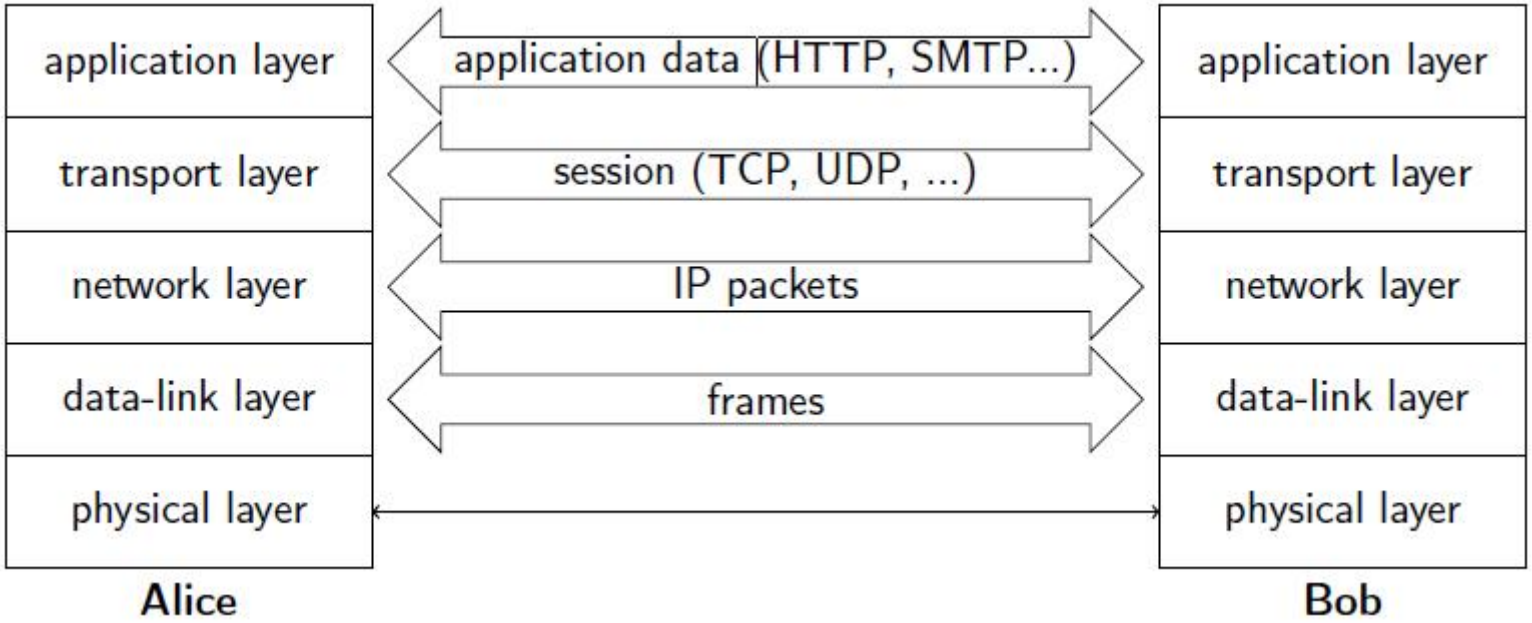
第3章 TLS协议



- application layer security (SSH, S-MIME, PGP, ……)
- transport layer security (TLS/SSL, ……)
- network layer security (IPsec, ……)
- data-link layer security (WEP, WPA, WPA2, ……)

内容提要

- TLS的发展历程
- TLS协议概述
- TLS协议描述语言
- TLS Record协议
- TLS Handshake协议
- TLS ChangeCipherSpec协议
- TLS Alert协议
- TLS Application Data协议
- HTTPS（HTTP Over TLS）
- TLS流量分析，TLS协议攻击

TLS的发展历程

- SSL: Secure Socket Layer
  - Netscape公司设计的主要用于web的安全传输协议
  - SSL1.0→SSL3.0
- TLS: Transportation Layer Security
  - IETF
  - SSL3.0→TLS1.0→TLS1.2→TLS1.3

TLS的发展历程

- 协议发展的主要驱动因素：
  - 商业竞争
    - 微软公司 vs. 网景公司
  - 协议本身不完善
    - 协议出现漏洞

今天TLS已经广泛应用

https://www.baidu.com

https://www.tmall.com

https://www.jd.com

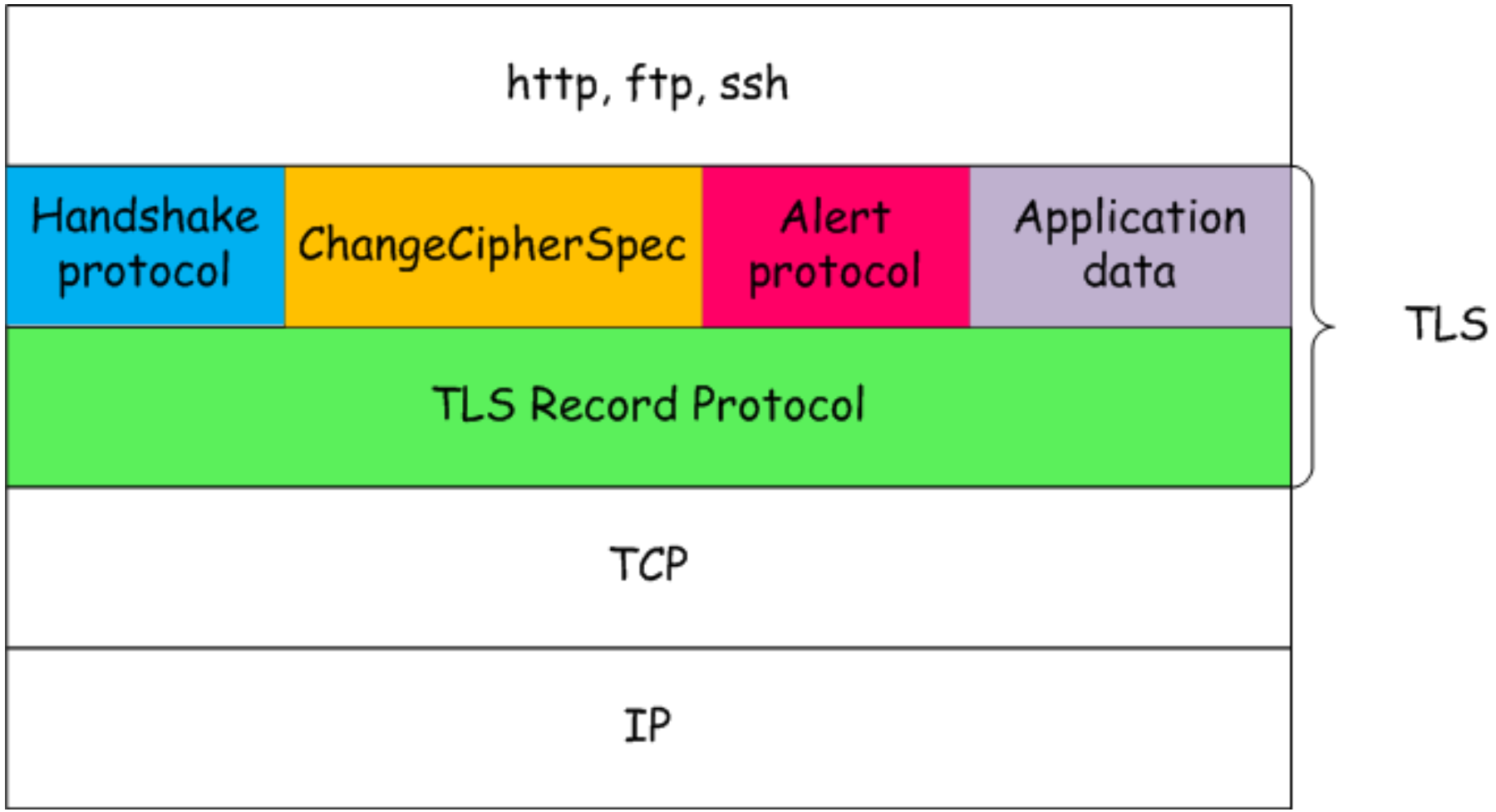
https://www.tencent.com/

TLS已经成为至关重要的网络安全协议

TLS协议概述

The primary goal of the TLS protocol is to provide **privacy** and **data integrity** between two communicating applications

