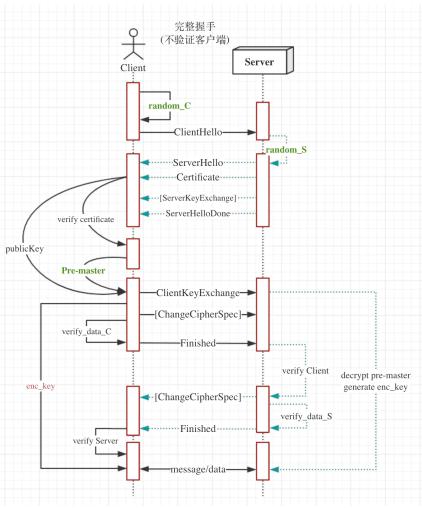
SSL 培训

首先，我们提出一个问题，大家讨论一下：在抛开ssl协议，我们如何能实现数据安全的传递？如何确认传递数据是否完整和是否被篡改？如何确认 发送/接受 数据端的真实身份？

SSL协议里面，客户端和服务器的SSL握手协商实现的工作：

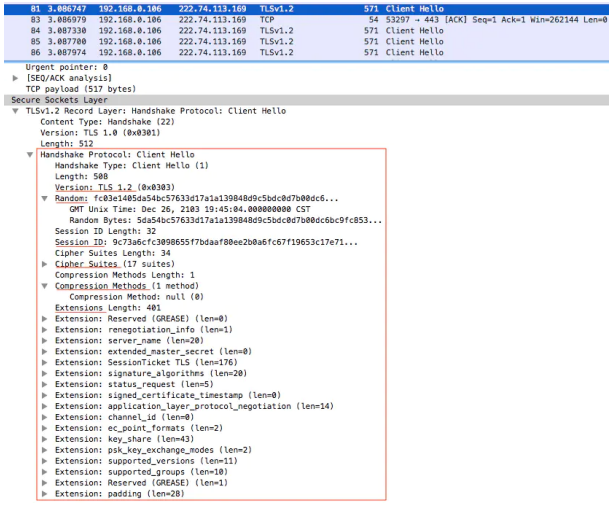
- 交换各自支持的功能，对需要的连接参数达成一致
- 验证出示的证书，或使用其他方式进行身份验证
- 对将用于保护会话的共享主密钥达成一致
- 验证握手消息并未被第三方团体修改

