实验 1 TLS 配置与流量分析

1.1 【实验目的】

1） 理解 TLS 协议原理；

2）掌握 apache 服务器的 HTTPS 部署方法；

3）掌握 TLS 流量分析方法。

1.2 【实验内容】

1） 配置 TLS 协议分析环境；

1） 配置 apache 服务器的 https 协议

2） 对指定域名发起 HTTPS 请求，抓包分析 TLS 协议流程、提取其中的关键信息。

1.3 【实验原理】

TLS 协议分层

TLS 的密码学安全目标包括：保密性、完整性、身份认证

对于保密性来说，通常是通过对称加密组件实现。对称加密的前提是通信双方需要有共享密钥，因此需要一个密钥协商组件。TLS 的设计中，将上述功能分为：

（1）对称加密传输的记录协议，即：Record Protocol

（2）认证密钥协商的握手协议，即：Handshake Protocol

另外，还有三个辅助协议：

（1）Change Cipher Spec 协议

（2）Alert 协议

（3）Application Data 协议

因此，在设计上，TLS 协议是由 TLS 记录协议（Record Protocol）和 TLS 握手协议（Handshake Protocol）两层协议构成。记录协议位于下层，握手协议位于上层，记录协

议对上层数据包进行封装，然后利用 TCP 协议进行传输。握手协议又进一步划分为 4 个子协议。如图 1 所示：

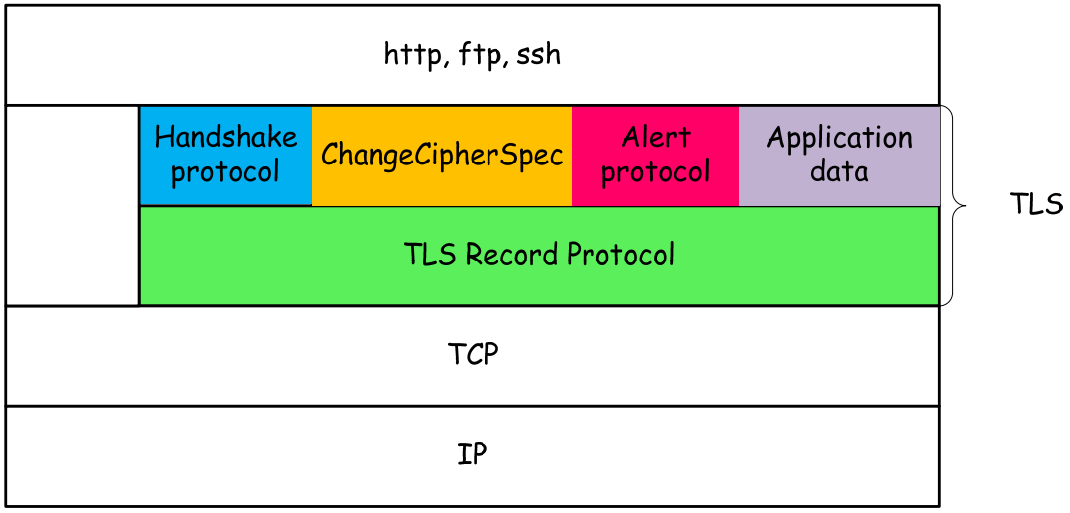


图 1 TLS 协议分层结构

记录协议负责实际的数据传输，需要确保传输数据的保密性和完整性，因此需要使用对称密码和消息认证码，而具体使用的算法及相关参数则是通过上层的握手协议来协商的。握手协议又分为 4 个子协议：handshake protocol, change cipher specific, alert protocol, 和 Application data 协议。这些协议又是如何完整协商的呢？下面我们来稍微具体地说明各协议的功能。

（1）TLS 记录协议

位于 TLS 协议的下层，负责安全传输数据，也就是确保数据传输的保密性和完整性。保密性和完整性通过对称密码和消息验证码来完成，因为候选的加密算法和消息验证码算法很多，因此需要通信双方协商来确定一致性，而这个协商过程就是由握手协议完成。

（2）TLS 握手协议

TLS 握手协议又细分为：握手协议（Handshake Protocol）、变更密码规格协议（ChangeCipherSpec Protocol）、警告协议（Alter Protocol）和应用数据协议（Aplication data Protocol）。

1) 握手协议

负责在客户端和服务器之间协商密码算法和共享密钥，同时完成基于证书的认证操作，为后面的应用数据传输做准备。握手协议的目的是使得通信两端就加密算法和相应的安全参数（比如共享密钥等）达成一致，在这种条件下，通信双方才能够正确理解（正确解密）

握手协议相当于下述对话：

客户端：“你好。我支持的协议版本号是 1.2，我能够支持的密码套件有 RSA/3DES、DSS/AES，请问我们使用哪个密码套件来通信？”

服务器：“你好哦。我们就用 RSA/3DES 密码套件；我的证书也给你看。”

客户端和服务器通过握手协议协商一致后，就会相互发出信号来切换密码，负责发出信号的就是下面的变更密码规格协议。

2) ChangeCipherSpec 协议

负责向通信对端传达变更密码规格的信号。

这个协议发送的消息相当于下面的对话。

客户端：“好，我们按照刚才的约定切换密码吧。”

如果在协议执行的中途发送错误，则会通过下面的警告协议传送相关信息给对端。

3) Alter 协议

负责在发生错误时将错误信息传给对端。

相当于如下对话：

服务器：“刚才的消息无法正确解密!”

变更密码规格后，如果没有出现错误，则会使用应用数据协议进行应用数据传输。

4) Application data 协议

负责将 TLS 承载的应用数据传递给通信对象。具体来说，就是把把 `http`、`ftp`、`smtp` 的数据流传人 `record` 层做处理并传输。

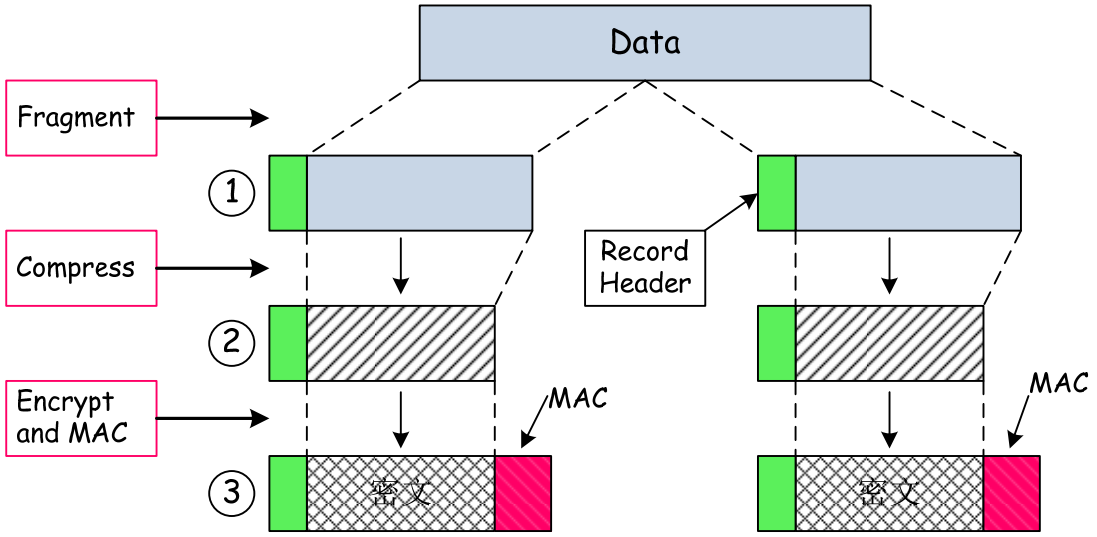
TLS Record Protocol

SSL 记录协议为 SSL 连接提供两个服务：

✧ Confidentiality：

✧ Message Integrity:

SSL 记录协议的封装过程如图所示：



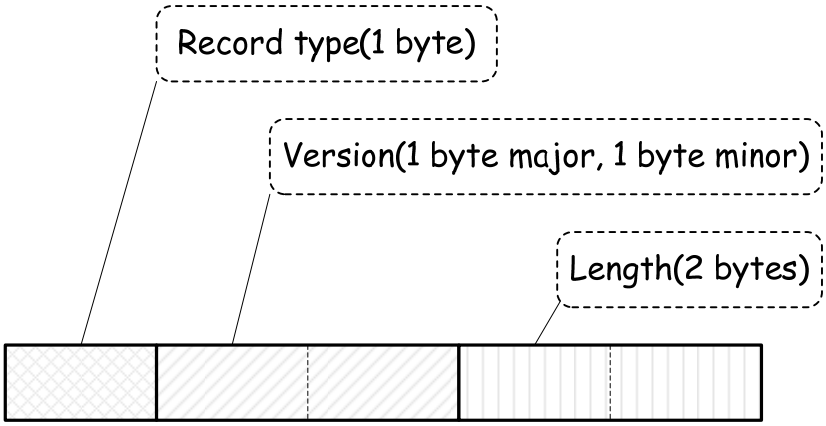
首先，消息被分割成多个较短的片段（fragment），然后分别对每个片段进行压缩。具体所采用的压缩算法通过前面的握手协议确定。实际使用中，很少采用压缩操作。

接下来，经过压缩的片段会被加上消息认证码，目的是为了保证消息完整性和数据源发认证。这里利用了 MAC 的功能。同时，为了防止重放攻击，在计算消息认证码时，还加上了片段的编号。这一步操作所需的算法和密钥同样是在握手协议阶段协商获得，包括：哈希函数算法、消息认证码所使用的共享密钥。

然后，经过压缩的片段拼接上消息认证码作为输入，通过对称加密算法进行加密。如果加密使用 CBC 模式，所需的初始化向量（IV）通过主密钥（master secret）生成，具体所采用的对称加密算法和共享密钥也是通过前面的握手协议确定。

