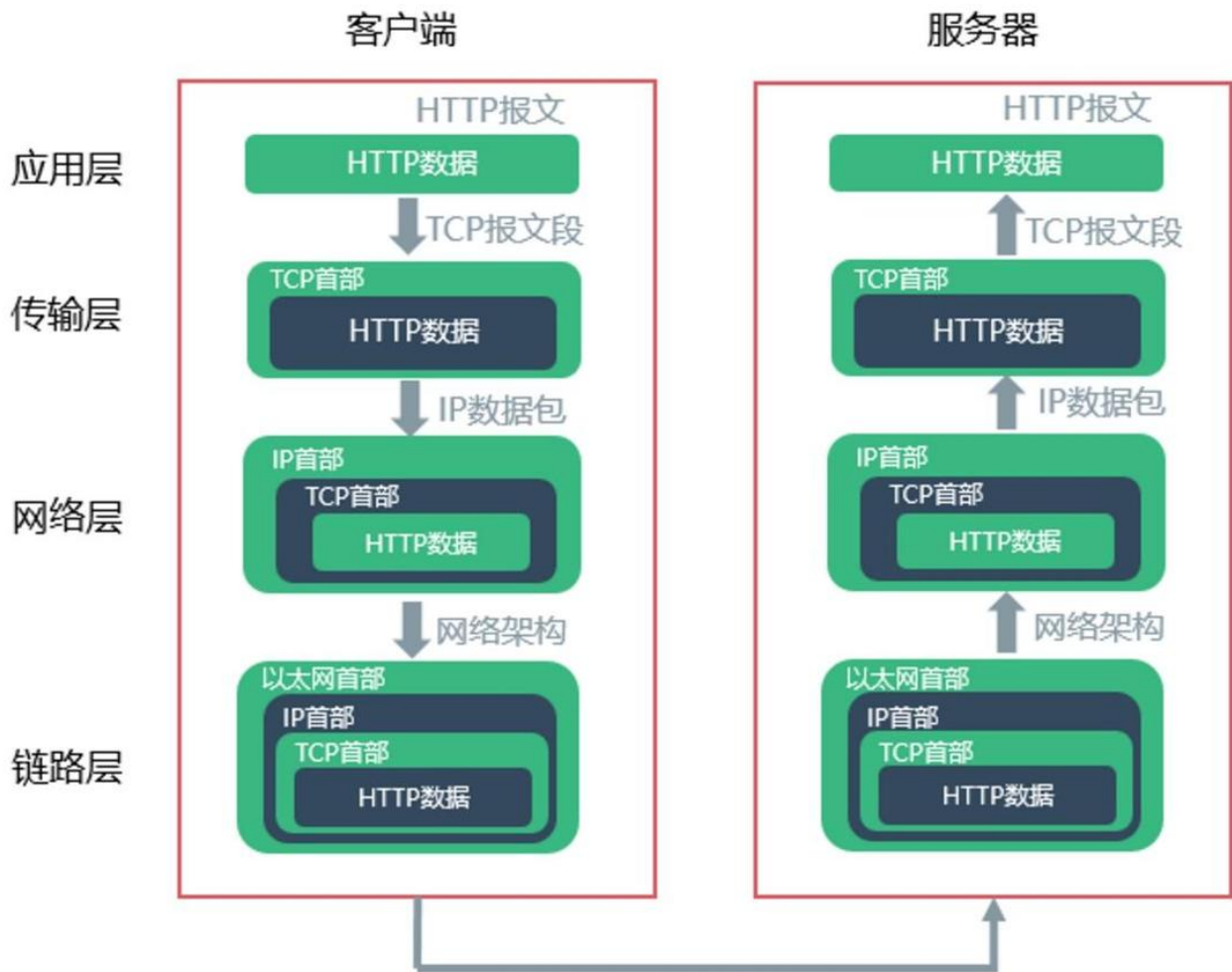


- 文章比较长，赠于有缘人

## HTTPS简介

- HTTPS，是一种网络安全传输协议，在HTTP的基础上利用SSL/TLS来对数据包进行加密,以提供对网络服务器的身份认证，保护交换数据的隐私与完整性。
  - TLS ( Transport Layer Security ) 1.0是SSL ( Secure Sockets Layer ) 3.0的升级版，安全套接字层协议，承担的角色都是一样的，是HTTPS方式握手以及传输数据的一个协议。只是改了名字，其中的八卦，感兴趣的朋友可以自己查。
- HTTP(S)协议是在TCP/IP协议基础上建造的。
- TCP/IP协议的分层管理，按层次分为：应用层、传输层、网络层、数据链路层。（我们常说的四层协议、四层模型就是指的这个啦）
- 没有经过加密层时，数据传输的路径是：应用层->传输层->网络层->数据链路层
- 经过加密层之后，数据传输的路径是：应用层->SSL/TLS层->传输层->网络层->数据链路层



- 每层常见的协议：
  - 应用层协议有：FTP、Telnet、SMTP、HTTP、RIP、NFS、DNS。
  - 传输层协议有：TCP协议、UDP协议。
  - 网络层协议有：IP协议、ICMP协议、ARP协议、RARP协议。

## HTTPS用途

- 防窃听：HTTPS协议对传输数据的内容加密，保障数据在传输过程的安全（加密传播）
- 防冒充：确认网站的真实性（身份证书）
- 防篡改：防止内容被第三方篡改（校验机制）

## HTTPS协议安全性

- HTTPS协议本身是安全的，并且能够保障数据传输安全、两端身份真实、以及检查数据是否被篡改。

- 但近几年有https相关的漏洞频发，如心血漏洞、中间人攻击、DROWN溺水攻击、FREAK漏洞、降维攻击、POODLE等（近期会将每个漏洞原理进行分析）。不由让人从1.0升级为TLS 1.1）。而且频发漏洞的并不是https协议本身，而是各个开源或商业服务在具体实现https时，出现了安全问题。

## https如何进行数据传输的？

- 大致流程是：进行握手流程建立https连接（此时是明文传输），然后再进行真正的数据传输（此时使用对称加密进行密文传输）
- 首先需要了解TLS/SSL协议握手的过程

## 握手过程

- 整个过程，如访问www.baidu.com
- 先进行DNS解析，再建立TCP连接，然后进行https握手，最后传输加密数据。

The image displays two Wireshark packet capture screenshots. The top screenshot shows the TLS handshake process for an HTTPS connection to 103.235.46.39. Key packets include Client Hello (255 bytes), Server Hello (1514 bytes), Certificate (1150 bytes), Server Key Exchange (54 bytes), Client Key Exchange (180 bytes), and Encrypted Handshake Message (491 bytes). The bottom screenshot shows the DNS response for the domain www.baidu.com, which includes a CNAME record pointing to www.a.shifen.com. Red annotations explain the CNAME record and the subsequent connection to the actual IP address.