TLS协议分层


http, ftp, ssh
Handshake protocol
ChangeCipherSpec
Alert protocol
Application data
TLS
TLS Record Protocol
TCP
IP

TLS Record Protocol

- 安全服务
  - Confidentiality
  - Integrity
- 基于**握手协议**协商确定的安全参数对应用数据传输提供保密性和完整性保护
  - 分片
  - 压缩
  - 加密
  - MAC

TLS Handshake Protocol

- 通信双方协商确定用于记录层的安全参数
  - 密码套件
  - Premaster secret
  - 身份认证

ChangeCipherSpec Protocol

- 发信号给通信对端，表示要切换到新协商确定的密码规格。

Alert Protocol

- 用于传递协议运行过程中出现的警报，包括警报的严重性（warning or fatal）和对警报的描述。
- fatal级别的警报会导致连接的立即终止

Application Data Protocol

- 给record层提供application data用于传输。

TLS协议描述语言

TLS描述语言

- TLS采用一种专门的语言来描述数据格式，这种语言与C有点**类似**，但是不一样，而且这种语言仅用于TLS，不用于其它地方。
- 基本原则：
  - 基本数据块的大小为一个字节（8bits）
  - 注释采用“/*”开始，“*/”结束。
  - “[[ ]]”表示可选的部件
  - 单字节实体，如果其中包含无具体含义的数据，其类型为opaque

TLS描述语言

- 定长向量：T *T'* [n];
  - T为数据类型，*T'* 是向量名称
  - n是向量中**字节数**
- uint8 foo[4]; /*4个单字节整数*/
- uint16 foo[4]; /*2个双字节整数*/

TLS描述语言

- 变长向量： T T'<floor..ceiling>;
  - T为数据类型，T' 是向量名称
  - floor、ceiling分别表示所占字节数的下界和上界
- uint32 number<4..20>;
  - 变长向量number最少4个字节（1个4字节整数）
  - 变长向量number最多20个字节（5个4字节整数）

TLS描述语言

- 枚举：表示某个变量的可能取值
- enum{e1(v1),e2(v2),…,en(vn)[[,n]]} *Te;*
  - Te变量名，e1到en是Te可能的取值；
  - v1到vn则是为e1到en指定一个具体指代数值；
  - n表示可取值的个数。
- enum {warning(1),fatal(2), (255)}AlertLevel;
  - AlertLevel这个变量可以取值为warning或者fatal，最多可有255个取值。

TLS描述语言

- 结构：某个变量由不同类型数据组成

```
struct {
    T1 f1;
    T2 f2;
    …
    Tn fn;
} [[T]];
```

- 其中，T1,…,Tn表示结构中成员的数据类型，T是变量的名字

TLS描述语言

- 变体：根据实际选择符不同，成员可以选择不同的数据类型。

```
struct {
        T1 f1;
        T2 f2;
        …
        Tn fn;
        select(E){
                case e1: Te1;
                case e2: Te2;
                …
                case en: Ten;
        }[[fv]];
} [[Tv]];
```

根据E的值来确定fv选择哪个类型

TLS Record protocol

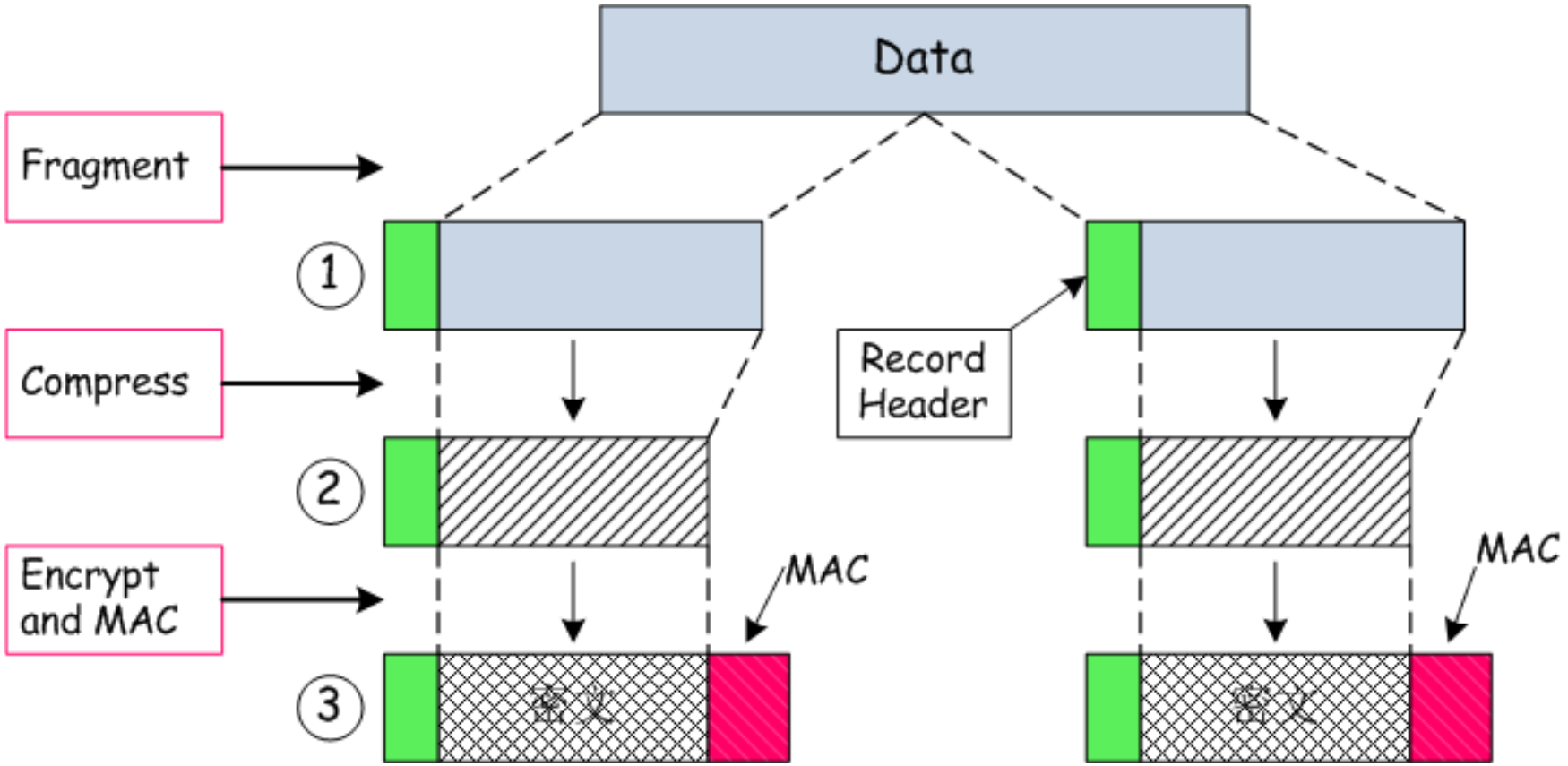
记录协议：功能目标

- 消息传输
  - 记录协议传输由上层协议提交给它的数据缓冲区，如果缓冲区超过长度限制（2^14），则需分片。属于**同一协议**的小缓冲区也可组合成单个记录。
- 加密及完整性验证
  - 按照握手协议协商的安全参数进行加密和完整性验证（少量ciphersuite不加密仅完整性验证）。

记录协议：功能目标

- 压缩
  - 设计上，记录协议在加密之前需要对数据进行压缩，以提高传输效率。实践中，基本都没有实现压缩功能。
- 扩展性
  - 记录协议只关注数据传输和加密，其余功能由其他子协议完成，这种设计使得TLS协议具有很好的扩展性，可以根据需要添加子协议，所有子协议都会受到协商出的安全参数保护。