最后，上面所得的密文加上记录协议头，则构成了记录协议报文，交由下层的 TCP 协议发送。

在 TCP/IP 协议栈中，每层协议都体现为封装上层协议，即：给上层协议报文添加一个头部和尾部（根据具体协议而言）。相应地，TLS 记录层协议也会给上层协议的报文添加一个头部，即 TLS Record header，如图所示。



TLS Record header

TLS Record header 由 5 个字节构成，分别表示：

记录类型（record type）——1 个字节

版本号（version）——2 个字节

长度（length）——2 个字节

记录类型用于说明后面所承载内容的协议类型，说明如下：

Record Type Values	Dec	Hex
CHANGE_CIPHER_SPEC	20	0X14
ALTER	21	0X15
HANDSHAKE	22	0X16
APPLICATION DATA	23	0X17

版本号用于说明所采用的 TLS 的版本号，包括主版本号和次版本号，说明如下：

Version values	Dec	Hex
SSL 3.0	3,0	0x0300
TLS 1.0	3,1	0x0301
TLS 1.1	3,2	0x0302
TLS 1.2	3,3	0x0303

具体使用的版本号是在 ServerHello 消息之后才确定的。因为 ClientHello 消息中说明了支持的 TLS 版本号，可能包括多个 TLS 版本号，而 ServerHello 消息确定了采用哪个版本。

长度用于说明记录中数据的长度（不包括记录头本身），tls 协议规定 length 必须小

于 2^{14}字节。如果 length 太长，接收方需要收到一个完整的 record 才能解密，可能需要较长时间，从而降低用户体验。

TLS HandShake Protocol

握手协议的工作原理

握手协议是 TLS 握手协议的一个子协议，用于在客户端和服务器之间协商产生用于记录协议中所要使用的密码算法和共享密钥，基于证书的认证操作也在这个协议中完成。

大致过程相当于如下对话：

客户端：“你好。我能够支持的密码套件有 RSA/3DES 或者 DSS/AES，我们使用哪个密码套件来通信呢？”

服务器端：“你好。我们就用 RSA/3DES 来进行通信吧，这是我的证书。”

服务器端和客户端通过握手协议协商一致后，就会通过 Change Cipher Spec Protocol 来发出信号，切换密码规格。

握手协议实现如下功能：

（1）客户端和服务器端协商 TLS 协议版本号和一个密码套件

（2）认证对端身份（可选，https 中一般只认证服务器的身份）

（3）使用密钥协商算法协商共享的 master secret

具体流程如下：

（1）交换 Hello 消息，协商出算法，交换 random 值，检查 session resumption.

（2）交换必要的密码学参数，来允许 client 和 server 协商出 premaster secret。

（3）交换证书和密码学参数，让 client 和 server 做认证，证明自己的身份。

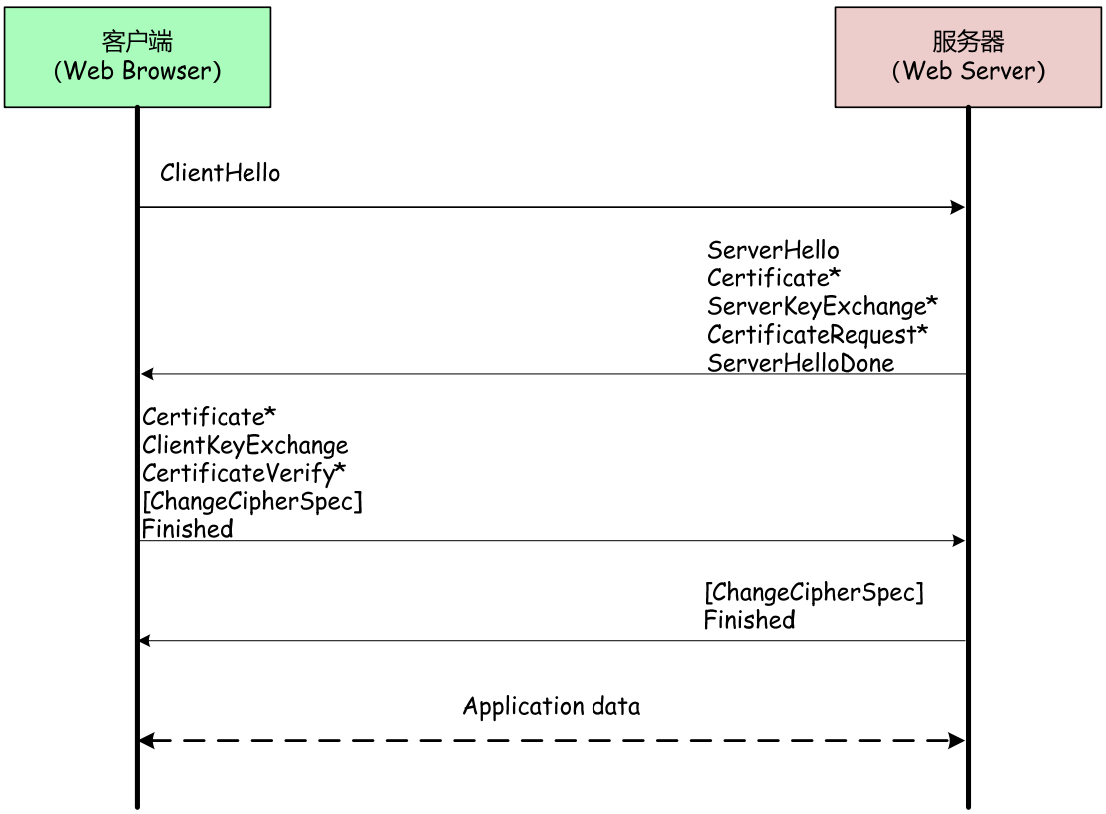
（4）从 premaster secret 和交换的 random 值 ，生成出 master secret。

（5）把 SecurityParameters 提供给 record 层。

（6）允许 client 和 server 确认对端得出了相同的 SecurityParameters，并且握手过程

的数据没有被攻击者篡改。

完整的握手过程如下：



*表示可选的消息或者视具体情况而定的消息，并不总是需要发送。

如上图所示，握手阶段除了 ChangeCipherSpec 外，所有交互的消息都是 handshake 消息，也就是 handshake 协议发送了不同类型的 handshake 消息。定义如下：

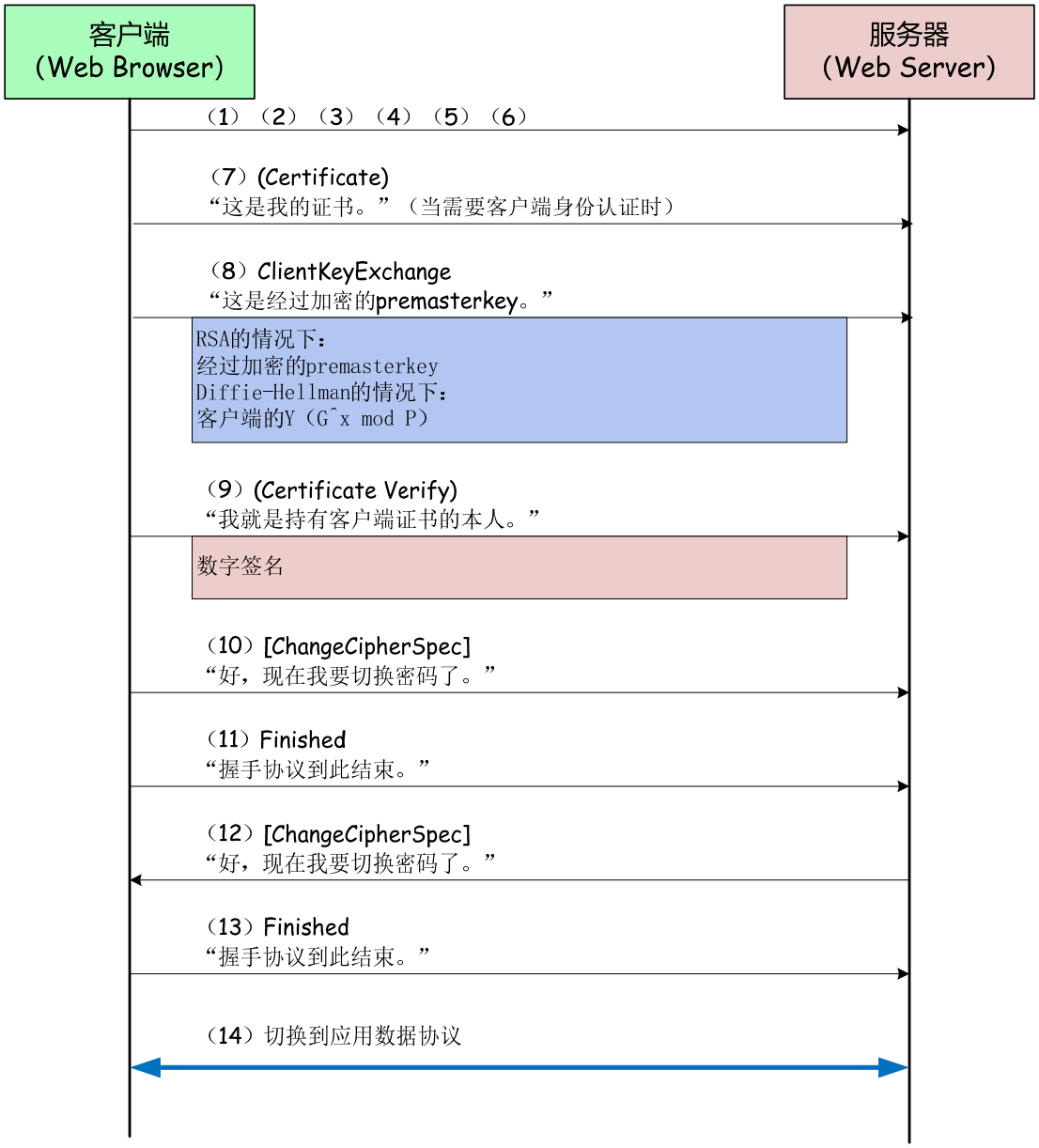
```
enum {
        hello_request(0), client_hello(1), server_hello(2),
        certificate(11), server_key_exchange (12),
        certificate_request(13), server_hello_done(14),
        certificate_verify(15), client_key_exchange(16),
        finished(20), (255)
    } HandshakeType;
```