**Frame 16: 255 bytes on wire (2040 bits), 255 bytes captured (2040 bits)**

- Ethernet II, Src: Apple\_2a:d5:d9 (8c:85:90:2a:d5:d9), Dst: D-LinkIn\_1b:e0:30 (10:be:f5:1b:e0:30)
- Internet Protocol Version 4, Src: 192.168.0.145, Dst: 103.235.46.39
- Transmission Control Protocol, Src Port: 53189, Dst Port: 443, Seq: 1, Ack: 1, Len: 201
- Secure Sockets Layer
  - Client Hello

**Frame 12: 116 bytes on wire (928 bits), 116 bytes captured (928 bits)**

- Ethernet II, Src: D-LinkIn\_1b:e0:30 (10:be:f5:1b:e0:30), Dst: Apple\_2a:d5:d9 (8c:85:90:2a:d5:d9)
- Internet Protocol Version 4, Src: 192.168.0.1, Dst: 192.168.0.145
- User Datagram Protocol, Src Port: 53, Dst Port: 18137
- Domain Name System (response)
  - Request In: 11
  - [Time: 0.006716000 seconds]
  - Transaction ID: 0xd5f1
  - Flags: 0x8180 Standard query response, No error
  - Questions: 1
  - Answer RRs: 2
  - Authority RRs: 0
  - Additional RRs: 0
  - Queries
    - www.baidu.com: type CNAME, class IN, cname www.a.shifen.com
    - www.a.shifen.com: type A, class IN, addr 103.235.46.39

**Q: 为什么会有CNAME地址呢?**  
**A:** 现在比较常见的做法不是将域名记录直接指向IP地址，而是先指向另一个域名，再指向IP地址。这中间的域名就是CNAME地址。

比如你使用了新浪的云托管服务部署了网站，新浪云不会直接给你IP，而是xxx.sinaapp.com的域名，你可以直接使用这个域名，不过哪天你又想把网站迁移到其他云服务上，访客又需要记忆新的域名。这时你可以自己购买一个域名abc.com，将域名解析设置CNAME地址解析到xxx.sinaapp.com。当更换了云服务，又可以把这个CNAME地址修改了，而你的网站访客只需要访问abc.com。

握手消息

动作描述

消息内容

1. Client —> ClientHello —> Server

2. Server —> ServerHello —> Client

3. Server —> Certificate —> Client

4. Server --> ServerKeyExchange —> Client

5. Server —> ServerHelloDone —> Client

6. Client —> ClientKeyExchange —> Server

7. Client —> ChangeCipherSpec —> Server

8. Client —> Finished —> Server

9. Server —> ChangeCipherSpec —> Client

10. Server —> Finished —> Client

11. Application Data

客户端(浏览器)发送一个hello消息给服务端，发起建立SSL会话的请求，并告诉服务端，自己支持哪些加密算法(Cipher Suite

1)支持的协议版本，如TLS 1.0版<br/>2)由客户端生成的随机数，用于生成后面的“对称密钥”

List)。除此之外，还需要产生一个随机数（第一个随机数，用于以后生成对称密钥），发送给服务端。

服务端的首次响应，会确定加密协议版本，以及加密的算法。也生成一个随机数，发送给客户端。

1) 服务端证书<br/> 证书颁发机构的名称<br/>

服务端的数字签名也可被用于持有者握手过程中生成的对称密钥

证书签名用到的Hash算法<br/>

指定使用哪种密钥协商协议。服务端可以在ServerKeyExchange消息中使用椭圆曲线Diffie-Hellman

服务器发送ServerHelloDone消息，告知客户端服务器这边握手相关的消息发送完毕。

消息中包含客户端这边的EC Diffie-Hellman算法相关参数，然后服务器和客户端都可根据接收到的对方参数和自身参数运算出对称密钥。

1) 密钥协商时服务端需要的信息

ChangeCipherSpec消息，通知服务器此消息以后客户端将以加密方式传输数据

客户端计算生成对称密钥，然后使用该对称密钥加密之前所有收发握手消息的Hash值，发送给服务器，服务器将相同

ChangeCipherSpec消息，通知客户端此消息以后服务器将以加密方式传输数据

服务器使用对称密钥加密（生成方式与客户端相同）之前所发送的所有握手消息的hash值，发送给客户端去校验。

真正的数据传输（使用对称加密）

## 1. Client Hello

- 客户端发起TLS握手请求

```
struct {
    ProtocolVersion client_version;
    Random random;
    SessionID session_id;
    CipherSuite cipher_suites<2..2^16-2>;
    CompressionMethod compression_methods<1..2^8-1>;
    select (extensions_present) {
        case false:
            struct {};
        case true:
            Extension extensions<0..2^16-1>;
    };
} ClientHello;
```

- 数据包括内容：

- ProtocolVersion / 协议版本（客户端期望支持的握手[协议版本](#)）
  - Random / 安全随机数（MasterSecret生成用到，协议文档里面说是28个字节，但是实际抓包看到是32个字节，这里怀疑是各个协议文档版本不同，还有使用加密套件
  - SessionID / 会话ID
    - 这个值是被服务端设置的，如果这个值为空，表示客户端与服务端没有存活的https会话，需要与服务端进行完整的握手。
    - 如果这个值存在，则表明客户端期望恢复上一次的https会话，这时候客户端与服务端只需要进行快速的握手过程。（这里我们只会分析完整的握手过程进行学习）
  - CipherSuite / 加密套件（客户端支持的加密套件列表）
    - 如果sessionid不为空，可以不传这个值，服务端可以从上一次会话中恢复这个值。
- 每个加密组件(Cipher Suite)都包括了下面5类算法 [TLS Cipher Suite Registry](#)，图中百度使用的是就是[TLS\\_ECDHE\\_RSA\\_WITH\\_AES\\_128\\_GCM\\_SHA256](#) 这个加密套件：
- 1、authentication (认证算法)：RSA
  - 2、encryption (加密算法)：AEAD\_AES\_128\_GCM
  - 3、message authentication code (消息认证码算法 简称MAC)：SHA256
  - 4、key exchange (密钥交换算法)：ECDHE
  - 5、key derivation function（密钥衍生算法）
- CompressionMethod / 压缩方法
    - 加密前进行数据压缩
    - 因为压缩方法被攻击，在TLS1.3协议版本上已经彻底禁止压缩了。（这里有两种攻击方式BREACH、CRIME，有时间博主会来研究）
  - Extension / 扩展数据（session ticket在扩展里面，可见下图）