TLS Record协议：封装过程



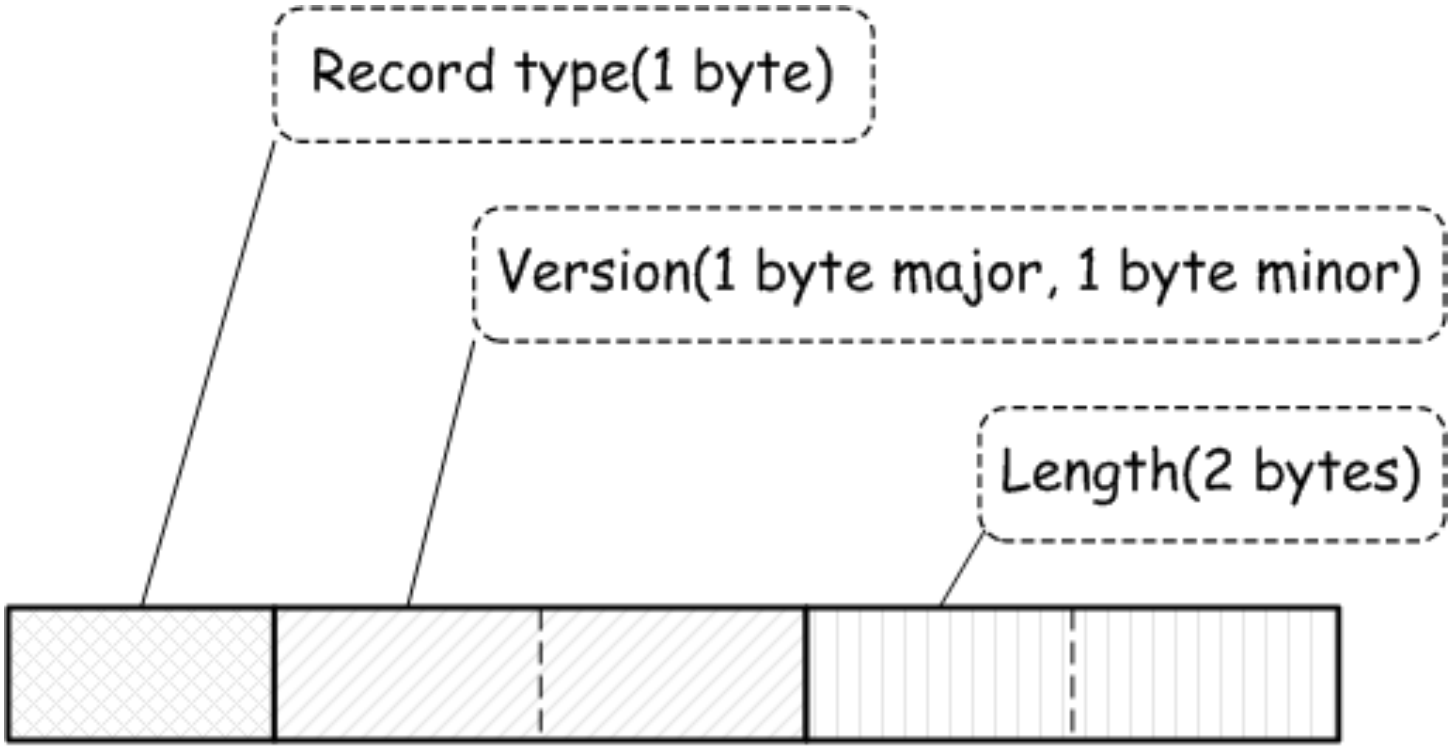
Record：①明文分片

```
struct {
    ContentType type;
    ProtocolVersion version;
    uint16 length;
opaque fragment[TLSPlaintext.length];
} TLSPlaintext;
```

Record Header

明文数据分片

TLS Record Header



TLS Record header

Record Header（续）

- 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

Record Header：记录类型

- 记录类型用于说明后面所承载内容的协议类型

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

Record header：record type表示

```
enum {
    change_cipher_spec(20),
    alert(21),
    handshake(22),
    application_data(23),
    (255)
} ContentType;
```

Record Header：版本号

- 用于说明当前所协商采用的TLS的版本号，包括主版本号和次版本号

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

Record header: Version表示

```
struct {
        uint8 major;
        uint8 minor;
} ProtocolVersion;
```

Record Header：长度

- 长度用于说明记录中数据的长度（不包括记录头本身），最大支持的长度是2^14字节

Record：①明文分片

```
struct {
    ContentType type;
    ProtocolVersion version;
    uint16 length;
opaque fragment[TLSPlaintext.length];
} TLSPlaintext;
```

Record: ②压缩

```
struct {
    ContentType type;
    ProtocolVersion version
    uint16 length;
opaque fragment[TLSCompressed.length];
      } TLSCompressed
```

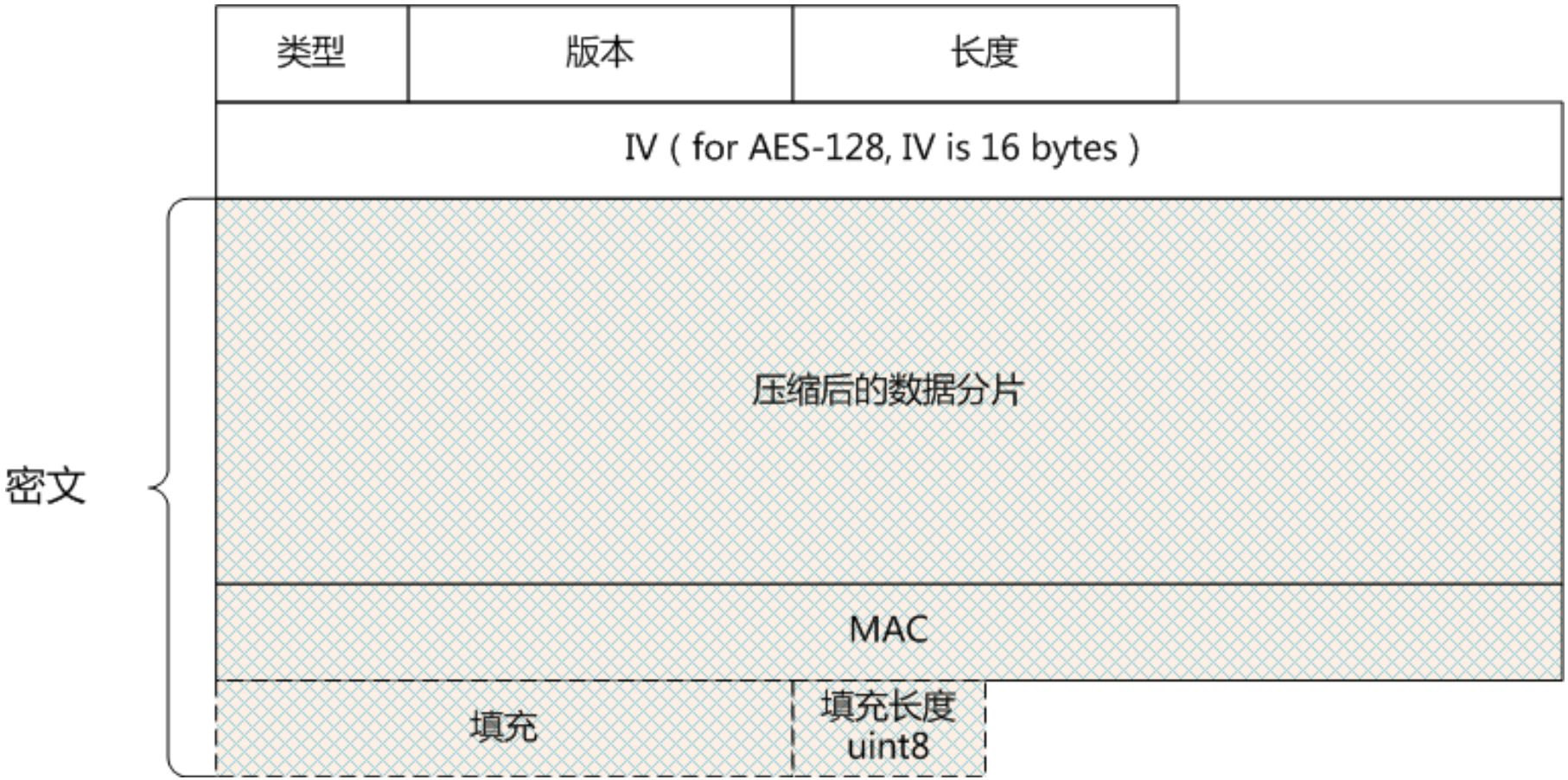
Record：③安全处理

```
struct {
      ContentType type;
      ProtocolVersion version;
      uint16 length;
      select
(SecurityParameters.cipher_type) {
          case stream: GenericStreamCipher;
          case block:  GenericBlockCipher;
          case aead:   GenericAEADCipher;
      } fragment;
  } TLSCiphertext;
```