handshake 消息表示如下：

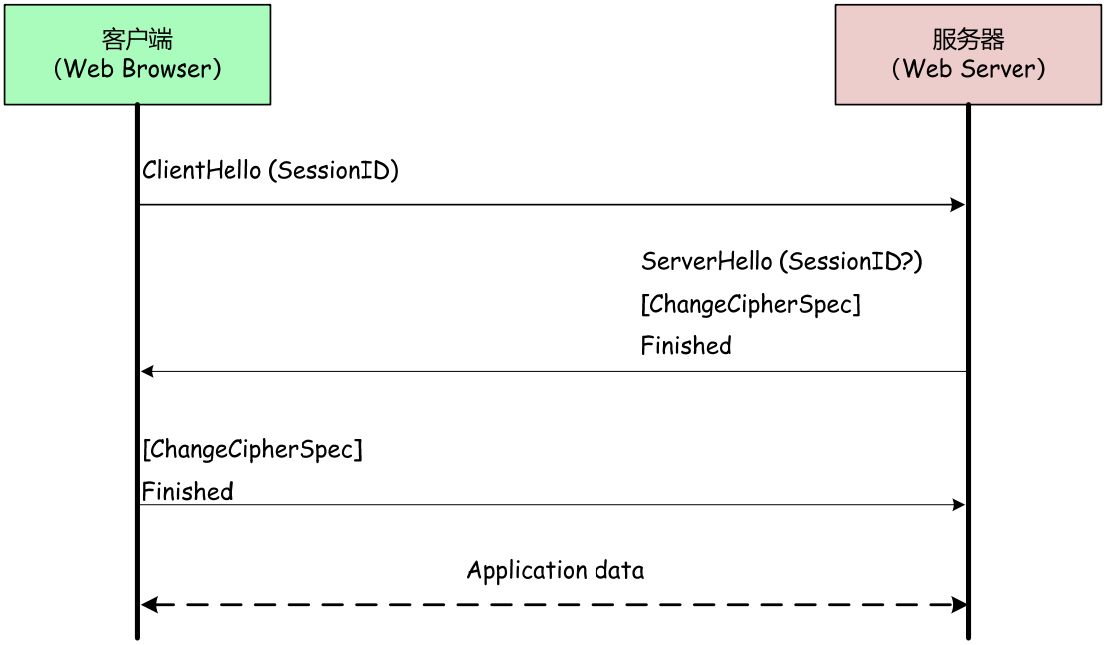
```
struct {
        HandshakeType msg_type;       /* handshake type */
        uint24 length;                /* bytes in message */
        select (HandshakeType) {
            case hello_request:       HelloRequest;
            case client_hello:        ClientHello;
            case server_hello:        ServerHello;
            case certificate:         Certificate;
            case server_key_exchange: ServerKeyExchange;
            case certificate_request: CertificateRequest;
            case server_hello_done:   ServerHelloDone;
            case certificate_verify:  CertificateVerify;
            case client_key_exchange: ClientKeyExchange;
            case finished:            Finished;
        } body;
    } Handshake;
```

握手协议的具体过程：


客户端
(Web Browser)
服务器
(Web Server)
（1）ClientHello
“你好。我支持的协议版本号是1.2，我能够支持的密码套件有RSA/3DES、DSS/AES，请问我们使用哪个密码套件来通信？”
可用的版本号；当前时间；客户端随机数；会话ID；支持的密码套件清单；支持的压缩方式清单
（2）ServerHello
“你好哦。我们就用RSA/3DES密码套件。”
使用的版本号；当前时间；服务器的随机数；会话ID；使用的密码套件；使用的压缩方式
（3）(Certificate)
“好，这是我的证书。”（非匿名通信时）
证书清单
（4）(ServerKeyExchange)
“我们用这些信息进行密钥交换吧。”（当证书信息不足时）
RSA的情况下：
公钥密码参数
N(modulus)
E(exponent)
散列值
Diffie-Hellman的情况下：
密钥交换的参数
P(prime modulus)
G(generator)
服务器的Y（G^x mod P）
散列值
（5）(CertificateRequest)
“对了，给我看一下你的证书吧。”（当需要认证客户端身份时）
证书类型清单
认证机构名称清单
（6）(ServerHelloDone)
“问候到此结束。”



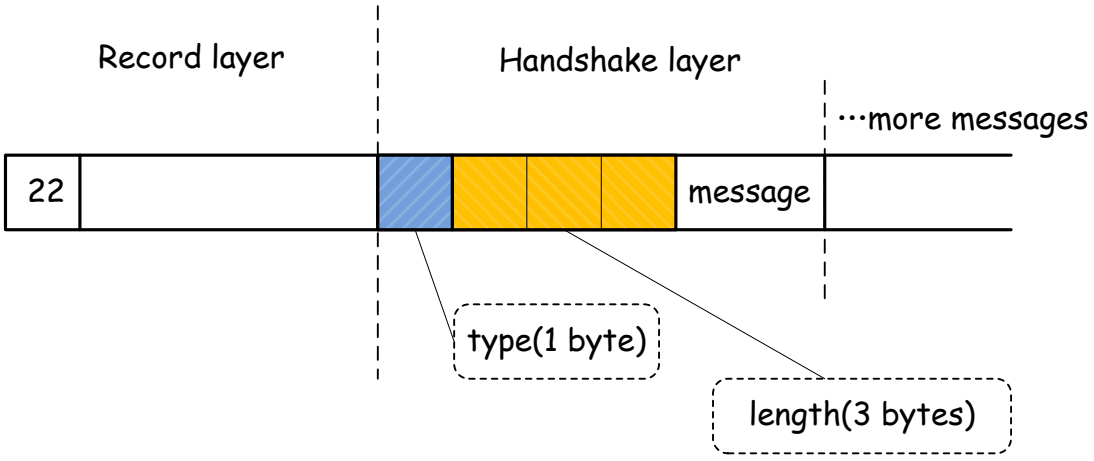
当客户端和服务器决定要恢复一个以前的会话或者复制一个现有的会话（而不是重新协商一个新的会话），消息流为如下：



客户端发送一个 ClientHello 消息，里面包含想要恢复的会话的 SessionID。服务器收到后，检查 Session cache 中是否有匹配的项。如果找到一个匹配的且服务器愿意以指定的会话状态重建连接，服务器将以相同的 SessionID 值发送 ServerHello 消息。然后，客户端和服务器均必须发送 ChangeCipherSpec 消息且直接继续发送 Finished 消息。一旦重建连接完成，客户端和服务器可以开始交换应用层数据。如果服务器没有在 Session cache 里面找到匹配的 SessionID，服务器就生成一个新的 Session ID，然后 TLS 客户端和服务器完成一个完整的 handshake。

握手协议的细节

握手协议包格式如图所示。



根据图示，Handshake 层的头部由 4 个字节构成，分为两个字段。第一个字段为 type，

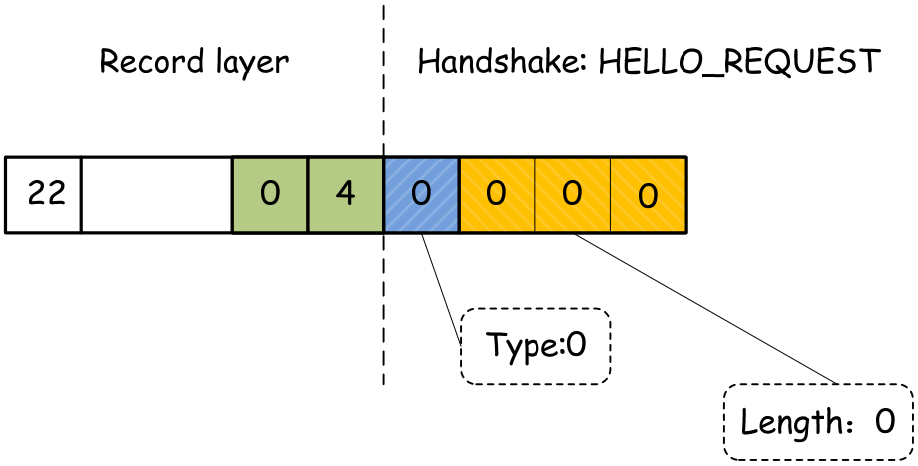
由一个字节构成，用于说明后面所承载消息的类型。第二个字段为 length，由 3 个字节构成，用于说明后面所承载消息的长度（为什么用 3 个字节表示？感觉太长了吧？）。随后，是消息内容。同时注意，handshake 层报文会封装在 Record 层里面，因此图示中，Record header 的类型字段值为 22，表示后面承载的内容是 handshake 协议消息。

handshake 层的类型用于说明后面消息的类型，共有 10 种（不包括扩展），如下表所示。

Handshake type values	Dec	Hex
HELLO_REQUEST	0	0X00
CLIENT_HELLO	1	0X01
SERVER_HELLO	2	0X02
CERTIFICATE	11	0X0B
SERVER_KEY_EXCHANGE	12	0X0C
CERTIFICATE_REQUEST	13	0X0D
SERVER_HELLO_DONE	14	0X0E
CERTIFICATE_VERIFY	15	0X0F
CLIENT_KEY_EXCHANGE	16	0X10
FINISHED	20	0X14