- 消息内容如下图：

Wireshark · Packet 16 · https\_baidu

Frame 16: 255 bytes on wire (2040 bits), 255 bytes captured (2040 bits)

Ethernet II, Src: Apple\_2a:d5:d9 (8c:85:90:2a:d5:d9), Dst: D-LinkIn\_1b:e0:30 (10:be:f5:1b:e0:30)

Internet Protocol Version 4, Src: 192.168.0.145, Dst: 103.235.46.39

Transmission Control Protocol, Src Port: 53189, Dst Port: 443, Seq: 1, Ack: 1, Len: 201

Secure Sockets Layer

▼ TLSv1.2 Record Layer: Handshake Protocol: Client Hello

Content Type: Handshake (22)

Version: TLS 1.0 (0x0301)

Length: 196

▼ Handshake Protocol: Client Hello

Handshake Type: Client Hello (1)

Length: 192

Version: TLS 1.2 (0x0303)

Random: ea100ce266a198303aab77fda15c4a6f46158a4ef558728e... 由客户端生成的32个字节的安全随机数

Session ID Length: 0

Cipher Suites Length: 28

Cipher Suites (14 suites) 客户端也就是浏览器支持的加密算法

Compression Methods Length: 1

Compression Methods (1 method) 用于加密前的压缩方法

Extensions Length: 123

Extension: Reserved (GREASE) (len=0) 保留字段

Extension: renegotiation\_info (len=1) 密钥协商协议

Extension: server\_name (len=18) 请求的域名信息

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

Extension: SessionTicket TLS (len=0) 服务端支持session ticket机制的话, 将会把session ticket返回, 此数据存储在客户端

Extension: signature\_algorithms (len=20) 客户端支持的签名hash算法

Extension: status\_request (len=5)

Extension: signed\_certificate\_timestamp (len=0) 签名证书时间戳

Extension: application\_layer\_protocol\_negotiation (len=14) 应用层协议HTTP

Extension: channel\_id (len=0)

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

Extension: supported\_groups (len=10)

Extension: Reserved (GREASE) (len=1)

session id: 是被服务端定义的, 如果客户端hello消息中的session id不为空, 说明两端曾经握手成功过。服务端将以前协商好的信息存储起来, 快速进行握手过程。session id 为空, 说明没有记录的会话, 需要完整的进行握手。

每个加密组件(Cipher Suite)都包括了下面5类算法:

1. authentication (认证算法)
2. encryption (加密算法)
3. message authentication code (消息认证码算法 简称MAC)
4. key exchange (密钥交换算法)
5. key derivation function (密钥衍生算法)

No.: 16 · Time: 6.190224 · Source: 192.168.0.145 · Destination: 103.235.46.39 · Protocol: TLSv1.2 · Length: 255 · Info: Client Hello

Help Close

## 2. Server Hello

- 服务端回应Client Hello请求

```
struct {
    ProtocolVersion server_version;
    Random random;
    SessionID session_id;
    CipherSuite cipher_suite;
    CompressionMethod compression_method;
    select (extensions_present) {
        case false:
            struct {};
        case true:
            Extension extensions<0..2^16-1>;
    };
} ServerHello;
```

- 主要发送数据内容：

- ProtocolVersion / 握手协议版本
  - 服务端最高支持的握手协议版本，TLS / SSL协议都是向下兼容的。
- Random / 随机数
  - 服务端生成32字节安全随机数 ( MasterSecret生成会用到 )
- SessionID / 会话ID
  - 如果客户端hello有发送session id，服务端从内存中查找，并尝试恢复之前的会话状态。
    - 恢复成功，服务端返回同样的session id。
    - 恢复不成功，服务端此字段返回空。
- CipherSuite / 加密组件
  - 服务端从客户端hello的cipher suite列表选择一个加密套件，如果是恢复上一次的会话，则从会话状态中恢复上一次相同的加密套件。
- CompressionMethod / 压缩方法
  - 服务端从客户端hello的compression\_methods列表选择一个压缩方法，如果是恢复上一次的会话，则从会话状态中恢复上一次相同的压缩方法。
- Extension / 扩展 ( 如下图 )

- 消息如下面所示：

Wireshark · Packet 18 · https\_baidu

Frame 18: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits)

Ethernet II, Src: D-LinkIn\_1b:e0:30 (10:be:f5:1b:e0:30), Dst: Apple\_2a:d5:d9 (8c:85:90:2a:d5:d9)

Internet Protocol Version 4, Src: 103.235.46.39, Dst: 192.168.0.145

Transmission Control Protocol, Src Port: 443, Dst Port: 53189, Seq: 1, Ack: 202, Len: 1460

Secure Sockets Layer

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

Content Type: Handshake (22)

Version: TLS 1.2 (0x0303)

Length: 80

Handshake Protocol: Server Hello

Handshake Type: Server Hello (2)

Length: 76

Version: TLS 1.2 (0x0303) **TLS 1.2版本**

Random: ccc25289593748896e1e254345631dfc6d6fb055c9e46ca... **服务端产生的32个字节随机数**

Session ID Length: 0

Cipher Suite: TLS\_ECDHE\_RSA\_WITH\_AES\_128\_GCM\_SHA256 (0xc02f) **服务端从客户端hello的 cipher suite 中选择一个加密套件, 如果是恢复上一次的会话, 则返回于上一次相同的加密套件。**

Compression Method: null (0) **加密前的压缩方法: null 没有**

Extensions Length: 36

Extension: server\_name (len=0)

Extension: renegotiation\_info (len=1) **密钥协商协议**

Extension: ec\_point\_formats (len=4)