Record：③安全处理--GenericBlockCipher

```
struct {
      opaque IV[SecurityParameters.record_iv_length];
      block-ciphered struct {
          opaque content[TLSCompressed.length];
          opaque MAC[SecurityParameters.mac_length];
          uint8 padding[GenericBlockCipher.padding_length];
          uint8 padding_length;
        };
    } GenericBlockCipher;
```

Record封装结果：block cipher



拓展阅读

- Q. Sun, D. R. Simon, Y.-M. Wang, et al., Statistical Identification of Encrypted Web Browsing Traffic, in Proceedings of the 2002 IEEE Symposium on Security and Privacy: IEEE Computer Society, 2002, p. 19.
- Abstract:
  - Encryption is often proposed as a tool for protecting the privacy of World Wide Web browsing. However, encryption particularly implemented in, or in concert with popular Web browsers--does not hide all information about the encrypted plaintext. Specifically, *HTTP object count and sizes are often revealed* (or at least incompletely concealed). We investigate the identifiability of World Wide Web traffic based on this unconcealed information in a large sample of Web pages, and show that it suffices to identify a significant fraction of them quite reliably. We also suggest some possible countermeasures against the exposure of this kind of information and experimentally evaluate their effectiveness.

拓展阅读

- Maciá-Fernández, Y. Wang, R. Rodríguez-Gómez, et al., ISP-enabled behavioral ad targeting without deep packet inspection, in Proceedings of the 29th conference on Information communications San Diego, California, USA: IEEE Press, 2010, pp. 1469-1477.

Abstract:

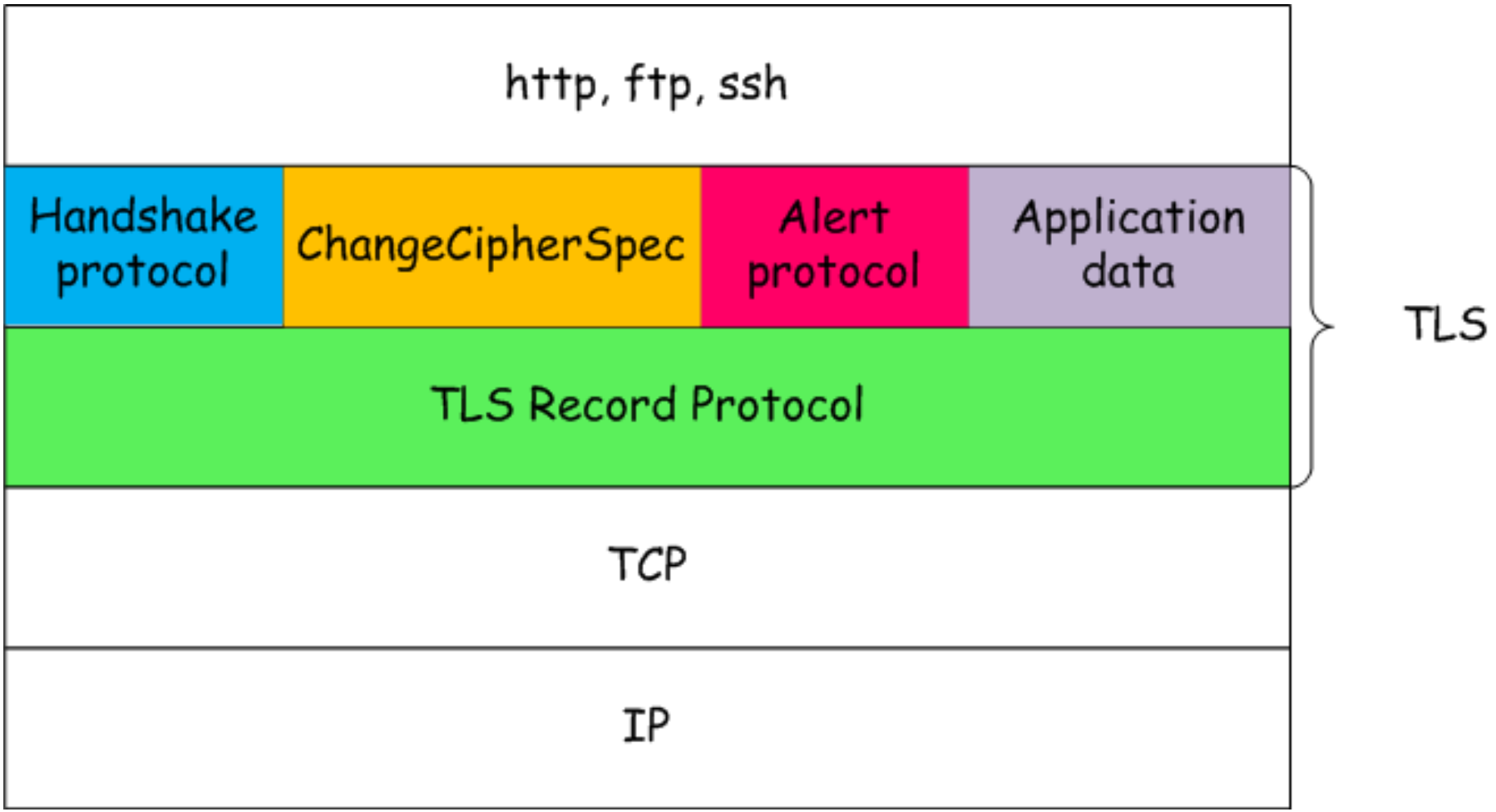
- Online advertising is a rapidly growing industry currently dominated by the search engine 'giant' Google. In an attempt to tap into this huge market, Internet Service Providers (ISPs) started deploying deep packet inspection techniques to track and collect user browsing behavior. However, such techniques violate wiretap laws that explicitly prevent intercepting the contents of communication without gaining consent from consumers. In this paper, we show that it is possible for ISPs to extract user browsing patterns without inspecting contents of communication. Our contributions are threefold. First, we develop a methodology and implement a system that is capable of extracting web browsing features from stored non-content based records of online communication, which could be legally shared. When such browsing features are correlated with information collected by independently crawling the Web, it becomes possible to recover the actual web pages accessed by clients. Second, we systematically evaluate our system on the Internet and demonstrate that it can successfully recover user browsing patterns with high accuracy. Finally, our findings call for a comprehensive legislative reform that would not only enable fair competition in the online advertising business, but more importantly, protect the consumer rights in a more effective way.

拓展阅读

- Andrew Reed, Michael Kranch. Identifying HTTPS-Protected Netflix Videos in Real-Time. CODASPY 2017.
- Shuo Chen, Rui Wang, XiaoFeng Wang, Kehuan Zhang. Side-Channel Leaks in Web Applications: a Reality Today, a Challenge Tomorrow. 31st IEEE Symposium on Security and Privacy ("Oakland") 2010.

Handshake protocol

TLS协议分层


http, ftp, ssh
Handshake protocol
ChangeCipherSpec
Alert protocol
Application data
TLS Record Protocol
TLS
TCP
IP