（1）HELLO_REQUEST（Server——>Client）：

服务器用 HELLO_REQUEST 消息来重启握手协商。如果一个连接已经建立了足够长时间，比如数小时，那么这个连接的安全性可能会降低，因此服务器使用该消息强制客户端重新协商会话密钥。该功能使用较少。HELLO_REQUEST 包的格式如下所示。



（2）ClientHello（客户端——>服务器）

该消息通常用于发起一个 TLS 握手协商，发送给服务器一个客户端所支持的密码套件列表，以供服务器选择一个最适合的密码套件（倾向于安全性最强的），一个压缩方法列表，一个扩展列表。也可包含 SessionId 字段，通过该字段，客户端可以重启一个前面的会话。具体来说包括：

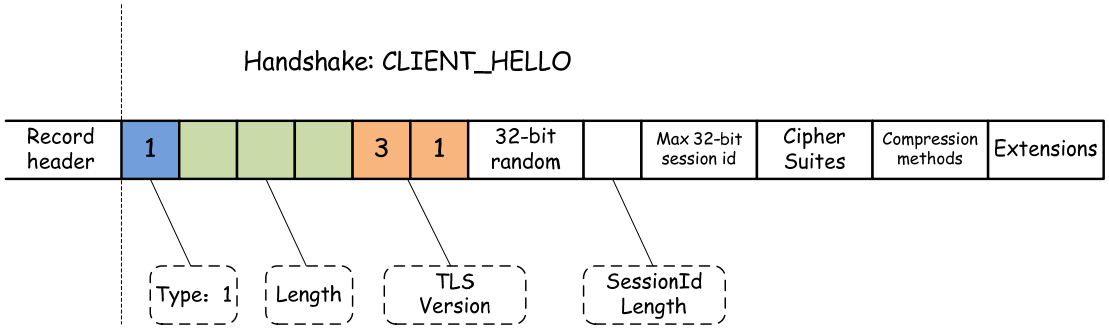
- 可用的 TLS 版本号
- 当前时间
- 客户端随机数
- 会话 ID
- 可用的密码套件清单
- 可用的压缩方式清单

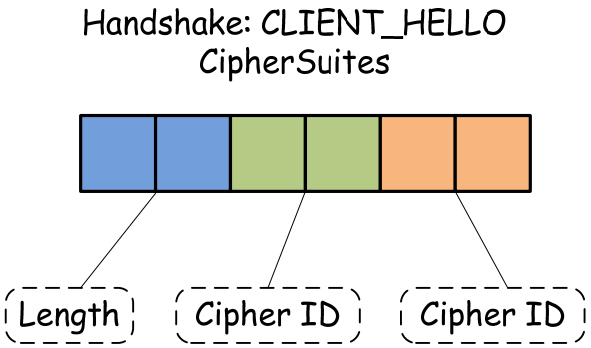
因为不同的客户端支持的方式不同，如果不进行协商一致的话，是无法正确通信的。客户端通过提供“可用版本号”，“可用密码套件清单”，“可用的压缩方式清单”来和服务器协商一种双方都支持的版本号、密码套件、压缩方式。

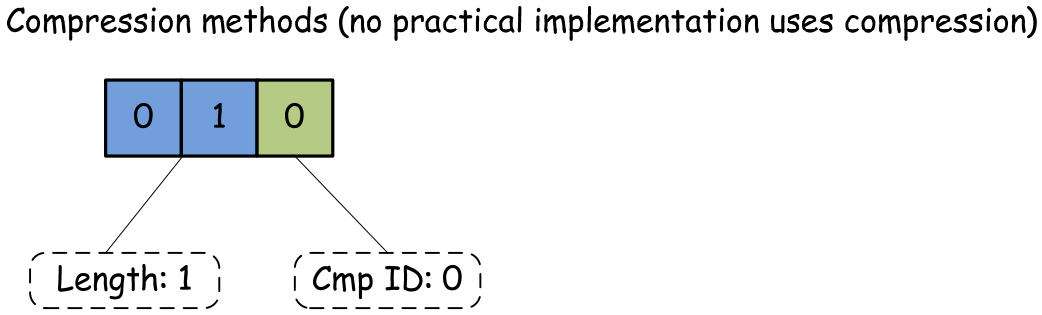
“当前时间”在基本的 TLS 中不会使用，但上层协议可能会用到这个信息。“客户端随机数”，在后面的步骤中需要用到。

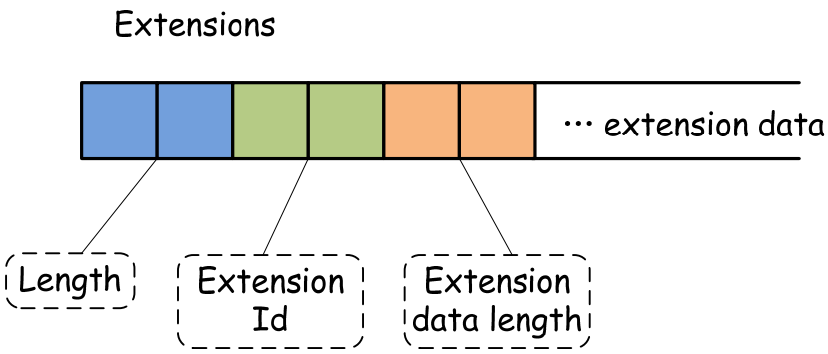
“会话 ID”用于客户端恢复以前建立的会话。当 client 决定恢复一个之前的 session，或复用一个已有的 session 时(可以不用协商一个新的 SecurityParameters)，消息流程如下：

客户端使用要被恢复的 session，发送一个 ClientHello，把 Session ID 包含在其中。server 在自己的 session cache 中，查找客户端发来的 Session ID，如果找到，sever 把找到的 session 状态恢复到当前连接，然后发送一个 ServerHello，在 ServerHello 中把 Session ID 带回去。然后，client 和 server 都必须 ChangeCipherSpec 消息，并紧跟着发送 Finished 消息。这几步完成后，client 和 server 开始交换应用层数据（如下图所示）。如果 server 在 session cache 中没有找到 Session ID，那 server 就生成一个新的 session ID 在 ServerHello 里给客户端，并且 client 和 server 进行完整的握手。









（3）ServerHello（服务器——>客户端）

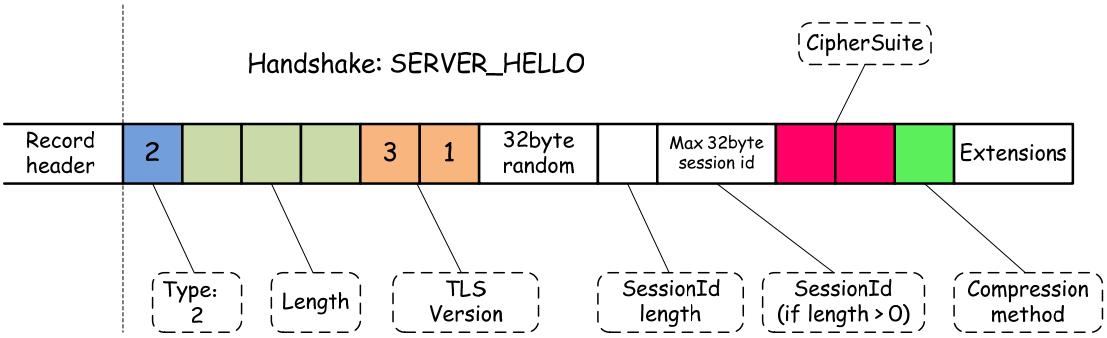
当服务器收到来自客户端的 ClientHello 消息后，如果它能够找到一套可以接受的算法（即可以就加密算法等取得协商一致），服务器将发送 ServerHello 消息来响应客户端的 ClientHello 消息。如果不能找到一套匹配的算法，则服务器将响应应 handshake failure alert。

ServerHello 消息和 ClientHello 消息相似，例外的地方是只包括一个密码套件和一个压缩方法。如果包含了 SessionId（SessionId Length＞0），表明服务器告诉客户端将来可重用该会话。具体的消息包括：

- 使用的版本号
- 当前时间
- 服务器随机数
- 会话 ID
- 使用的密码套件
- 使用的压缩方式

服务器根据客户端在 ClientHello 消息中发送的信息来确定后面通信中使用的“版本号”、“密码套件”、“压缩方式”。

“服务器随机数”在后续步骤中会使用。



（4）Certificate（服务器——>客户端）

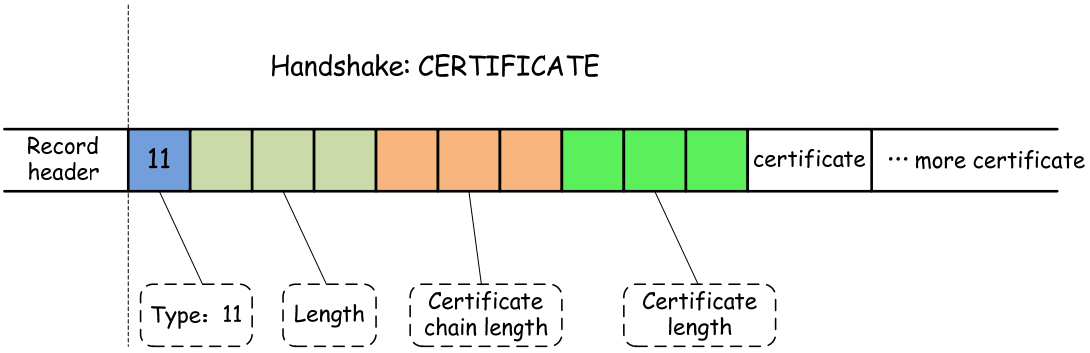
服务器发送给客户端的 Certificate 消息，也就是告诉客户端，“这是我的证书”。Certificate 消息里面包含如下信息。