Extension: SessionTicket TLS (len=0) **Session Ticket 空, 表明服务端支持这种方式, 并期望发起**

Extension: application\_layer\_protocol\_negotiation (len=11) **应用层协议HTTP**

**Session ID:**

1) 如果客户端hello有发送session id, 服务端从内存中查找, 并尝试恢复之前的会话状态。

1.1) 恢复成功, 服务端返回同样的session id。

1.2) 恢复不成功, 服务端此字段返回空。

### 3. Server Certificate

- 服务端发送的是一个证书链, 可能包含多个证书。
  - 第一个证书为网站的证书。
  - 第二个证书为颁发证书给网站的机构的证书。
  - 在这个例子中第三个证书是CA机构的根证书, 可以忽略不用发送, 因为这个CA的根证书是CA自己给自己颁发的。这里构成了一个证书信任链, 也就是 GlobalSign Root CA信任GlobalSign Organization Validation CA, 而他又信任baidu.com的证书。如下图所示:

GlobalSign Root CA

GlobalSign Organization Validation CA - SHA256 - G2

baidu.com

**baidu.com**

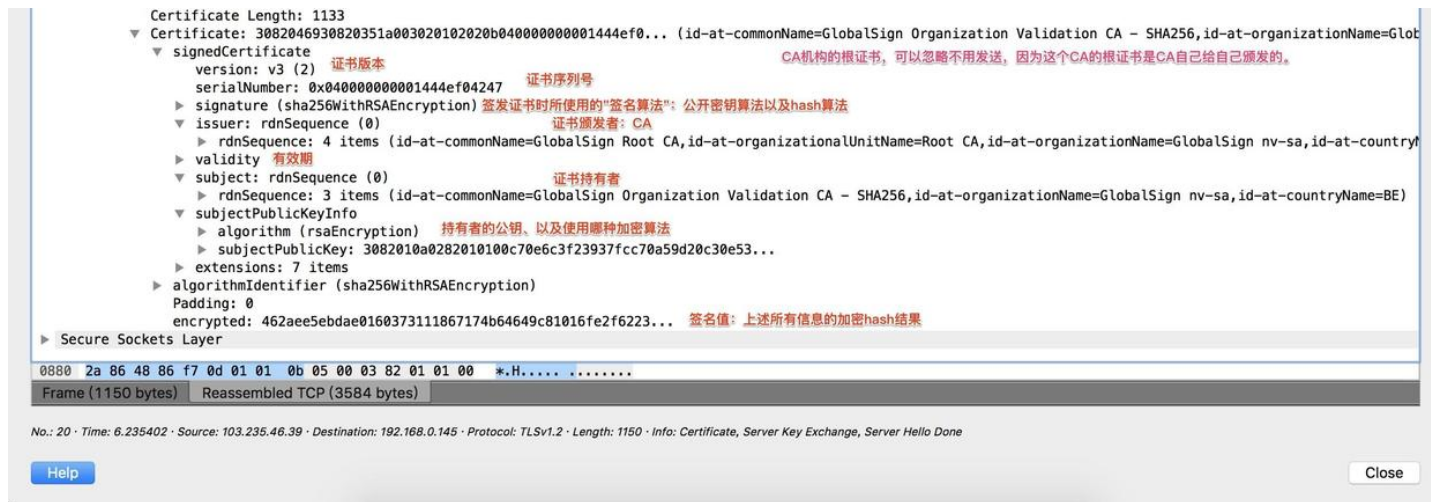
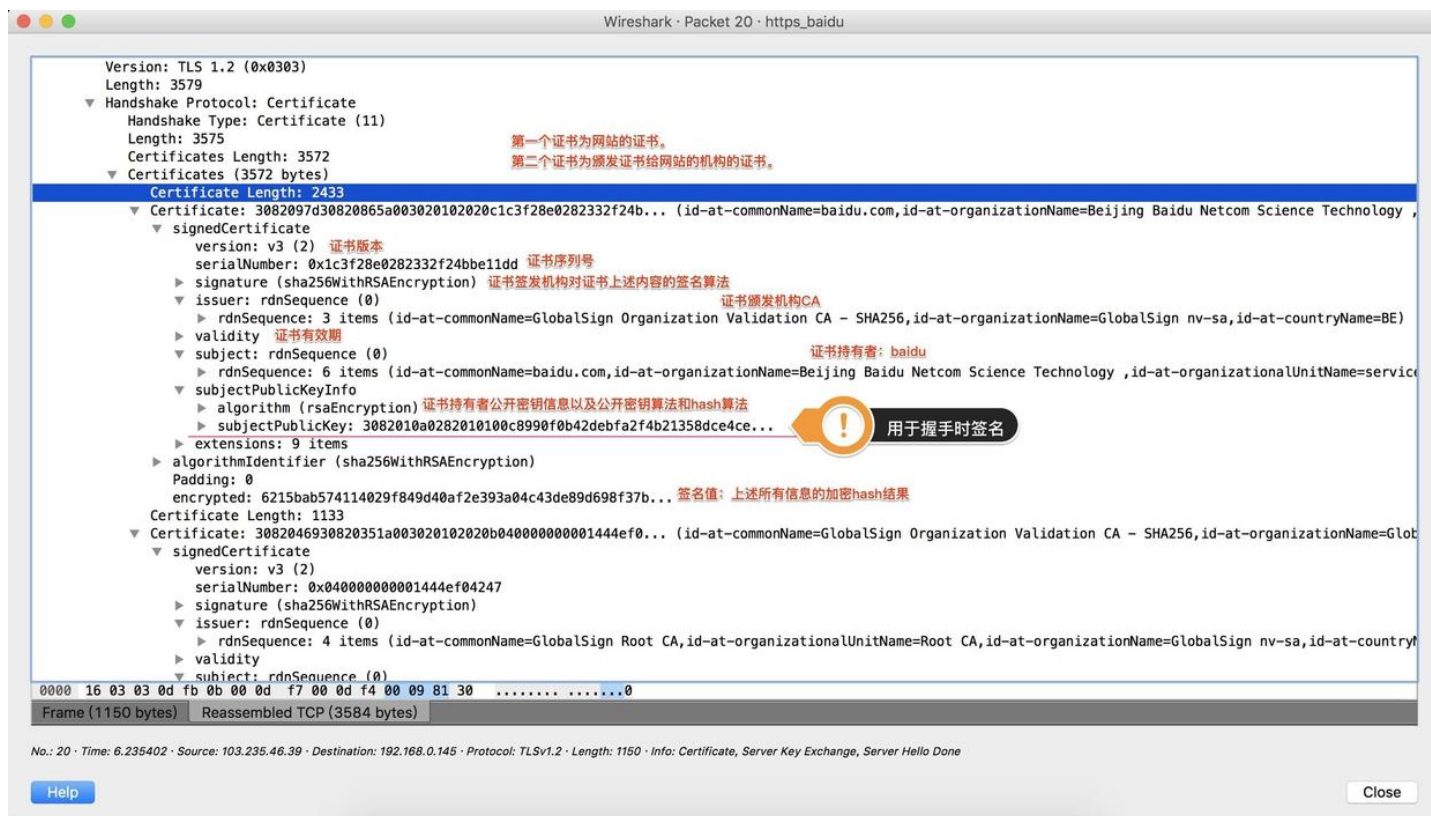
签发者: GlobalSign Organization Validation CA - SHA256 - G2

