V3.2","deviceExtInfo":{"authMode":0,"mac":"C8478C428849","sn":"0711059038400000001","reserve":"X1S","manuDate":"2023-01","OS":"FreeRTOS v9.0.0","chips":[{"type":"WIFI","factory":"BEKEN","model":"BK7231M"}]}}' --trace trace.info.mbedtls

输出结果如下：

{"andlinkToken":"5VCkKALqeXWd1ysO","deviceId":"CMCC-590384-C8478C428848","deviceToken":"79733e4d8ab5442d","dmToken":"+ezpdvJ9FYmpXraVFJwv1Ha5qhP8vDAmQ+63g5DcNDE=","gwToken":"14T/qrPZ1CiK8TkN","userId":1350655921}

过程交互则会存放在trace.info.mbedtls

## 示例三：百度带CA证书

代码参照：

CA证书宏打开：

#define CFG\_USE\_CA\_CERTIFICATE 1

#define CFG\_USE\_CA\_CERTIFICATE\_VERIFY 1

百度CA证书：

"-----BEGIN CERTIFICATE-----\r\n" \

"MIIDdTCCAl2gAwIBAgILBAAAAAABFUtaw5QwDQYJKoZIhvcNAQEFBQAwVzELMAkG\r\n" \

"A1UEBhMCQkUxGTAXBgNVBAoTEEdsb2JhbFNpZ24gbnYtc2ExEDAOBgNVBAsTB1Jv\r\n" \

"b3QgQ0ExGzAZBgNVBAMTEkdsb2JhbFNpZ24gUm9vdCBDQTAeFw05ODA5MDExMjAw\r\n" \

"MDBaFw0yODAxMjgxMjAwMDBaMFcxCzAJBgNVBAYTAkJFMRkwFwYDVQQKExBHbG9i\r\n" \

"YWxTaWduIG52LXNhMRAwDgYDVQQLEwdSb290IENBMRswGQYDVQQDExJHbG9iYWxT\r\n" \

"aWduIFJvb3QgQ0EwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQDaDuaZ\r\n" \

"jc6j40+Kfvvxi4Mla+pIH/EqsLmVEQS98GPR4mdmzxzdzxtIK+6NiY6arymAZavp\r\n" \

"xy0Sy6scTHAHoT0KMM0VjU/43dSMUBUc71DuxC73/OlS8pF94G3VNTCOXkNz8kHp\r\n" \

"1Wrjsok6Vjk4bwY8iGlbKk3Fp1S4bInMm/k8yuX9ifUSPJJ4ltbcdG6TRGHRjcdG\r\n" \

"snUOhugZitVtbNV4FpWi6cgKOOvyJBNPc1STE4U6G7weNLWLBYy5d4ux2x8gkasJ\r\n" \

"U26Qzns3dLlwR5EiUWMWea6xrkEmCMgZK9FGqkjWZCrXgzT/LCrBbBlDSgeF59N8\r\n" \

"9iFo7+ryUp9/k5DPAgMBAAGjQjBAMA4GA1UdDwEB/wQEAwIBBjAPBgNVHRMBAf8E\r\n" \

"BTADAQH/MB0GA1UdDgQWBBRge2YaRQ2XyolQL30EzTSo//z9SzANBgkqhkiG9w0B\r\n" \

"AQUFAAOCAQEA1nPnfE920I2/7LqivjTFKDK1fPxsnCwrvQmeU79rXqoRSLblCKOz\r\n" \

"yj1hTdNGCbM+w6DjY1Ub8rrvrTnhQ7k4o+YviiY776BQVvnGCv04zcQLcFGUl5gE\r\n" \

"38NflNUVyRRBnMRddWQVDf9VMOyGj/8N7yy5Y0b2qvzfvGn9LhJIZJrglfCm7ymP\r\n" \

"AbEVtQwdpf5pLGkkeB6zpxxxYu7KyJesF12KwvhHhm4qxFYxldBniYUr+WymXUad\r\n" \

"DKqC5JlR3XC321Y9YeRq4VzW9v493kHMB65jUr9TU/Qr6cf9tveCX4XSQRjbgbME\r\n" \

"HMUfpIBvFSDJ3gyICh3WZlXi/EjJKSZp4A==\r\n" \

"-----END CERTIFICATE-----"

命令测试方式如下：

// GET 请求

AT+HTTPCLIENT=2,0,https://www.baidu.com/,www.baidu.com,/,2

//POST 请求

AT+HTTPCLIENT=3,0,https://www.baidu.com/,www.baidu.com,/,2,hello

命令格式如下：

AT+HTTPCLIENT=<opt>,<content-type>,<url>,[<host>],[<path>],<transport\_type>[,<data>][,<http\_req\_header>][,<http\_req\_header>][...]