SSl完整握手



ClientHello

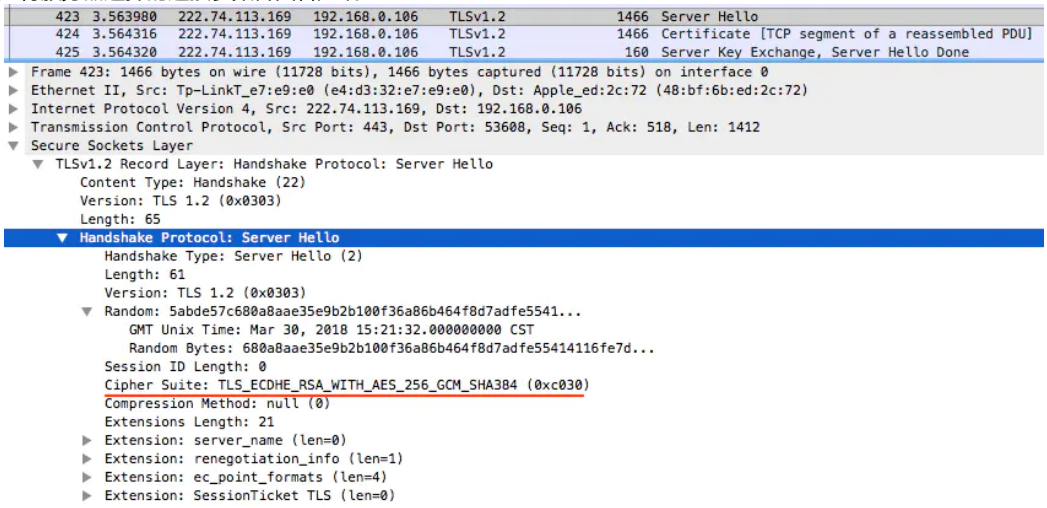
这条消息将客户端的功能和首选项传送给服务器。



- Version: 协议版本（protocol version）指示客户端支持的最佳协议版本
- Random: 一个 32 字节数据，28 字节是随机生成的 (图中的 Random Bytes)；剩余的 4 字节包含额外的信息，与客户端时钟有关 (图中使用的是 GMT Unix Time)。在握手时，客户端和服务器都会提供随机数，客户端的暂记作 random_C (用于后续的密钥的生成)。这种随机性对每次握手都是独一无二的，在身份验证中起着举足轻重的作用。它可以防止 重放攻击，并确认初始数据交换的完整性。
- Session ID: 在第一次连接时，会话 ID（session ID）字段是空的，这表示客户端并不希望恢复某个已存在的会话。典型的会话 ID 包含 32 字节随机生成的数据，一般由服务端生成通过 ServerHello 返回给客户端。
- Cipher Suites: 密码套件（cipher suite）块是由客户端支持的所有密码套件组成的列表，该列表是按优先级顺序排列的
- Compression: 客户端可以提交一个或多个支持压缩的方法。默认的压缩方法是 null，代表没有压缩
- Extensions: 扩展（extension）块由任意数量的扩展组成。这些扩展会携带额外数据

ServerHello

是将服务器选择的连接参数传回客户端。



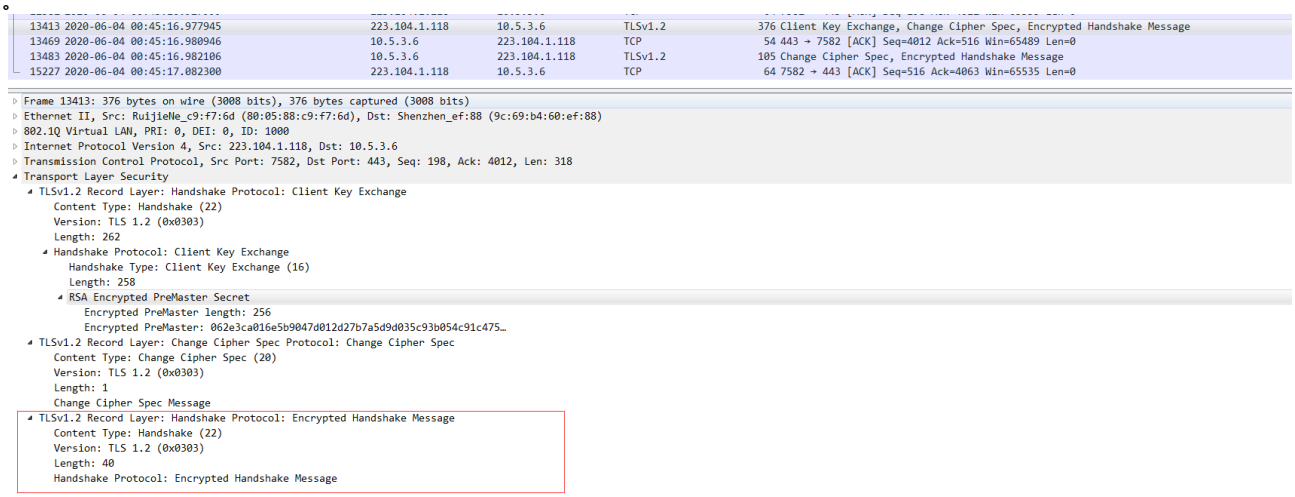
这个消息的结构与 ClientHello 类似，只是每个字段只包含一个选项，其中包含服务端的 random_S 参数
(用于后续的密钥协商)。服务器无需支持客户端支持的最佳版本。如果服务器不支持与客户端相同的版本，可以提供某个其他版本以期待客户端能够接受

图中的 `Cipher Suite` 是后续密钥协商和身份验证要用的加密套件，此处选择的密钥交换与签名算法是 ECDHE_RSA，对称加密算法是 AES-GCM，后面会讲到这个

还有一点默认情况下 TLS 压缩都是关闭的，因为 CRIME 攻击会利用 TLS 压缩恢复加密认证 cookie，实现会话劫持，而且一般配置 gzip 等内容压缩后再压缩 TLS 分片效益不大又额外占用资源，所以一般都关闭 TLS 压缩