过期时间: 2018年4月25日星期三 新加坡标准时间 13:31:02

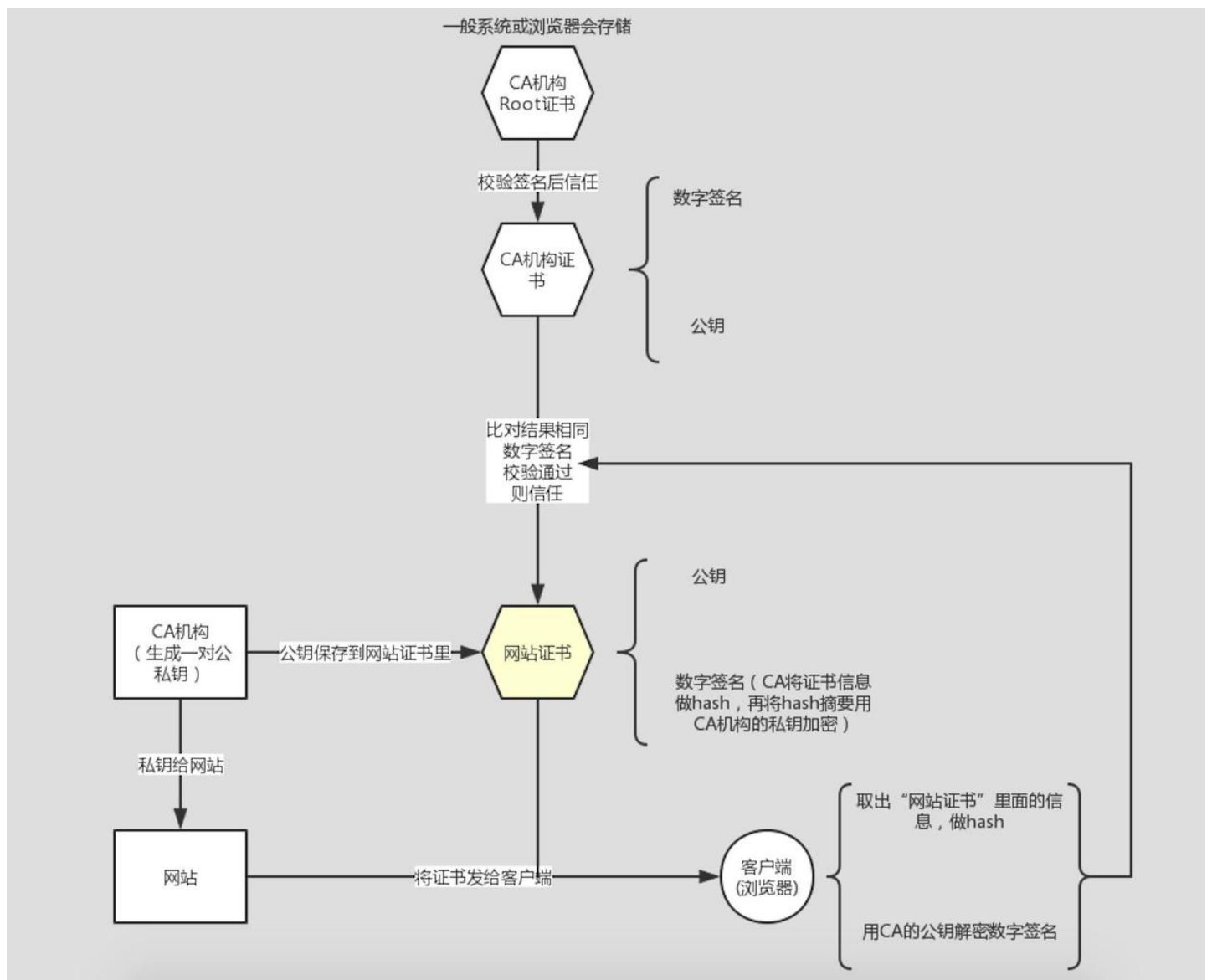
此证书有效

- CA证书的类型有3类: DV ( domain validation), OV ( organization validation), EV ( extended validation), 证书申请难度从前往后递增。
- 证书中都包含了哪些信息?
  - html
- 证书版本号(Version)
- 证书序列号(Serial Number)
- 签名算法标识符(Signature Algorithm)
  - 签名算法标识用来指定由CA签发证书时所使用的"签名算法"。算法标识符用来指定CA签发证书时所使用的:
  - 1) 公开密钥算法
  - 2) hash算法
  - example: sha256WithRSAEncryption
  - 须向国际知名标准组织(如ISO)注册
- 签发机构名(Issuer)
- 有效期(Validity): 指定证书的有效期
- 证书用户名(Subject)
- 证书持有者公开密钥信息(Subject Public Key Info)
  - 证书持有者公开密钥信息域包含两个重要信息:
  - 1) 证书持有者的公开密钥的值
  - 2) 公开密钥使用的算法标识符。此标识符包含公开密钥算法和hash算法。
- 扩展项(extension)
- 签发者唯一标识符(Issuer Unique Identifier)

- 证书持有者唯一标识符(Subject Unique Identifier)
- 签名算法(Signature Algorithm)
- 签名值(Issuer's Signature)
- ...
- 如下图所示