- 证书清单

证书清单是一组 X.509v3 证书序列（证书链），首先发送的是服务器的证书，然后是按顺序发送对服务器证书签名的 CA 的证书。

客户端通过对服务器发送的证书进行验证来确定所通信的对象是否合法。

消息格式如下：

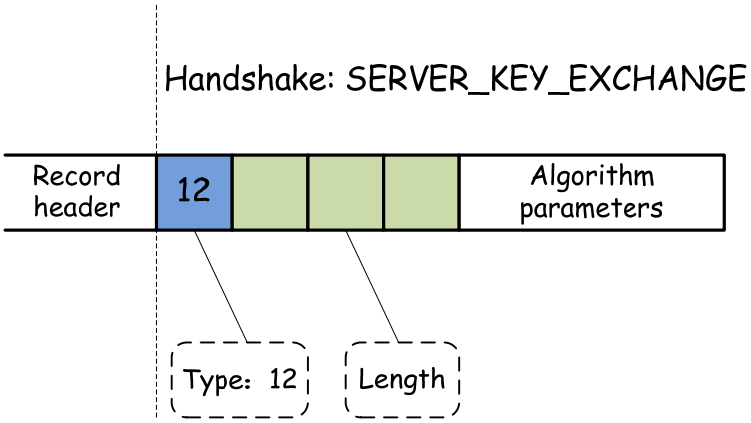


（5）ServerKeyExchange（服务器——>客户端）

When：服务器发送 server Certificate 消息后，立即发送 ServerKeyExchange 消息（如果是匿名协商，则在 ServerHello 后立即发送该消息）。同时，仅当 server Certificate 消息包含的信息不足以让客户端交换一个 premaster secret 时，才发送 ServerKeyExchange 消息。比如：DHE_DSS、DHE_RSA、DH_anon。而对于密钥交换算法 RSA、DH_DSS、DH_RSA，如果发送 ServerKeyExchange 消息则是非法。

服务器发送 ServerKeyExchange 消息，里面包括客户端需要从服务器获得的密钥交换算法参数，相当于告诉客户端“我们用这些信息来进行密钥交换吧。”该消息的可选的，不是所有的密钥交换都需要服务器显式地发送该消息。对于某些密码套件，Certificate 里面的信息并不足以完成密钥交换，因此服务器会根据所选择的密码套件来决定是否需要发送 ServerKeyExchange 消息。也就是，如果 Certificate 的信息就足够完成密码套件里面的密钥交换，则无需发送 ServerKeyExchange 消息。

ServerKeyExchange 包格式如下：



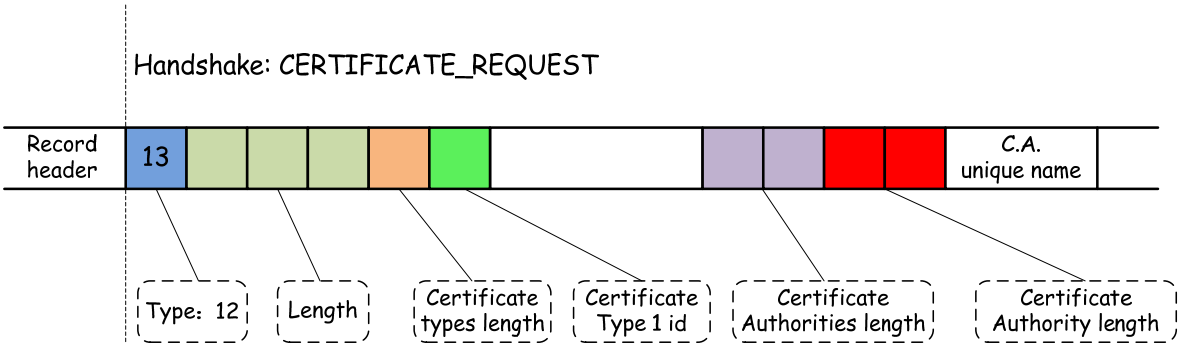
（6）CertificateRequest（服务器——>客户端）

服务器发送 CertificateRequest 消息，用于向客户端请求证书，目的是为了对客户端进行认证。CertificateRequest 消息包含如下信息：

- 服务器能够接受的证书类型清单
- 服务器信任的认证机构名称清单

如果无需对客户端进行认证，则不会发送 CertificateRequest 消息。在 web 服务器中，该消息不常用，也就是不需要对客户端进行身份认证。对客户端的身份认证通常是通过口令登录形式完成的。

消息格式如下：



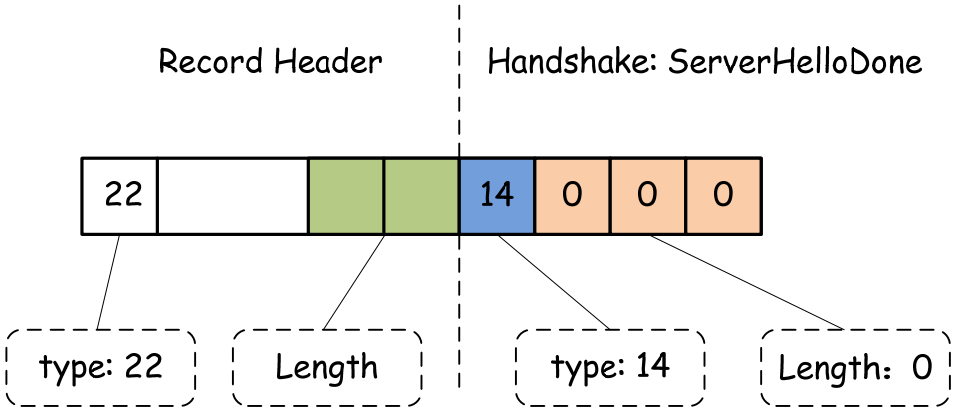
（7）ServerHelloDone（服务器——>客户端）

When：服务器发送 ServerHelloDone 消息来表示 ServerHello 及相关消息的结束，这些消息用于完成密钥交换，发送该消息后，服务器将等待客户端响应。而客户端收到该消息后，可以继续他的密钥交换阶段。

服务器发送 ServerHelloDone 消息，向客户端表示：从 ServerHello 消息开始的一系

列消息的结束，也就是“Server 端的握手协商到此结束了”。消息中不包括任何额外信息。

消息格式如下：



（8）Certificate（客户端——>服务器）

客户端发送 Certificate 消息，告诉服务器“这是我的证书”。如果服务器发送过 CertificateRequest 消息，则客户端会回复 Certficate 消息，其中包含客户端的证书。从而，服务器能够通过客户端的证书来进行验证。反之，如果服务器没有发送 CertificateRequest 消息，则客户端不会发送 Certificate 消息。

（9）ClientKeyExchange（客户端——>服务器）

When：如果客户端发送了 client Certificate 消息，ClientKeyExchange 消息应该在该消息后立即发送。否则，在客户端收到服务器发送的 ServerHelloDone 后立即发送该消息。

客户端发送 ClientKeyExchange 消息，向服务器提供用于生成对称加密密钥所需的数据，相当于告诉服务器：“这是经过加密的 Premaster secret。”

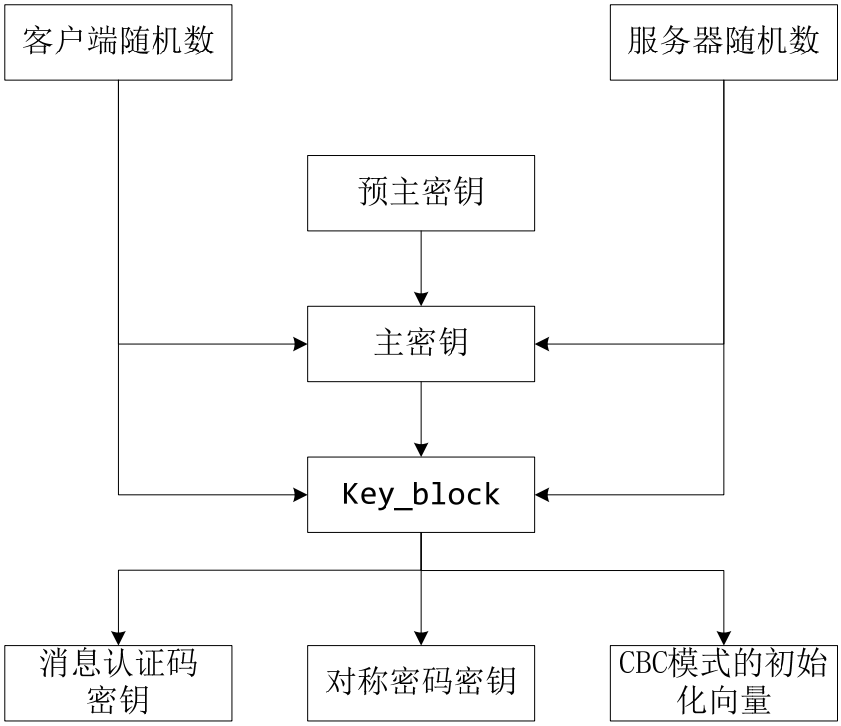
当密码套件包含 RSA 时，会随 ClientKeyExchange 消息一起发送经过 RSA-encrypted 的预备主密码。