Certificate

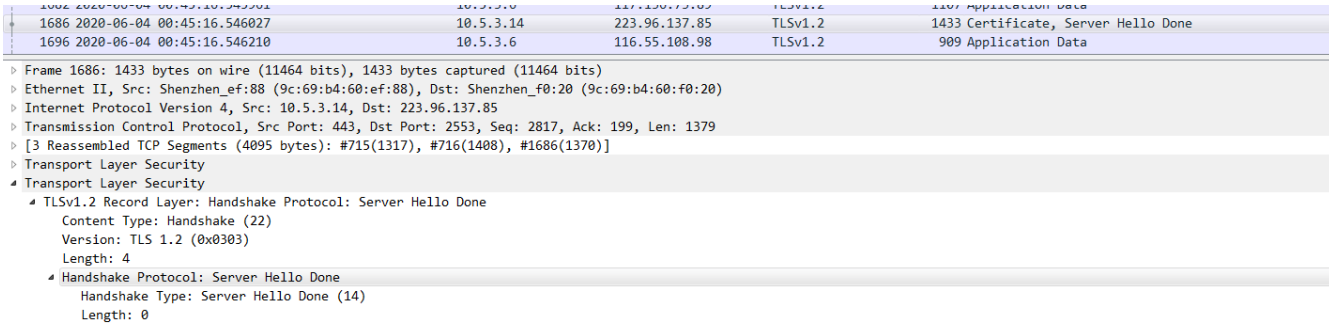
典型的 Certificate 消息用于携带服务器 X.509 证书链。
服务器必须保证它发送的证书与选择的算法套件一致。比方说，公钥算法与套件中使用的必须匹配。除此以外，一些密钥交换算法依赖嵌入证书的特定数据，而且要求证书必须以客户端支持的算法签名。所有这些都表明服务器需要配置多个证书（每个证书可能会配备不同的证书链）。



Certificate 消息是可选的，因为并非所有套件都使用身份验证，也并非所有身份验证方法都需要证书。更进一步说，虽然消息默认使用 X.509 证书，但是也可以携带其他形式的标志；一些套件就依赖 PGP 密钥

ServerHelloDone

表明服务器已经将所有预计的握手消息发送完毕。在此之后，服务器会等待客户端发送消息。

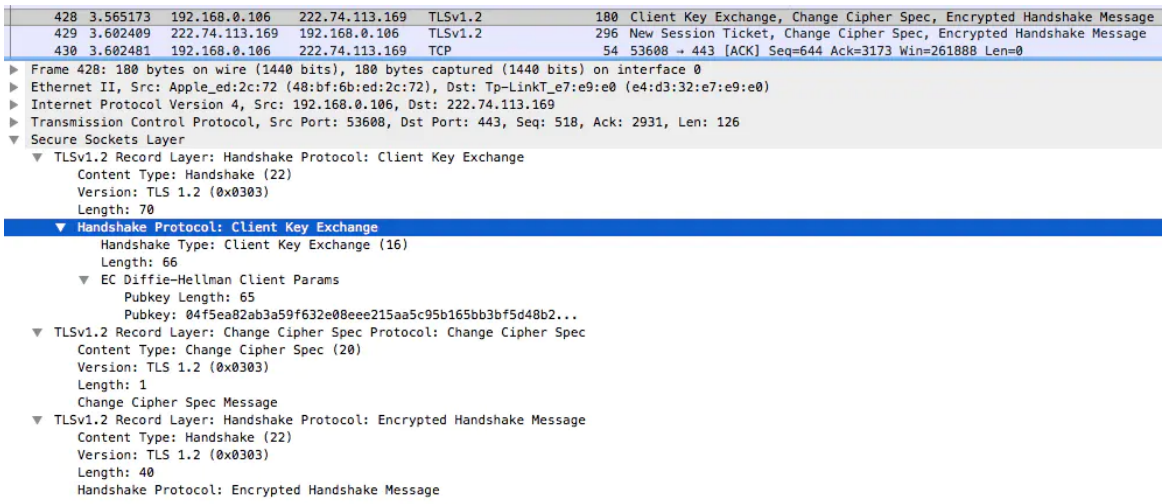


ClientKeyExchange

合法性验证通过之后，客户端计算产生随机数字的预主密钥（Pre-master），并用证书公钥加密，发送给服务器并携带客户端为密钥交换提供的所有信息。这个消息受协商的密码套件的影响，内容随着不同的协商密码套件而不同。

此时客户端已经获取全部的计算协商密钥需要的信息: 两个明文随机数 random_C 和 random_S 与自己计算产生的 Pre-master，然后得到协商密钥(用于之后的消息加密)

```
enc_key = PRF(Pre_master, "master secret", random_C + random_S)
```