TLS HandShake协议

- 握手协议，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。

① 算法协商

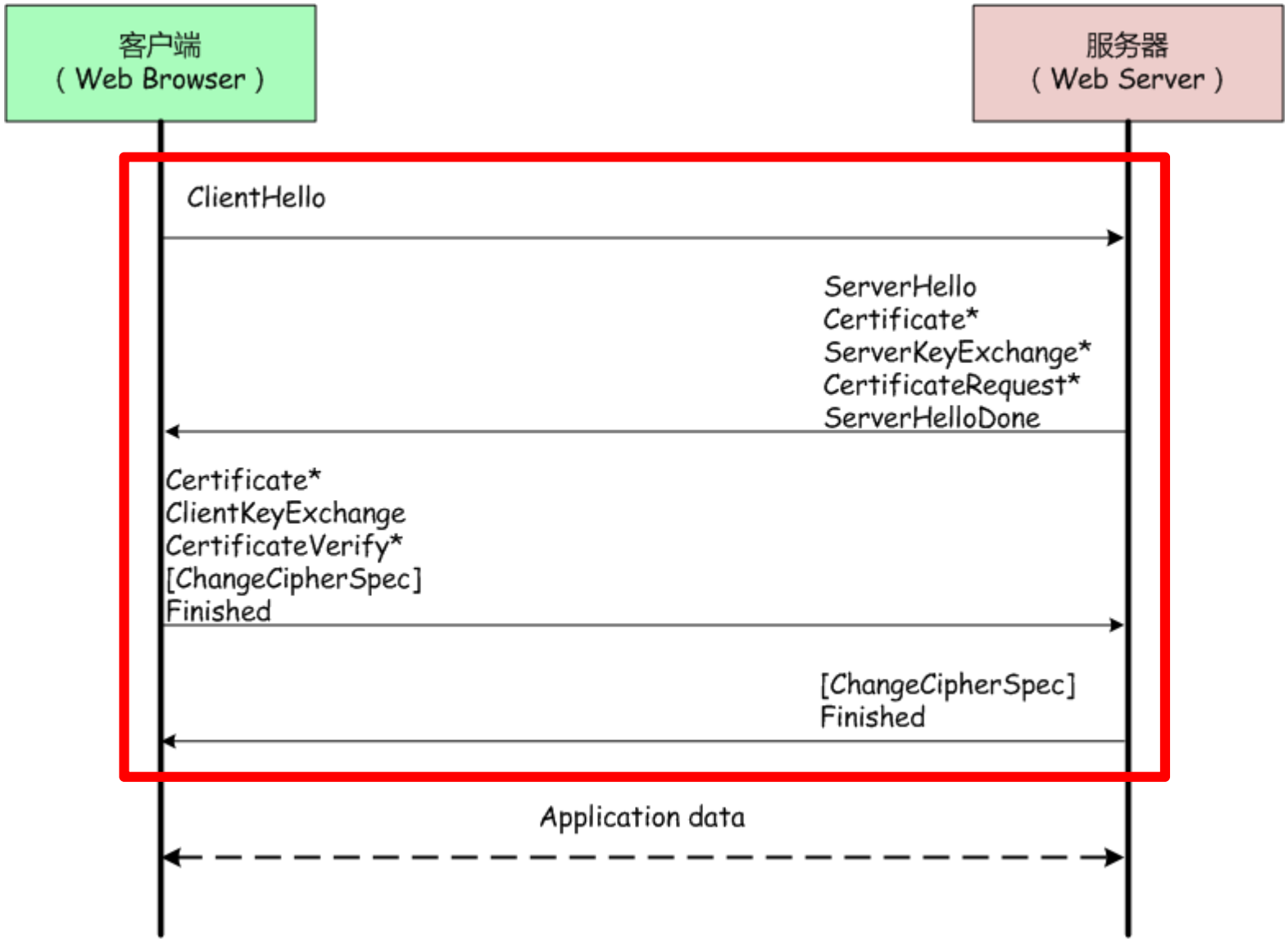
② 密钥协商

③ 身份认证

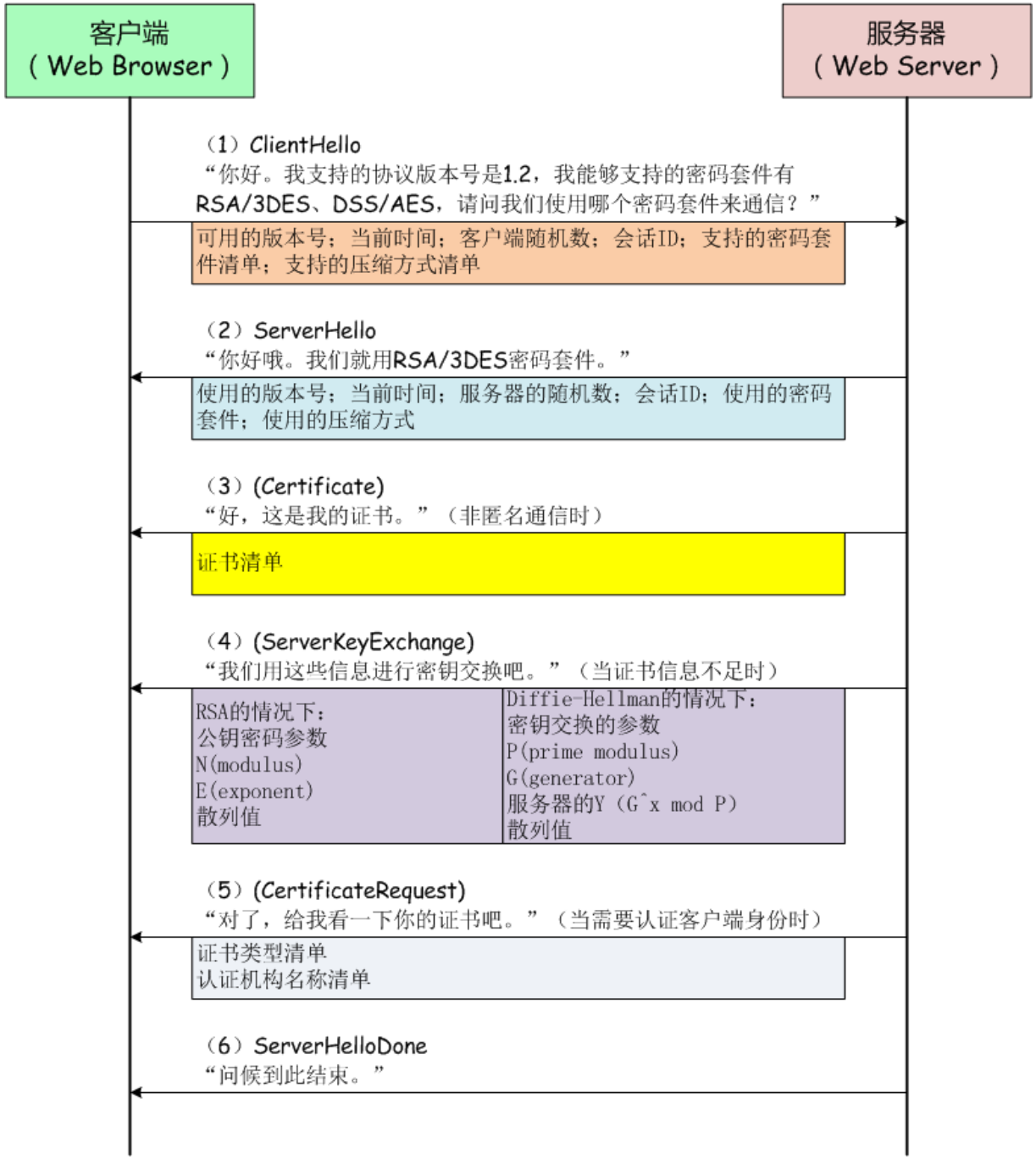
Handshake协议：对话过程

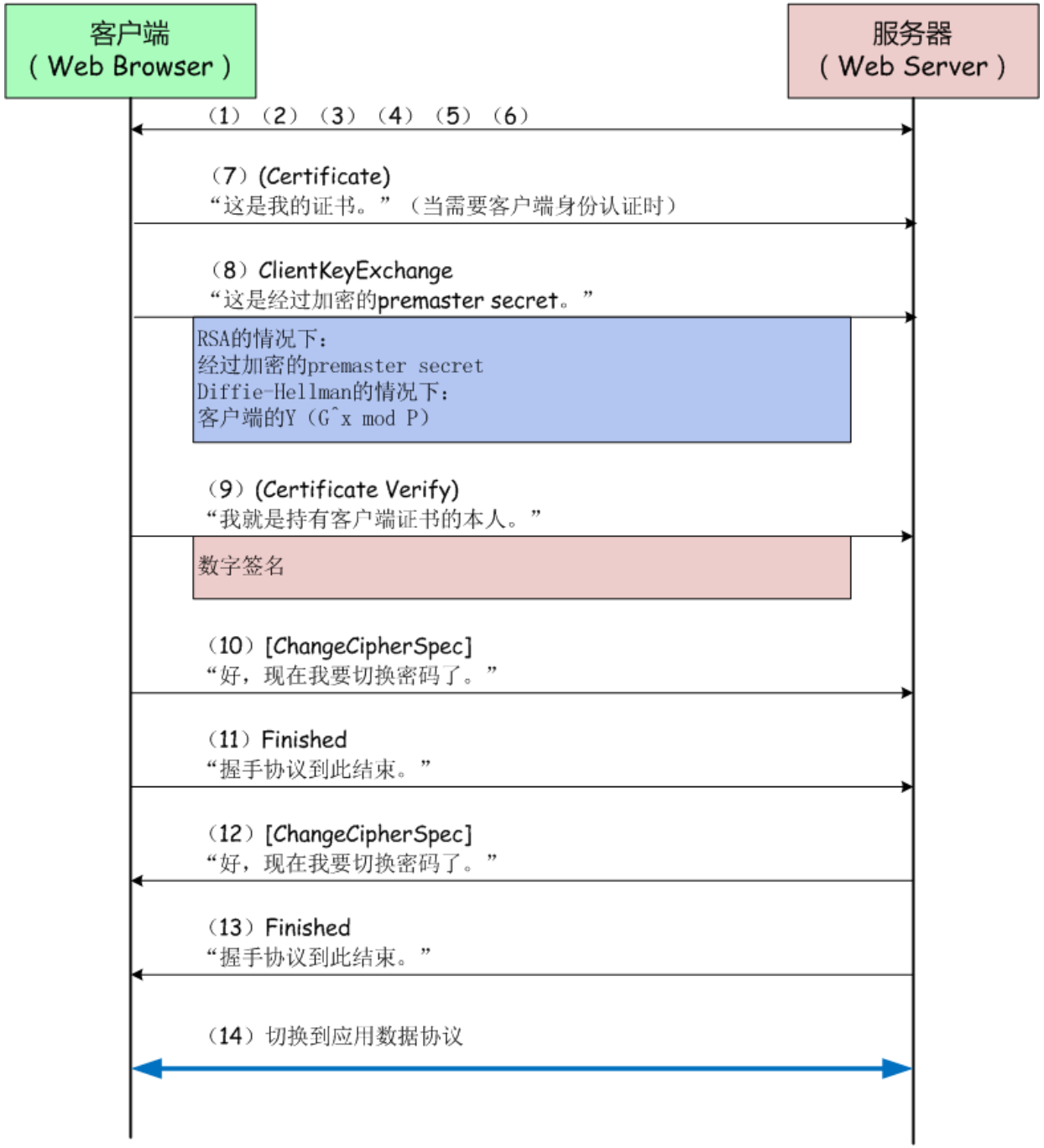
- 客户端：“你好。我能够支持的**密码套件**有RSA/3DES或者DSS/AES，我们使用哪个密码套件来通信呢？”
- 服务器端：“你好。我们就用RSA/3DES来进行通信吧，这是我的证书。”
- 服务器端和客户端通过握手协议协商一致后，就会通过Change Cipher Spec Protocol来发出信号，切换到约定的密码规格。

Message flow for a full handshake

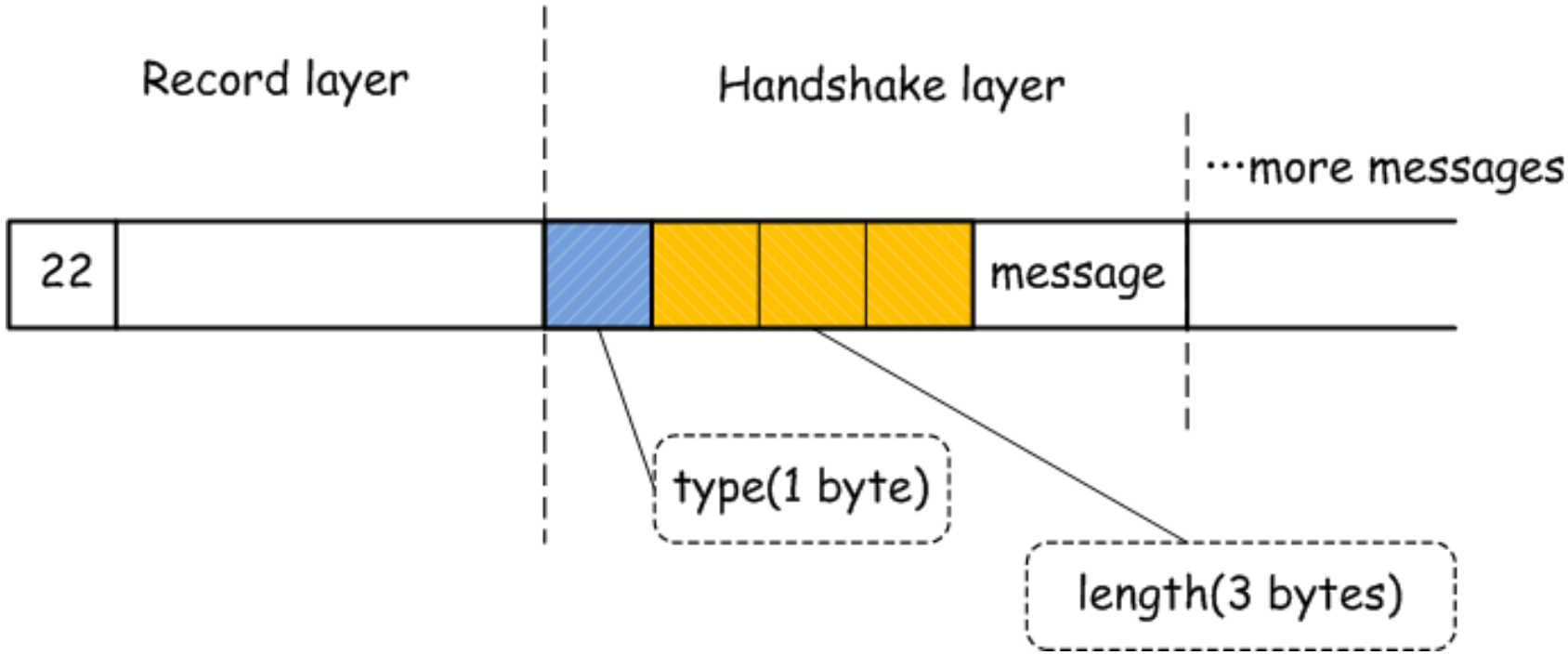



详细版本
Handshake
完整流程


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。”


客户端
(Web Browser)
服务器
(Web Server)
（1）（2）（3）（4）（5）（6）
（7）(Certificate)
“这是我的证书。”（当需要客户端身份认证时）
（8）ClientKeyExchange
“这是经过加密的premaster secret。”
RSA的情况下：
经过加密的premaster secret
Diffie-Hellman的情况下：
客户端的Y（G^x mod P）
（9）(Certificate Verify)
“我就是持有客户端证书的本人。”
数字签名
（10）[ChangeCipherSpec]
“好，现在我要切换密码了。”
（11）Finished
“握手协议到此结束。”
（12）[ChangeCipherSpec]
“好，现在我要切换密码了。”
（13）Finished
“握手协议到此结束。”
（14）切换到应用数据协议

Handshake包图示



Handshake消息格式

```
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;
```

Handshake消息格式：HandShakeType

```
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;
```