<opt>：客户端请求方式,常用如GET/POST

1:HEAD

2:GET

3:POST

4:PUT

5:DELETE:

<content-type>：客户端请求数据类型

0: application/x-www-form-urlencoded

1:application/json

2:multipart/form-data

3:text/xml

<url>：url链接，当<host> 和 <path> 参数为空，则用url解析的数据替换。

<host>：域名或者IP地址

<path>：HTTP/HTTPS路径

<transport\_type>：客户端传输类型，1：HTTP 2：HTTPS

1:HTTP\_TRANSPORT\_OVER\_TCP

2:HTTP\_TRANSPORT\_OVER\_SSL

<data>:当<opt>是POST/PUT请求时，此参数是发送给服务器的数据，如果不是这两种请求则不需要此参数

<http\_req\_header>：可发送请求头给服务器

回应如下：

<html>

<head>

<script>

location.replace(location.href.replace("https://","http://"));

</script>

</head>

<body>

<noscript><meta http-equiv="refresh" content="0;url=http://www.baidu.com/"></noscript>

</body>

</html>>

## 示例四：本地搭建服务端和客户端

代码参照如：



## 搭建服务端

1.配置mbedtls中的config.h,打开宏#define MBEDTLS\_SSL\_SRV\_C（不开此宏，客户端连接服务端的时候会报-0x7280错误，服务端会报-0x7080错误）

2.编译服务器端代码ssl\_server\_command，烧录到设备上

3.运行设备，联网，再启动服务端，可通过增加串口命令启动，如：ssl\_server 192.168.0.102 4433

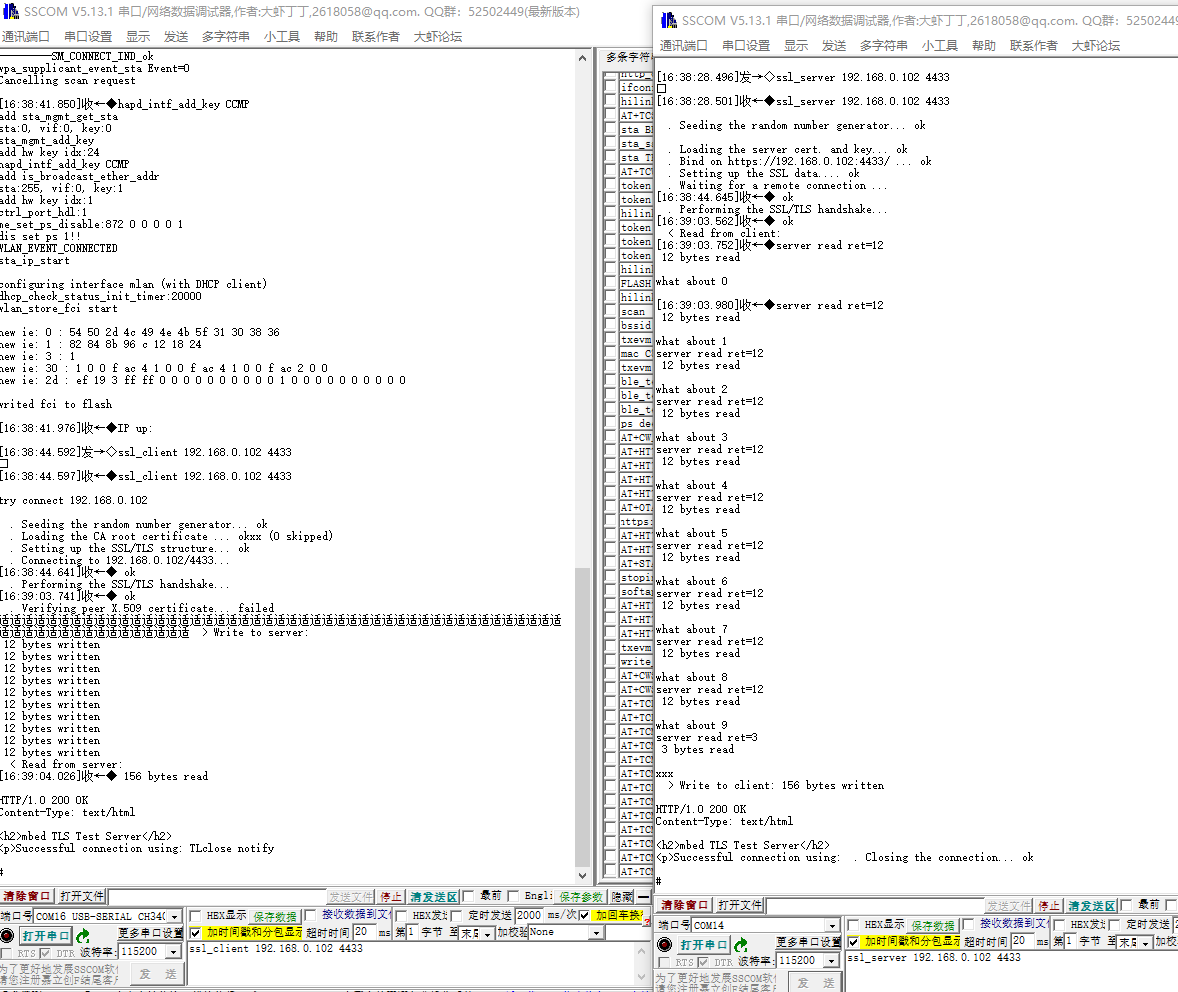
## 搭建客户端

1.配置mbedtls中的config.h,打开宏#define MBEDTLS\_SSL\_CLI\_C

2.编译客户端代码ssl\_client\_command，烧录到设备上

3.运行设备，联网，再启动客户端，可通过增加串口命令启动，如：ssl\_client 192.168.0.102 4433

**注意：两台设备连接同一网络，设备的mac地址也需不同，不然客户端和服务端分配的ip地址相同，mbedtls连接会失败。**



## 测试bin



## 示例五、ali\_mqtt+mbedtls的适配



# CA证书文件

参考文件：

# Wireshark包参考