当密码套件包含 Diffie-Hellman 密钥交换时，会随 ClientKeyExchange 消息一起发送 Diffie-Hellman 的公开值。

最终双方取得一致的 premaster secret。

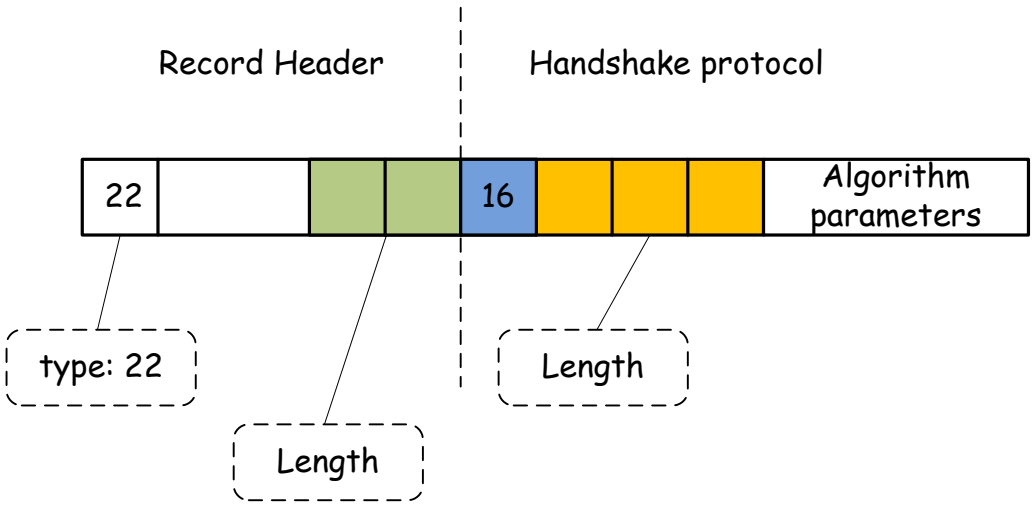
根据 pre-master secret，服务器和客户端会计算出相同的 master secret，然后根据 master secret 生成如下密钥信息。

- 对称加密的密钥
- 消息认证码的密钥
- 对称密码的 CBC 模式中使用的初始化向量（IV）



密钥导出过程

消息格式与 ServerKeyExchange 相似，如下：



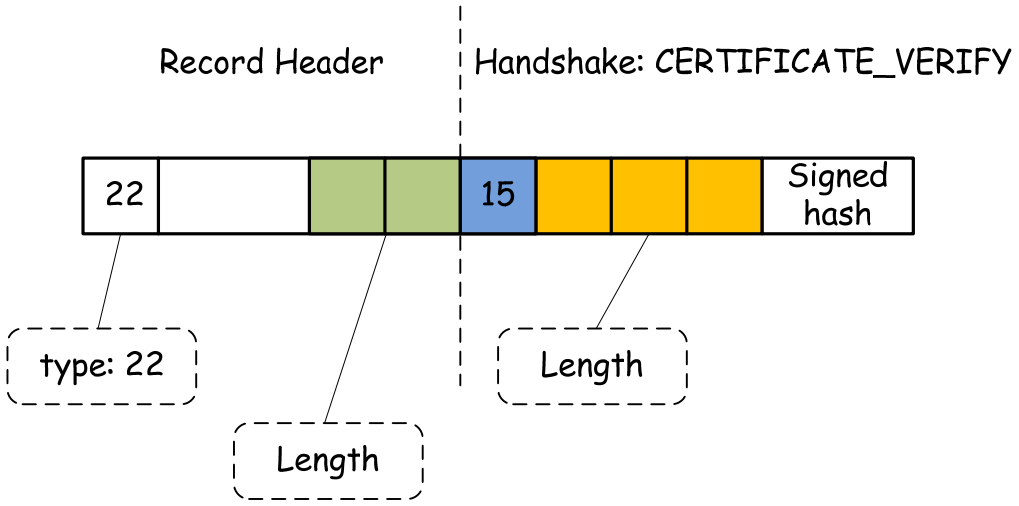
（10）CertificateVerify（客户端——>服务器）

客户端发送 CertificateVerify 消息，向服务器证明拥有对应公钥证书的私钥，相当于告诉服务器：“我是客户端证书持有者本人。”

只有服务器向客户端发送 CertificateRequest 消息的情况下，客户端才会向服务器发送 CertificateVerify 消息，以向服务器证明自己的确持有客户端证书的私钥。

为实现该目的，客户端会计算“master secret”和“握手协议中传送的消息”的散列值，并加上自己的数字签名后，发送给服务器。

消息格式如下：



（11）ChangeCipherSpec（客户端——>服务器）

客户端发送 ChangeCipherSpec 消息，告诉服务器：“我要切换密码套件了。”

ChangeCipherSpec 消息不是 Handshake 协议的消息，而是 ChangeCipherSpec 协议的消息。在发送 ChangeCipherSpec 消息之前，通信双方已经交换了关于协商密码套件的信息，因此收到此消息后，通信双方同时切换密码。

在此消息后，通信双方的 TLS 记录协议开始使用协商好的密码套件通信。

（12）Finished（客户端——>服务器）

When: 在 ChangeCipherSpec 消息后（表明密钥交换和认证过程已经成功完成），需要发送一个 Finished 消息。

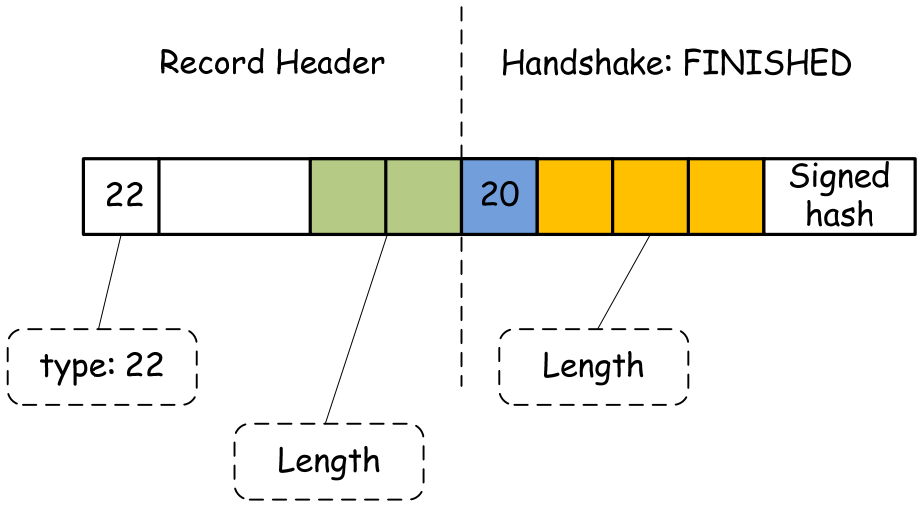
Finished 消息是第一个收到前面协商出的算法、keys、和 secret 保护的消息。接收方必须验证收到的内容是正确的。一旦通信一方发送了他自己的 Finished 消息且收到并验证了来自通信对端的 Finished 消息，他就可以开始通过建立的连接收发应用数据了。

客户端发送 Finished 消息，表明 TLS 握手协商完成，密码套件已经激活（在发送 Finished 消息之前必须发送 ChangeCipherSpec 来激活已经协商好的密码套件），相当于告诉服务器“握手结束。”

由于已经完成了密码切换，因此 Finished 消息是使用切换后的密码套件来发送的，也就是 Finished 消息不是以明文方式发送的，而是通过下层的记录协议进行加密发送。

Finished 消息的内容是哈希值，其输入是之前所有 handshake 消息的组合，后面跟随的用于识别服务器/客户端较少的特殊号码，然后是 master secret 和填充数据。因此，服务器可以通过对收到的密文进行解密来确认收到的 Finished 消息是否正确，从而可以确认握手协议是否正常结束，密码套件的切换是否正确。

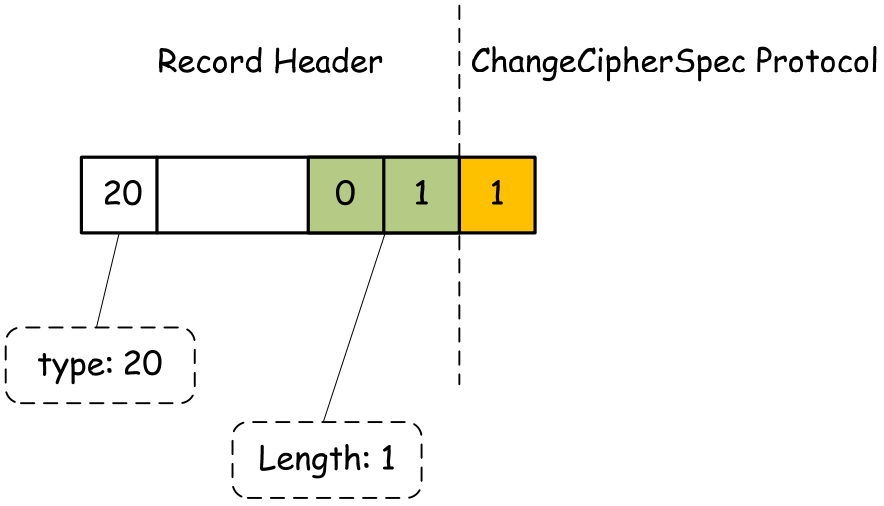
消息格式如下：



（13）ChangeCipherSpec（服务器——>客户端）

服务器发送 ChangeCipherSpec 消息，告诉客户端：“好，我开始切换密码套件了。”

消息格式如下：



（14）Finished（服务器——>客户端）

服务器发送 Finished 消息，告诉客户端“握手结束。”

同样地，该消息是通过加密方式传输的。

（15）切换至应用数据协议

从此开始，客户端和服务器使用应用数据协议和 TLS 记录协议进行加密通信。

Change Cipher Specific Protocol

TLS 握手协议的一部分，用于向通信对象传递变更密码规格的信号。这个协议发送的消息，大致相当于如下对话。

客户端：“好，我们按照刚才的约定切换密码吧。1，2，3，Go!”

Change Cipher specific 消息内容只有 1 个字节。由于这个协议是冗余的，在 TLS1.3 中已经删除。由于 TLS1.2 是目前大规模应用的，且安全的传输层安全协议，所以这里还是对这个协议进行了说明。

当协议中途发生错误时，就会通过 Alter protocol 通知对方。

Alert Protocol

Alert Protocol 是 TLS 握手协议的一个子协议，负责在发生错误时，将错误报告给对方。

这个协议所发送的消息，大致相当于下面的对话。

服务器：“刚才的消息无法正确解密哦!”