如何校验服务端证书呢？



- 签名的生成：CA先将证书信息生成hash摘要，然后再用CA机构的私钥进行加密。
- 签名的校验：使用CA机构的公钥进行解密签名，得到hash摘要A，再计算证书信息的hash摘要B。比对是否一致。  
#### 详细解释服务端的证书是怎么生成的？
- 服务端的证书是由CA (Certificate Authority, 证书认证中心)颁发的。他是一个负责发放和管理数字证书的第三方权威机构，它负责管理PKI (Public Key Infrastructure, 公开密钥基础设施) 结构下的所有用户(包括各种应用程序)的证书，把用户的公钥和用户的其他信息捆绑在一起，在网上验证用户的身份。
- 一般情况下网站方向CA申请一个证书。CA会给网站方生成一对非对称加密的公钥和私钥，公钥会做到证书里面，私钥则会给到网站方。
- CA会先做一个“数字签名” (生成过程：明文 --> hash运算 --> 摘要 --> 私钥加密 --> 数字签名)
  - 就是将网站方信息、网站方公钥、签名算法、签名值等信息（就是Wireshark Packet 20中的数据，除了“签名值”），计算一个hash值（图中hash算法是SHA256），然后CA再用自己私钥做加密（图中公开密钥算法是RSA），最后的这个密文就是“数字签名”。
- CA最后将“网站方信息”、“网站方公钥”、“签名算法”、“签名值”都做到证书里面（就是Wireshark Packet 20中的我们看到那些数据），证书就做好了，CA会把“证书”和“网站方的私钥”给到网站方。

CA怎么验证证书是不是自己颁发的呢？以及做证书内容校验？

- 首先浏览器（校验网站的证书）或操作系统（校验应用的证书），会在操作系统存储的系统信任的根证书里面去查找“证书颁发机构”是否是信任的。如下图系统根证书：



点按以解锁“系统根证书”钥匙串。

AAA Certificate Services

根证书颁发机构

过期时间：2029年1月1日星期一 新加坡标准时间 07:59:59

此证书有效

名称	种类	过期时间	钥匙串
AAA Certificate Services	证书	2029年1月1日 07:59:59	系统根证书
Actalis Authentication Root CA	证书	2030年9月22日 19:22:02	系统根证书
AddTrust Class 1 CA Root	证书	2020年5月30日 18:38:31	系统根证书
AddTrust External CA Root	证书	2020年5月30日 18:48:38	系统根证书
Admin-Root-CA	证书	2021年11月10日 15:51:07	系统根证书
AffirmTrust Commercial	证书	2030年12月31日 22:06:06	系统根证书
AffirmTrust Networking	证书	2030年12月31日 22:08:24	系统根证书
AffirmTrust Premium	证书	2040年12月31日 22:10:36	系统根证书
AffirmTrust Premium ECC	证书	2040年12月31日 22:20:24	系统根证书
ANF Global Root CA	证书	2033年6月6日 01:45:38	系统根证书
Apple Root CA	证书	2035年2月10日 05:40:36	系统根证书
Apple Root CA - G2	证书	2039年5月1日 02:10:09	系统根证书
Apple Root CA - G3	证书	2039年5月1日 02:19:06	系统根证书
Apple Root Certificate Authority	证书	2025年2月10日 08:18:14	系统根证书
ApplicationCA	证书	2017年12月12日 23:00:00	系统根证书
ApplicationCA2 Root	证书	2033年3月12日 23:00:00	系统根证书
Atos TrustedRoot 2011	证书	2031年1月1日 07:59:59	系统根证书
Autoridad de Certificacion Firmaprofesional CIF A62634068	证书	2030年12月31日 16:38:15	系统根证书
Autoridad de Certificacion Raiz del Estado Venezolano	证书	2030年12月18日 07:59:59	系统根证书
Baltimore CyberTrust Root	证书	2025年5月13日 07:59:00	系统根证书
Belgium Root CA2	证书	2021年12月15日 16:00:00	系统根证书
Buypass Class 2 Root CA	证书	2040年10月26日 16:38:03	系统根证书
Buypass Class 3 Root CA	证书	2040年10月26日 16:28:58	系统根证书
CA Disig Root R1	证书	2042年7月19日 17:06:56	系统根证书
CA Disig Root R2	证书	2042年7月19日 17:15:30	系统根证书
Certigna	证书	2027年6月29日 23:13:05	系统根证书
Certinomis - Autorité Racine	证书	2028年9月17日 16:28:59	系统根证书
Certinomis - Root CA	证书	2033年10月21日 17:17:18	系统根证书
certSIGN ROOT CA	证书	2031年7月5日 01:20:04	系统根证书