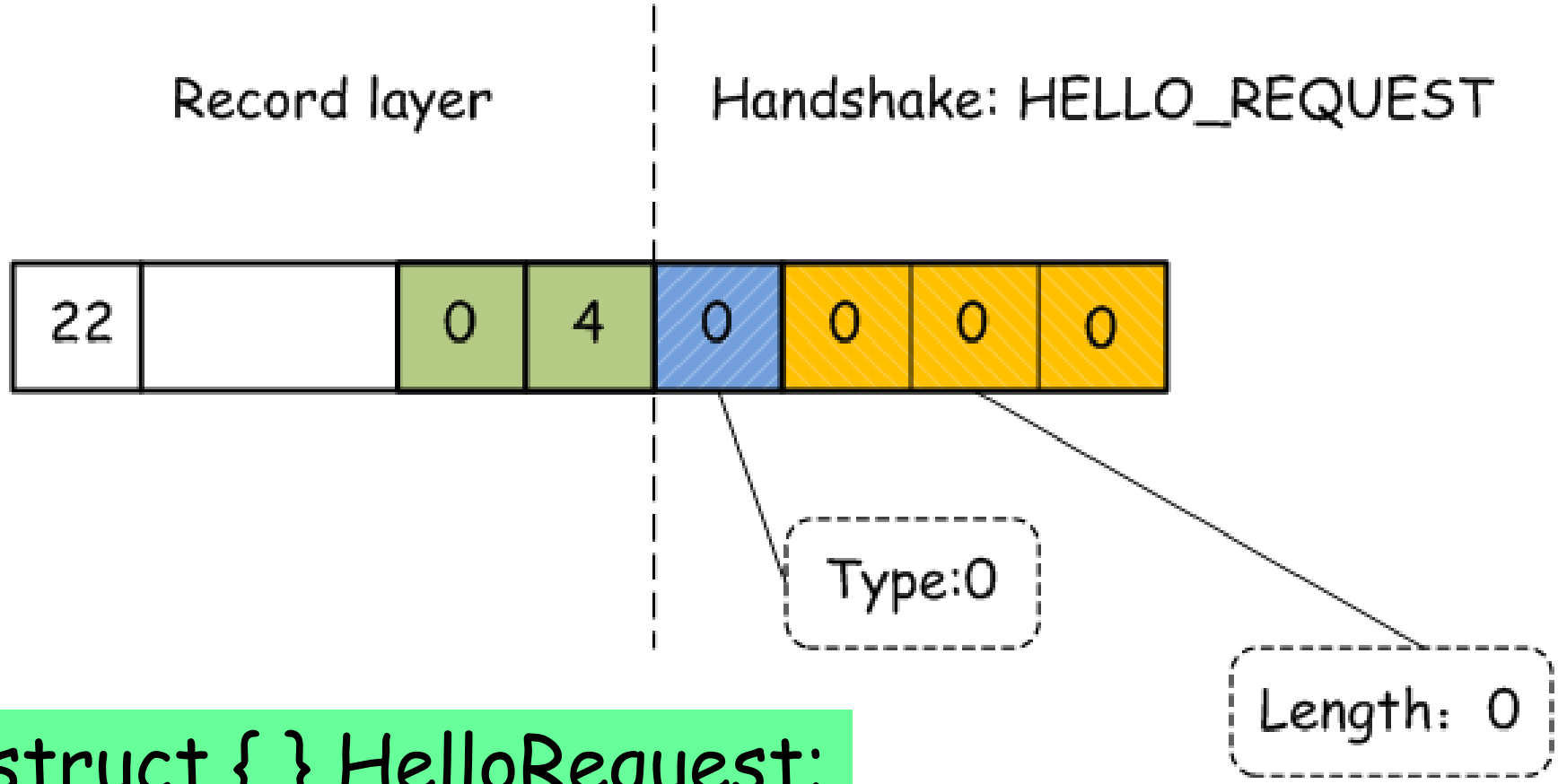
HandshakeType

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_DONE	14	0X0E
CERTIFICATE_VERIFY	15	0X0F
CLIENT_KEY_EXCHANGE	16	0X10
FINISHED	20	0X14

Handshake: HelloRequest

HelloRequest是一个简单的通知，告诉client应该重新开始一个协商过程。作为响应，client应该在合适的时候发送ClientHello消息。如果client当前正在协商一个会话，则该消息会被忽略。

HelloRequest消息图示




```
struct { } HelloRequest;
```

没有消息体

HandShake: ClientHello

ClientHello

- 在一次新的握手流程中，ClientHello消息总是第一条消息。ClientHello消息将客户端支持的功能和首选项发送给服务器。
- When: 发送ClientHello消息
  - 在新建连接时
  - 重新协商时
  - 响应服务器发起的重新协商请求（HelloRequest）时

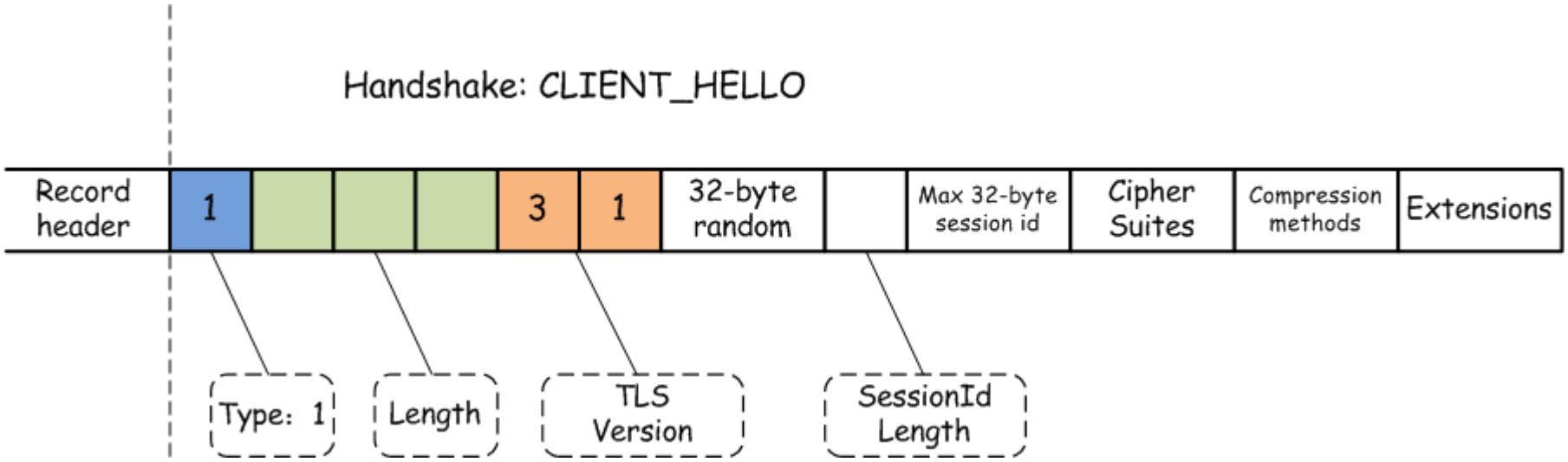
ClientHello消息格式

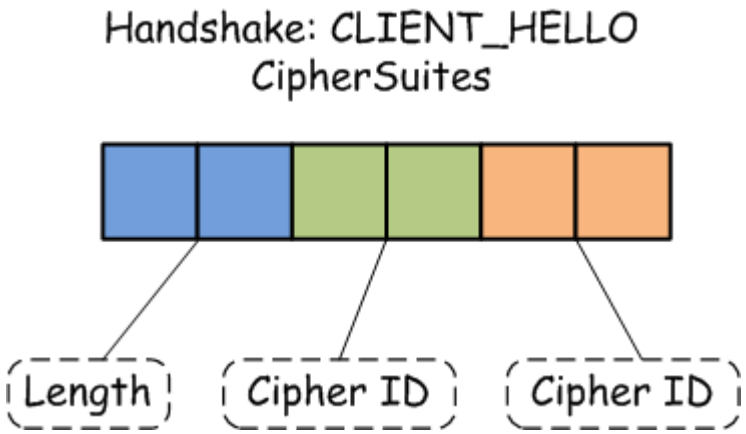
```
struct {
    ProtocolVersion client_version;
    Random random;
    SessionID session_id;
    CipherSuite cipher_suites<2..2^16-2>;
    CompressionMethod compression_methods<1..2^8-1>;
    select (extensions_present) {
        case false:
            struct {};
        case true:
            Extension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

ClientHello: 成员类型