如果没有错误发生，则会使用 Application Data Protocol 来进行通信。

消息格式定义如下：

```
enum { warning(1), fatal(2), (255) } AlertLevel;

struct {
    AlertLevel level;
    AlertDescription description;
} Alert;
```

其中 level 是警报等级，不同等级有不同的处理方式。

警报协议的消息由两个字段构成，第一个字段（1 个字节），表示严重性级别（severity

level)；第二个字段（1 个字节），表示警报描述。第一个字段，值 1 表示警告（warning)；值 2 表示致命错误(fatal)。如果级别是致命，TLS 将立即终止连接。第二个字节包含描述特定警报信息的代码。

Alter severity	Dec	Hex
WARNING	1	0X01
FATAL	2	0X02

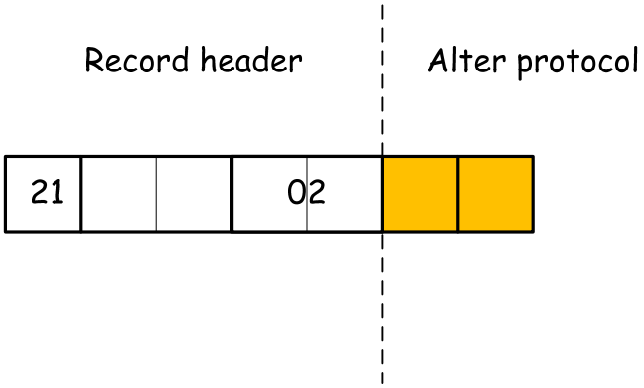
导致致命错误的警报：

- 意外消息（unexpected_message)：接收到不正确的消息
- MAC 记录出错（bad_record_mac)：接收到不正确的 MAC
- 解压失败（decompression_failure)：解压缩函数接收到不正确的输入（比如：不能正确解压或者解压长度大于允许值的数据长度)。
- 握手失败（handshake_failure)：发送者无法在给定选项中协商出一个可以接受的安全参数集。
- 非法参数（illegal_parameter)：握手消息中的某个域超出范围或者与其他域不一致。

其余警报如下：

- 结束通知（close_notify)：通知接收者，发送者将不再用此连接发送任何消息。各方在关闭连接的写端时均需发送结束通知。
- 无证书（no_certificate)：如果无适当的证书可用，在给证书请求者的相应中发送此警报。
- 证书出错（bad_certificate)：接收的证书被破坏（如签名无法通过验证)。
- 不支持的证书（unsupported_certificate)：证书不被支持。
- 证书撤销（certificate_revoked)：证书已经被其签名者撤销。
- 证书过期（certificate_expired)：证书已经过期
- 未知整数（certificate_unknown)：在处理证书时，出现其他错误，使得证书不被接受。

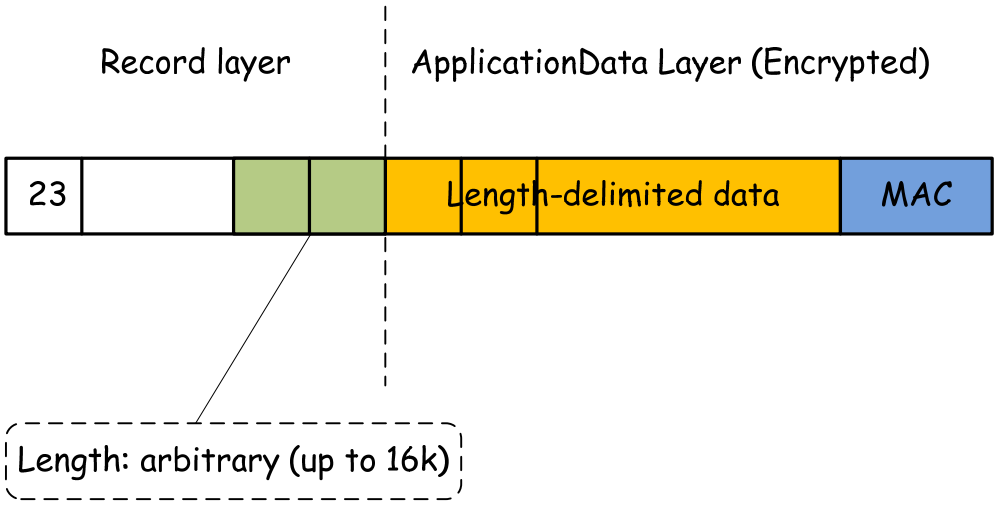
消息格式如下：



Application Data Protocol

Application Data Protocol 的作用是封装来自应用层的数据，以便能够由下层协议（TCP）无缝处理（无需修改协议栈中现有的协议）。具体功能包括将应用层数据输入给 Record 层，做分片、计算 MAC、加密、传输。

消息格式如下：



1.4 【实验步骤】

步骤一、环境搭建

配置两台网络联通的主机，可以分别为 ubuntu linux 操作系统和 windows 操作系统。可以采用如下任何一种方式，其中 linux 主机作为服务器，windows 主机作为客户机。

1) Windows 宿主机安装 wireshark 并配置一台 ubuntu linux 虚拟机，确保宿主机和虚拟机能够网络联通，并测试是否能够抓包。
2) 在虚拟化平台配置两台虚拟机，其中之一为 windows 操作系统（其上配置 wireshark 抓包软件），另一台为 ubuntu linux 操作系统，确保这两台虚拟机网络联通。

步骤二、Linux 服务器上配置 apache 服务器的 HTTPS

1、 创建证书、密钥等文件

命令：

```
sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout
/etc/ssl/private/apache-selfsigned.key                    -out
/etc/ssl/certs/apache-selfsigned.crt
```

openssl: 创建和管理 OpenSSL 证书、密钥等的基本工具。

req：子命令，用于说明要使用 X.509 证书签名请求（CSR）管理。

-x509：进一步说明要创建一个自签证书。

-nodes：用于说明跳过对证书的口令保护，这样就避免每次读取证书文件时需要输入口令。

-days 365：设置证书有效期为 365 天。

-newkey rsa:2048：用于说明要同时生成一个新的证书和新的密钥。rsa:2048 用于说明生成的 RSA key 是 2048 比特。

-keyout：说明私钥文件的存放位置

-out：说明证书文件的存放位置

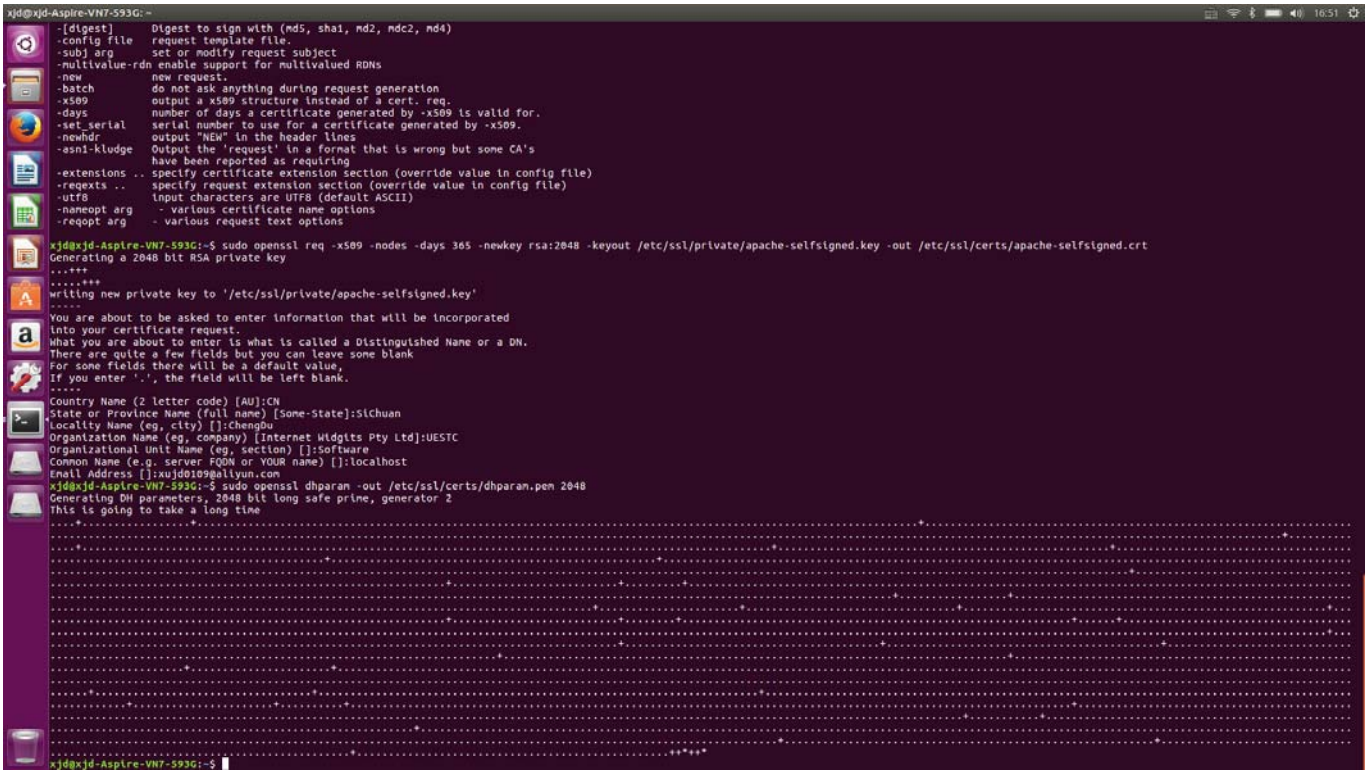
在创建证书的过程中，程序会提示用户输入一些信息，这些信息会嵌入到证书中。最重要的信息是 Common Name，你需要以服务器的域名或者 IP 地址来回答。下面是截图示例：

```
xjd@xjd-Aspire-VN7-593G: ~
 -nameopt arg    - various certificate name options
 -reqopt arg     - various request text options

xjd@xjd-Aspire-VN7-593G:~$ sudo openssl req -x509 -nodes -days 365 -newkey rsa:2
048 -keyout /etc/ssl/private/apache-selfsigned.key -out /etc/ssl/certs/apache-se
lfsigned.crt
Generating a 2048 bit RSA private key
...+++
.....+++
writing new private key to '/etc/ssl/private/apache-selfsigned.key'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:CN
State or Province Name (full name) [Some-State]:SiChuan
Locality Name (eg, city) []:ChengDu
Organization Name (eg, company) [Internet Widgits Pty Ltd]:UESTC
Organizational Unit Name (eg, section) []:Software
Common Name (e.g. server FQDN or YOUR name) []:localhost
```