\* CA

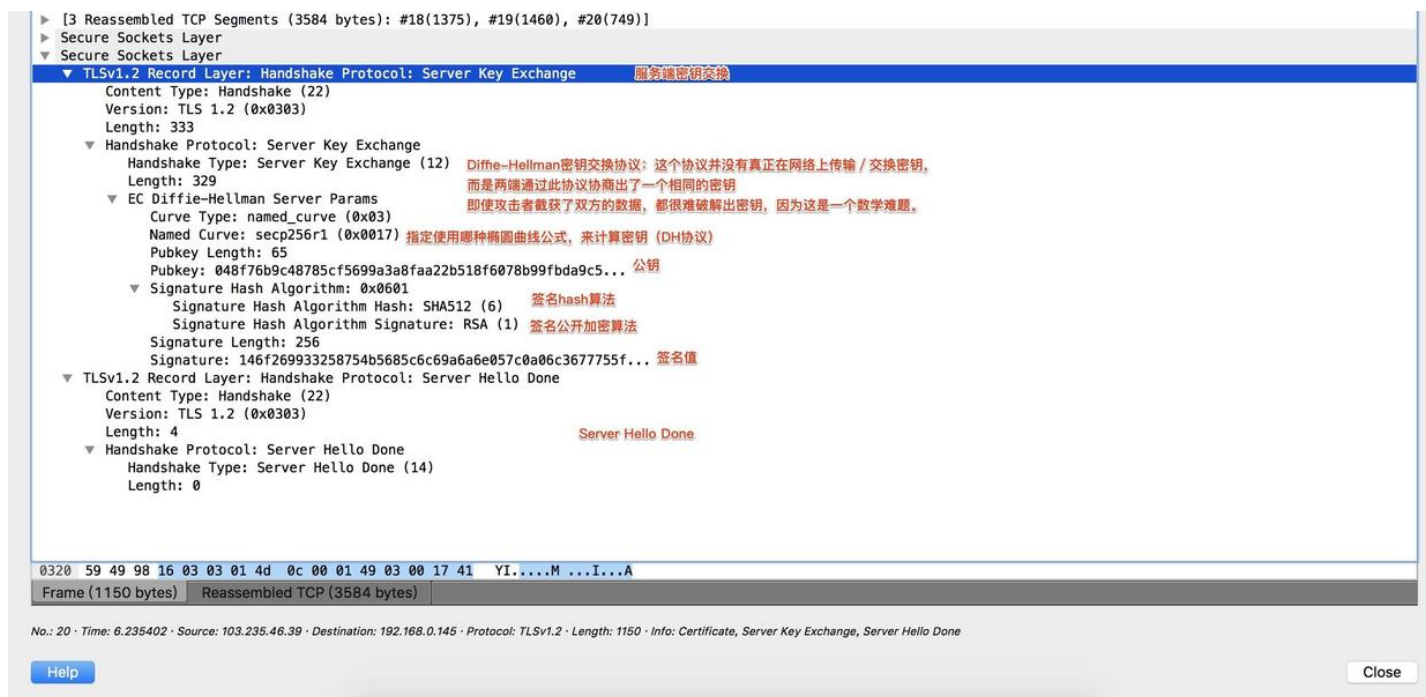
\* CA

- 如果找到对应的CA机构，则取出CA里面的信息，将中的（也就是数字签名）做解密，得到信息的hash摘要A。
- 然后将中的信息，做hash得到摘要B，比对摘要A和摘要B是否一致。如果不一致，说明中的信息被修改了。（浏览器则会提醒该证书不是可信任机构颁发的）
- 如果摘要hash一致，则说明证书中的信息未被修改，这时浏览器会比对您现在正在访问的网站与证书中网站信息是否一致，比如域名是否一致、证书是否在有效期内等。
- 另外大部分浏览器也会在线校验证书，是否在有效期内（将证书序列号通过在线证书状态协议“OCSP”发送给CA做校验）。
- 证书校验成功，最后将从证书中取出网站方的，用于后面的握手签名。

4. Server Key Exchange

- 这个步骤是密钥协商的服务端部分，最终的密钥将会用于传输数据对称加密。
- 服务端需要发送一个Diffie-Hellman算法的公钥，和指定使用哪种椭圆曲线多项式。
- 我们到Client Key Exchange的时候，再来讲这个密钥协商过程。
- 这里还有一个签名，校验这个流程的数据是否被篡改。如下图所示，客户端收到Server Key Exchange数据后，可以用上个流程中获得的公钥对签名值解密，获得摘要A。并将这次数据明文做SHA512的hash，获得摘要B，做比对。（这里对协商算法做签名校验）
- 发送的数据如下图所示：





## 5. Server Hello Done

- 服务端发送ServerHelloDone消息表示，已经发送完了密钥协商需要的消息，并且客户端可以开始进行客户端的密钥协商处理了，也就是Client Key Exchange。
- 收到ServerHelloDone后，客户端需要确认服务器是否提供了合法的证书，并且确认服务器的ServerHello消息里面的参数是否可以接受。

## 6. Client Key Exchange

- 这个过程就是告诉服务端，他已经准备好MasterSecret了，可以进行数据加密传输了。
- 这个协议是冗余的，在TLS 1.3里面直接被删除了。

- 这条消息是用来确定双方的MasterSecret是否正确生成，发送的是verify\_data消息。

- `verify_data = PRF(master_secret, finished_label, Hash(handshake_messages))`
  - PRF是伪随机函数 ( pseudorandom function , PRF )
  - `master_secret`是密钥协商时，计算出来的
  - `finished_label`：对客户端发的Finished消息来说，固定是字符串 "client finished". 对服务器发的Finished消息来说，固定是字符串 "server finished".
  - `handshake_messages`，是各端握手过程中发送的所有消息的，类型如下：

[illegible]

- 但不包括ChangeCipherSpec、alerts之类的消息。并且最后一个发送Finished的一方，需要把前一个发送Finished的内容包括进去。
- 注意这里每个端发送自己的握手消息就可以，比如Client发送内容包括ClientHello、Certificate（有发送的话）、CertificateVerify（如果有发送的话）、ClientKeyExchange。

### 8.1. Server New Session Ticket

- New Session Ticket方式与Session ID方式对比：
  - SessionID方式，客户端在ClientHello的时候带着上一次SessionID过来，服务端从自己内存中查找SessionID对应的session状态，并读取session状态快速恢复。
  - SessionTicket方式，则是将session状态加密后，发送给客户端存储。客户端在ClientHello时将SessionTicket带上，服务端就将其解密，读取出面存储的session状态。

```
struct {  
    ProtocolVersion protocol_version; //■■■■■  
    CipherSuite cipher_suite; //■■■■■■■  
    CompressionMethod compression_method; //■■■■■  
    opaque master_secret[48]; //■■■■■■
```