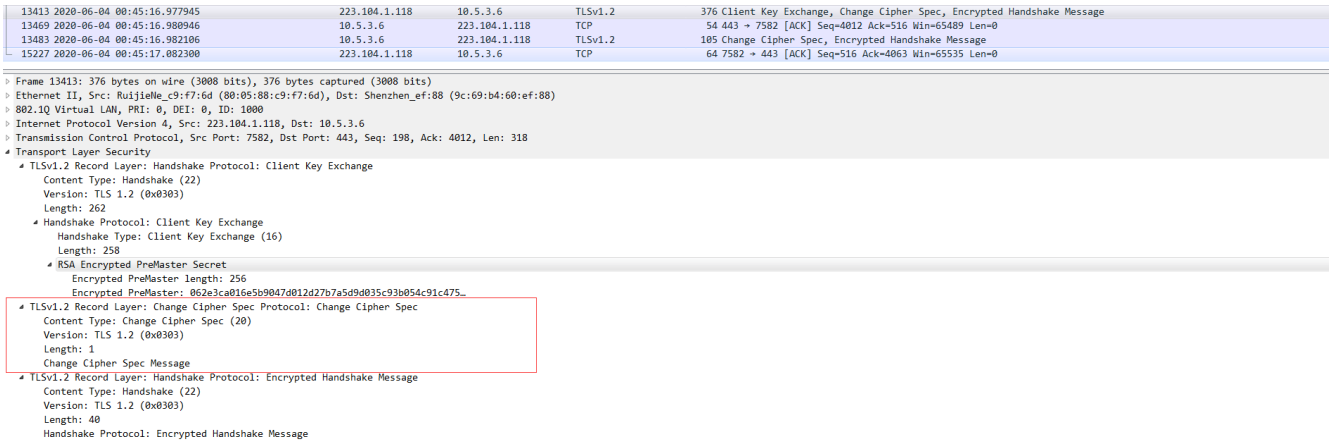


图中使用的是 ECDHE 算法，ClientKeyExchange 传递的是 DH 算法的客户端参数，
如果使用的是 RSA 算法则此处应该传递加密的预主密钥

ChangeCipherSpec

通知服务器后续的通信都采用协商的通信密钥和加密算法进行加密通信.

注意
ChangeCipherSpec 不属于握手消息，它是另一种协议，只有一条消息，作为它的子协议进行实现。



Finished (Encrypted Handshake Message)

Finished 消息意味着握手已经完成。消息内容将加密，以便双方可以安全地交换验证整个握手完整性所需的数据。

这个消息包含 verify_data
字段，它的值是握手过程中所有消息的散列值。这些消息在连接两端都按照各自所见的顺序排列，并以协商得到的主密钥 (enc_key)
计算散列。这个过程是通过一个伪随机函数（pseudorandom function，PRF）来完成的，这个函数可以生成任意数量的伪随机数据。
两端的计算方法一致，但会使用不同的标签（finished_label）：客户端使用 client finished，而服务器则使用 server finished。
`verify_data = PRF(master_secret, finished_label, Hash(handshake_messages))`

因为 Finished 消息是加密的，并且它们的完整性由协商 MAC 算法保证，所以主动网络攻击者不能改变握手消息并对 vertify_data
的值造假。在 TLS 1.2 版本中，Finished 消息的长度默认是 12 字节（96 位），并且允许密码套件使用更长的长度。在此之前的版本，除了
SSL 3 使用 36 字节的定长消息，其他版本都使用 12 字节的定长消息。

ServerKeyExchange

携带密钥交换需要的额外数据。ServerKeyExchange 是可选的，消息内容对于不同的协商算法套件会存在差异。部分场景下，比如使用 RSA
算法时，服务器不需要发送此消息。

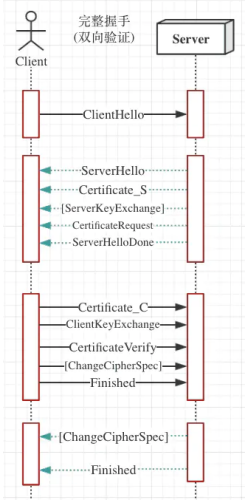
ServerKeyExchange 仅在服务器证书消息（也就是上述 Certificate 消息）不包含足够的数据以允许客户端交换预主密钥（premaster
secret）时才由服务器发送。