为了实现前向安全保密（Perfect Forward Secrecy），还需要创建一个强的 DH 群。命令为：

```
sudo openssl dhparam -out /etc/ssl/certs/dhparam.pem 2048
```



2、 配置 Apache 服务器

前面已经准备好密钥和证书文件，下面我们将利用这些文件来配置 apache 服务器。包括：

（1）安装 apache2

命令：sudo apt-get install apache2

启动服务：

sudo service apache2 start

或者：

sudo /etc/init.d/apache2 start

或者

sudo systemctl start apache2.service

上述命令有对应的 stop 和 restart

查看状态：

sudo systemctl status apache2.service

启动之后，可以通过浏览器访问对应的 URL 来验证是否成功。

（2）创建 Apache 配置文件,并进行强加密设置

在 /etc/apache2/conf-available 目录下创建一个新文件，命名为：ssl-params.conf。文件内容为：

```
# from https://cipherli.st/

# and https://raymii.org/s/tutorials/Strong_SSL_Security_On_Apache2.html



SSLCipherSuite EECDH+AESGCM:EDH+AESGCM:AES256+EECDH:AES256+EDH

SSLProtocol All -SSLv2 -SSLv3

SSLHonorCipherOrder On

# Disable preloading HSTS for now.  You can use the commented out header line that includes

# the "preload" directive if you understand the implications.

#Header always set Strict-Transport-Security "max-age=63072000; includeSubdomains; preload"
```

```
Header always set Strict-Transport-Security "max-age=63072000; includeSubdomains"

Header always set X-Frame-Options DENY

Header always set X-Content-Type-Options nosniff

# Requires Apache >= 2.4

SSLCompression off

SSLSessionTickets Off

SSLUseStapling on

SSLStaplingCache "shmcb:logs/stapling-cache(150000)"



SSLOpenSSLConfCmd DHParameters "/etc/ssl/certs/dhparam.pem"
```

需要注意的地方是，把上述文件的最后一行的参数设置为前面生成的 DH 文件。

（详细的信息可参看：https://cipherli.st/）

（3）修改默认的 Apache SSL 虚拟主机文件

默认的 Apache SSL 虚拟主机文件为：

/etc/apache2/sites-available/default-ssl.conf

修改之前，先做一下备份。命令为：

sudo cp /etc/apache2/sites-available/default-ssl.conf /etc/apache2/sites-available/default-ssl.conf.bak

然后，打开/etc/apache2/sites-available/default-ssl.conf 文件进行编辑：

原文件的内容大致如下：

```
<IfModule mod_ssl.c>

	<VirtualHost _default_:443>

		ServerAdmin webmaster@localhost



		DocumentRoot /var/www/html
```

```
                ErrorLog ${APACHE_LOG_DIR}/error.log
                CustomLog ${APACHE_LOG_DIR}/access.log combined

                SSLEngine on

                SSLCertificateFile      /etc/ssl/certs/ssl-cert-snakeoil.pem
                SSLCertificateKeyFile /etc/ssl/private/ssl-cert-snakeoil.key

                <FilesMatch "\.(cgi|shtml|phtml|php)$">
                                SSLOptions +StdEnvVars
                </FilesMatch>
                <Directory /usr/lib/cgi-bin>
                                SSLOptions +StdEnvVars
                </Directory>

                # BrowserMatch "MSIE [2-6]" \
                #               nokeepalive ssl-unclean-shutdown \
                #               downgrade-1.0 force-response-1.0

        </VirtualHost>
</IfModule>
```

修改后的版本如下：

```
<IfModule mod_ssl.c>
        <VirtualHost _default_:443>
                ServerAdmin your_email@example.com
                ServerName server_domain_or_IP

                DocumentRoot /var/www/html
```

```
            ErrorLog ${APACHE_LOG_DIR}/error.log
            CustomLog ${APACHE_LOG_DIR}/access.log combined

            SSLEngine on

            SSLCertificateFile     /etc/ssl/certs/apache-selfsigned.crt
            SSLCertificateKeyFile /etc/ssl/private/apache-selfsigned.key

            <FilesMatch "\.(cgi|shtml|phtml|php)$">
                        SSLOptions +StdEnvVars
            </FilesMatch>
            <Directory /usr/lib/cgi-bin>
                        SSLOptions +StdEnvVars
            </Directory>

            BrowserMatch "MSIE [2-6]" \
                        nokeepalive ssl-unclean-shutdown \
                        downgrade-1.0 force-response-1.0

      </VirtualHost>
</IfModule>
```

注意对上面红色字体部分是修改后的内容。

（4）修改未加密的 Virtual Host file 来自动重定向请求到加密的 Virtual host

为了更好的安全性，通常也推荐设置为自动重定向 http 访问到 https 访问，通过修改配置文件/etc/apache2/sites-available/000-default.conf 来完成。

打开文件进行编辑，仅需要加入一个 Redirect 指令来指向 SSL 版本的站点，示例如下：

```
<VirtualHost *:80>
        . . .

        Redirect "/" "https://your_domain_or_IP/"

        . . .
</VirtualHost>
```

注意上面的红色字体。

3、 设置防火墙（无防火墙设置的忽略此步）

如果系统设置 ufw 防火墙，则需要进行配置，以使得 SSL 流量能够进入。在安装的时候，apache 会在 ufw 注册一些 profile，以方便设置。命令查看：

```
sudo ufw app list
```

示例结果：

```
Available applications:
  Apache
  Apache Full
  Apache Secure
  OpenSSH
```

查看当前的设置。命令：

```
sudo ufw status
```

输出示例：

```
Status: active

To                         Action      From
--                         ------      ----
```

```
OpenSSH                    ALLOW       Anywhere
Apache                     ALLOW       Anywhere
OpenSSH (v6)               ALLOW       Anywhere (v6)
Apache (v6)                ALLOW       Anywhere (v6)
```

为了放行 SSL 流量，执行如下命令：

```
sudo ufw allow 'apache full'
```

然后，在删除一条冗余的 profile：

```
sudo ufw delete allow 'apache'
```

4、使设置生效：

需要使模块 mod_ssl、mod_headers（配置文件中某些设置需要）生效。命令为：

```
sudo a2enmod ssl
sudo a2enmod headers
```

接下来使得 SSL 虚拟主机生效，使用命令：

```
sudo a2ensite default-ssl
```

ssl-params.conf 生效，命令为：

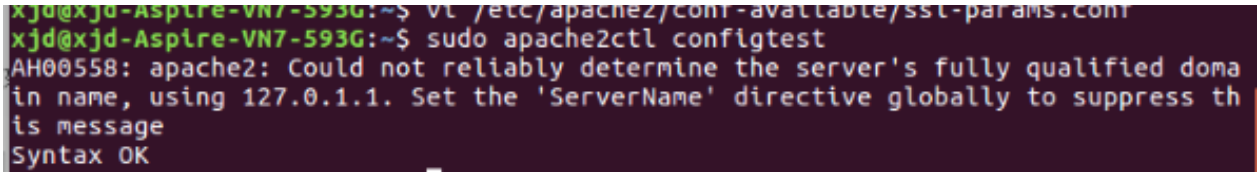
```
sudo a2enconf ssl-params
```

到这一步，服务器已经加载了必要的模块。为了进一步检查配置文件里面是否有语法错误，输入命令：

```
sudo apache2ctl configtest
```

如果都没有问题，示例输出为：

```
AH00558: apache2: Could not reliably determine the server's
fully qualified domain name, using 127.0.1.1. Set the
'ServerName' directive globally to suppress this message
Syntax OK
```



下一步，重启服务器：

```
sudo systemctl restart apache2
```

测试配置结果，从浏览器访问服务器：

```
https://ip
```

因为是自签证书，因此浏览器会弹出警告。

进一步测试 http 重定向 https 设置。

```
http://ip
```

如果测试通过，你确定只允许加密流量，则可以设置永久重定向。编辑文件：

```
/etc/apache2/sites-available/000-default.conf
```

结果如下：

```
<VirtualHost *:80>
        . . .

        Redirect permanent "/" "https://your_domain_or_IP/"

        . . .
</VirtualHost>
```

进一步需要测试配置语法是否正确：

```
sudo apache2ctl configtest
```

重启服务器：

```
sudo systemctl restart apache2
```

步骤三、TLS 流量分析

启动 wireshark，设置显示过滤器。因为只对 TLS 协议数据进行分析，所以这里仅显示 TLS 协议数据即可。设置显示过滤器为：ssl。开启抓包模式。如果浏览器启动后就会访问一些未指明的域名，可以等这一阶段过后，重新开启抓包模式。

启动 firefox 浏览器（或者 IE 浏览器、Chrome 浏览器），访问前面配置好的 apache 服务器或者域名 www.baidu.com（缺省就为 HTTPS）或者 www.amazon.com 等。为了防止 wireshark 占用太多资源，这个时候可以停止抓包功能。

根据 TLS 协议的原理，按照消息交换的顺序对每条消息的关键信息进行截图，并说明每个消息的关键字段的信息。

1.5 【实验报告】

1） 说明实验过程。
2） 进行结果分析。