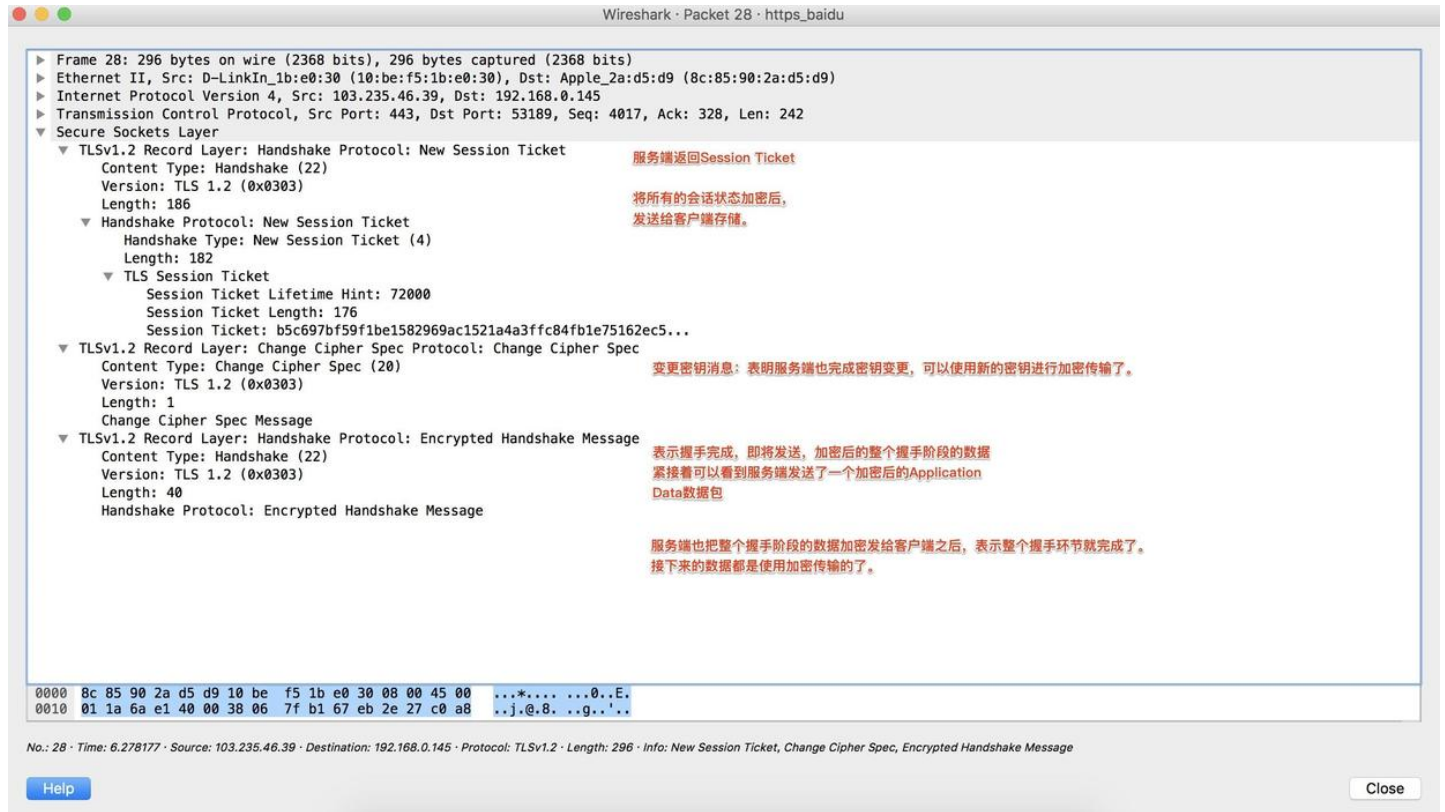
```
ClientIdentity client_identity; //■■■■ID
uint32 timestamp;//ticket■■■■
} StatePlaintext;
```

## 9. Server Change Cipher Spec

- 告诉客户端，我已经准备好进行加密传输了。

## 10. Server Finished

- 与8. Client Finished的情况一样，使用对称密钥加密，最后做一次验证，确定双方是否都准备好进行数据传输了。只是这里加密的数据还不是真正的网站内容数据，而是握手过程的



Wireshark · Packet 28 · https\_baidu

Frame 28: 296 bytes on wire (2368 bits), 296 bytes captured (2368 bits)

Ethernet II, Src: D-LinkIn\_1b:e0:30 (10:be:f5:1b:e0:30), Dst: Apple\_2a:d5:d9 (8c:85:90:2a:d5:d9)

Internet Protocol Version 4, Src: 103.235.46.39, Dst: 192.168.0.145

Transmission Control Protocol, Src Port: 443, Dst Port: 53189, Seq: 4017, Ack: 328, Len: 242

Secure Sockets Layer

- ▼ TLSv1.2 Record Layer: Handshake Protocol: New Session Ticket
  - Content Type: Handshake (22)
  - Version: TLS 1.2 (0x0303)
  - Length: 186
  - Handshake Protocol: New Session Ticket
    - Handshake Type: New Session Ticket (4)
    - Length: 182
    - ▼ TLS Session Ticket
      - Session Ticket Lifetime Hint: 72000
      - Session Ticket Length: 176
      - Session Ticket: b5c697bf59f1be1582969ac1521a4a3ffc84fb1e75162ec5...
- ▼ 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
- ▼ 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

服务端返回Session Ticket

将所有的会话状态加密后，发送给客户端存储。

变更密钥消息：表明服务端也完成密钥变更，可以使用新的密钥进行加密传输了。

表示握手完成，即将发送。加密后的整个握手阶段的数据紧接着可以看到服务端发送了一个加密后的Application Data数据包

服务端也把整个握手阶段的数据加密发给客户端之后，表示整个握手环节就完成了。接下来的数据都是使用加密传输的了。

0000 8c 85 90 2a d5 d9 10 be f5 1b e0 30 08 00 45 00 ...\*... ..0..E.  
0010 01 1a 6a e1 40 00 38 06 7f b1 67 eb 2e 27 c0 a8 ..j.@.8. ..g..'

No.: 28 · Time: 6.278177 · Source: 103.235.46.39 · Destination: 192.168.0.145 · Protocol: TLSv1.2 · Length: 296 · Info: New Session Ticket, Change Cipher Spec, Encrypted Handshake Message

Help Close

## 11. Application Data

- 真正的网站数据传输，但是这里的数据就是经过握手时协商好的对称密钥进行加密的了。
- 现在我们有KeyBlock（对称密钥块），也知道对称加密算法是AES-128-GCM [5.1. AEAD AES 128 GCM](#)

## 参考文献

- [The Transport Layer Security \(TLS\) Protocol Version 1.2](#)
- [OpenSSL 与 SSL 数字证书概念贴](#)
- [详解https是如何确保安全的？](#)
- [数字证书的基础知识](#)
- [TLS 握手优化详解](#)
- [Transport Layer Security \(TLS\) Parameters](#)
- [crypto101](#)
- [TLS协议分析 \(五\) handshake协议 证书与密钥交换](#)
- [HTTPS权威指南：在服务器和Web应用上部署SSL/TLS和PKI](#)
- [读图解HTTP](#)

点击收藏 | 1 关注 | 0

[上一篇：云计算时代哪能没有服务器，阿里云云...](#) [下一篇：【译】要么保证你的JENKINS绝...](#)

1. 3 条回复



[shades](#) 2017-08-29 08:45:31

奶猫妹纸 不容易啊~~



0 回复Ta



[happytree](#) 2017-08-29 08:54:18

欢迎交流、学习

0 回复Ta



[wooyun](#) 2017-08-30 02:00:47

写的很用下，非常不错

0 回复Ta

[登录](#) 后跟帖

先知社区

[现在登录](#)

热门节点

[技术文章](#)

[社区小黑板](#)

目录

[RSS](#) [关于社区](#) [友情链接](#) [社区小黑板](#)