比如基于 DH 算法的握手过程中，需要单独发送一条 ServerKeyExchange 消息带上 DH 参数:

```
    425  3.564320   222.74.113.169   192.168.0.106    TLSv1.2              160  Server Key Exchange, Server Hello Done
    426  3.564376   192.168.0.106    222.74.113.169   TCP                   54  53608 → 443 [ACK] Seq=518 Ack=2825 Win=26
    427  3.564376   192.168.0.106    222.74.113.169   TCP                   54  53608 → 443 [ACK] Seq=518 Ack=2931 Win=26
▶ Frame 425: 160 bytes on wire (1280 bits), 160 bytes captured (1280 bits) on interface 0
▶ Ethernet II, Src: Tp-LinkT_e7:e9:e0 (e4:d3:32:e7:e9:e0), Dst: Apple_ed:2c:72 (48:bf:6b:ed:2c:72)
▶ Internet Protocol Version 4, Src: 222.74.113.169, Dst: 192.168.0.106
▶ Transmission Control Protocol, Src Port: 443, Dst Port: 53608, Seq: 2825, Ack: 518, Len: 106
▼ [2 Reassembled TCP Segments (338 bytes): #424(241), #425(97)]
    [Frame: 424, payload: 0-240 (241 bytes)]
    [Frame: 425, payload: 241-337 (97 bytes)]
    [Segment count: 2]
    [Reassembled TCP length: 338]
    [Reassembled TCP Data: 160303014d0c0001490300174104f3bf10a4ec62cc92bb8e...]
▼ Secure Sockets Layer
  ▼ TLSv1.2 Record Layer: Handshake Protocol: Server Key Exchange
      Content Type: Handshake (22)
      Version: TLS 1.2 (0x0303)
      Length: 333
    ▼ Handshake Protocol: Server Key Exchange
        Handshake Type: Server Key Exchange (12)
        Length: 329
      ▼ EC Diffie-Hellman Server Params
          Curve Type: named_curve (0x03)
          Named Curve: secp256r1 (0x0017)
          Pubkey Length: 65
          Pubkey: 04f3bf10a4ec62cc92bb8edf684839e68fe9d3508a1aa7a0...
        ▼ Signature Algorithm: rsa_pkcs1_sha1 (0x0201)
            Signature Hash Algorithm Hash: SHA1 (2)
            Signature Hash Algorithm Signature: RSA (1)
          Signature Length: 256
          Signature: 1a576c92c63176aa2a52757ce8ae75ba03f9e762ebfbf1f0...
▼ Secure Sockets Layer
  ▼ 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
```

客户端身份验证

尽管可以选择对任意一端进行身份验证，但人们几乎都启用了对服务器的身份验证。如果服务器选择的套件不是匿名的，那么就需要在 Certificate 消息中跟上自己的证书。



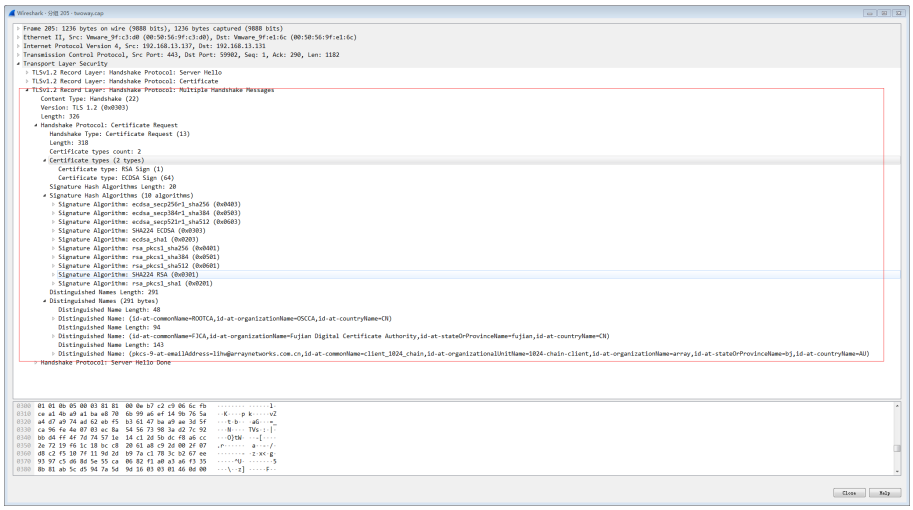
相比之下，服务器通过发送 CertificateRequest
消息请求对客户端进行身份验证。消息中列出所有可接受的客户端证书。作为响应，客户端发送自己的 Certificate
消息（使用与服务器发送证书相同的格式），并附上证书。此后，客户端发送 CertificateVerify 消息，证明自己拥有对应的私钥。

只有已经过身份验证的服务器才被允许请求客户端身份验证。基于这个原因，这个选项也被称为相互身份验证（mutual authentication）。

5.3.1 CertificateRequest

在 ServerHello 的过程中发出，请求对客户端进行身份验证，并将其接受的证书的公钥和签名算法传送给客户端。

它也可以选择发送一份自己接受的证书颁发机构列表，这些机构都用其可分辨名称来表示:



5.3.2 CertificateVerify

在 ClientKeyExchange
的过程中发出，证明自己拥有的私钥与之前发送的客户端证书中的公钥匹配。消息中包含一条到这一步为止的所有握手消息的签名：