```
struct {
        uint32 gmt_unix_time;
        opaque random_bytes[28];
    } Random;

opaque SessionID<0..32>;

uint8 CipherSuite[2];

enum {null(0), (255)} CompressionMethod;
```

ClientHello消息图示





CipherSuites：客户端支持的密码学选项列表，客户端最倾向于使用的选项放在第一位。如果SessionID不为空的话（表示要重用一个已经存在的会话），则CipherSuites必须至少包含要重用的这个会话所使用的密码套件。

ClientHello: Extension

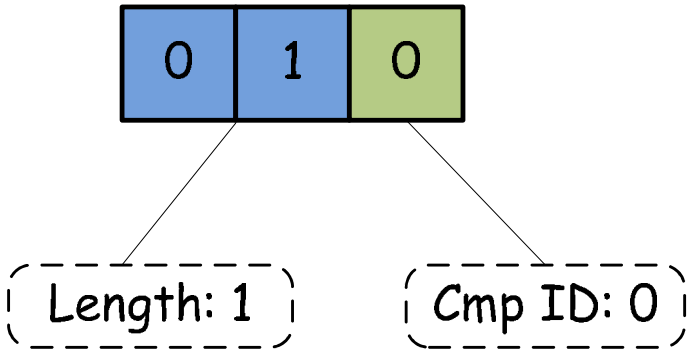
```
struct {
    ExtensionType extension_type;
    opaque extension_data<0..2^16-1>;
    } Extension;
enum {
    signature_algorithms(13), (65535)
    } ExtensionType;
```

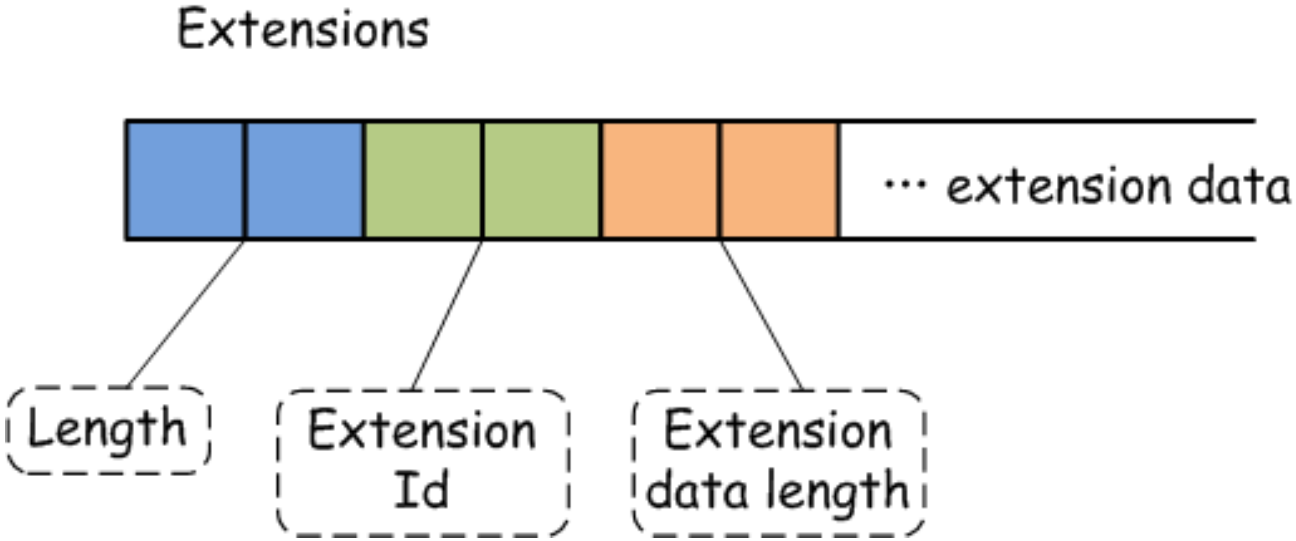
ClientHello：extension

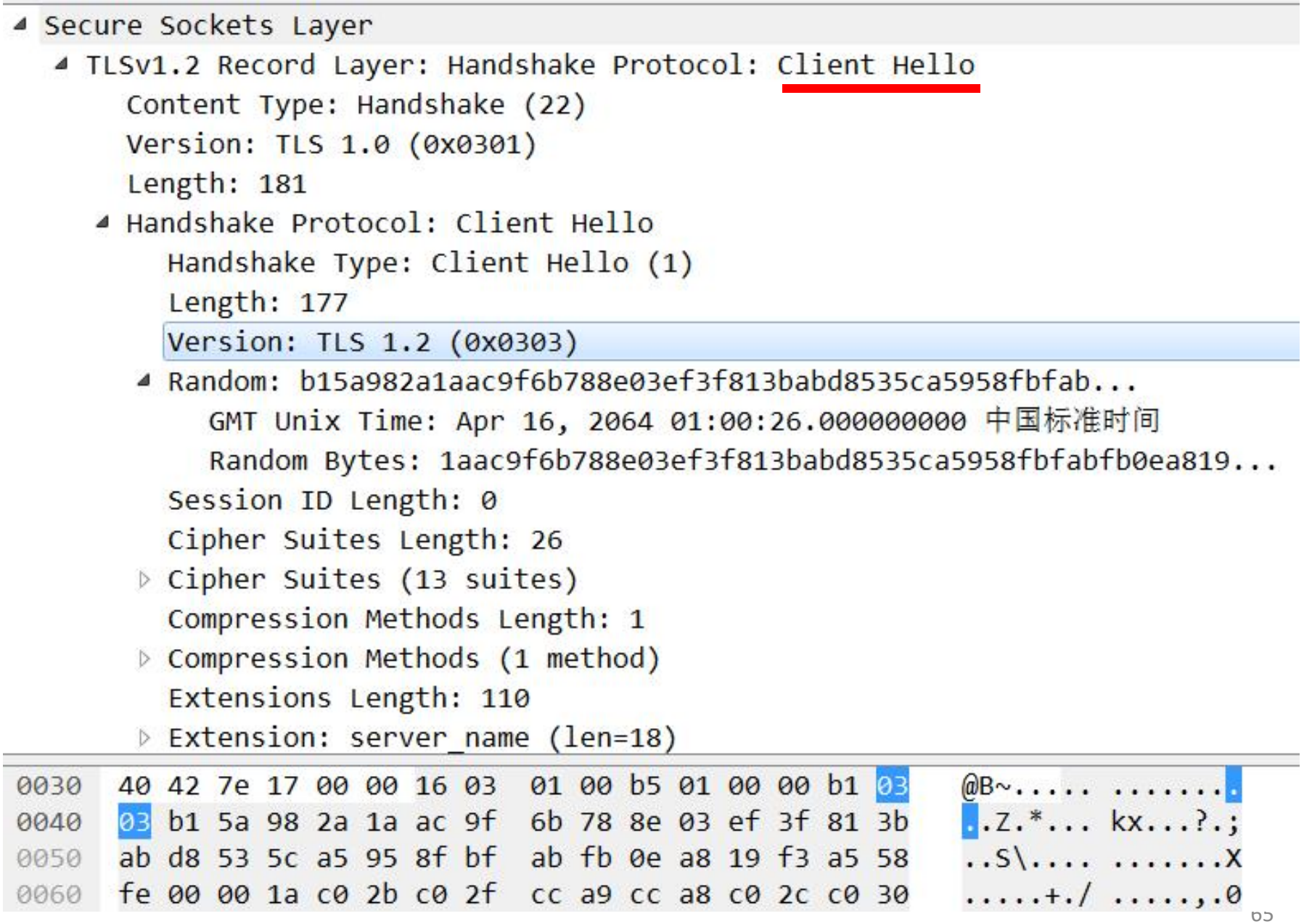
- Extension是TLS的一种机制，可以在不修改协议本身的条件下为TLS协议增加功能。
- 扩展以扩展块的形式出现ClientHello和ServerHello消息的末尾
- 扩展块由所需数量的扩展一个个堆叠而成
- 扩展的格式和期望的行为由每个扩展自己决定，通常用于支持某些新功能、用于在握手阶段传递所需的额外参数。

ClientHello消息图示

Compression methods (no practical implementation uses compression)






Secure Sockets Layer
TLSv1.2 Record Layer: Handshake Protocol: Client Hello
Content Type: Handshake (22)
Version: TLS 1.0 (0x0301)
Length: 181
Handshake Protocol: Client Hello
Handshake Type: Client Hello (1)
Length: 177
Version: TLS 1.2 (0x0303)
Random: b15a982a1aac9f6b788e03ef3f813babd8535ca5958fbfab...
GMT Unix Time: Apr 16, 2064 01:00:26.000000000 中国标准时间
Random Bytes: 1aac9f6b788e03ef3f813babd8535ca5958fbfabfb0ea819...
Session ID Length: 0
Cipher Suites Length: 26
Cipher Suites (13 suites)
Compression Methods Length: 1
Compression Methods (1 method)
Extensions Length: 110
Extension: server_name (len=18)
0030 40 42 7e 17 00 00 16 03 01 00 b5 01 00 00 b1 03 @B~..... ....
0040 03 b1 5a 98 2a 1a ac 9f 6b 78 8e 03 ef 3f 81 3b ..Z.*... kx...?.;
0050 ab d8 53 5c a5 95 8f bf ab fb 0e a8 19 f3 a5 58 ..S\.... X
0060 fe 00 00 1a c0 2b c0 2f cc a9 cc a8 c0 2c c0 30 .....+./ .....,.0

HandShake: ServerHello

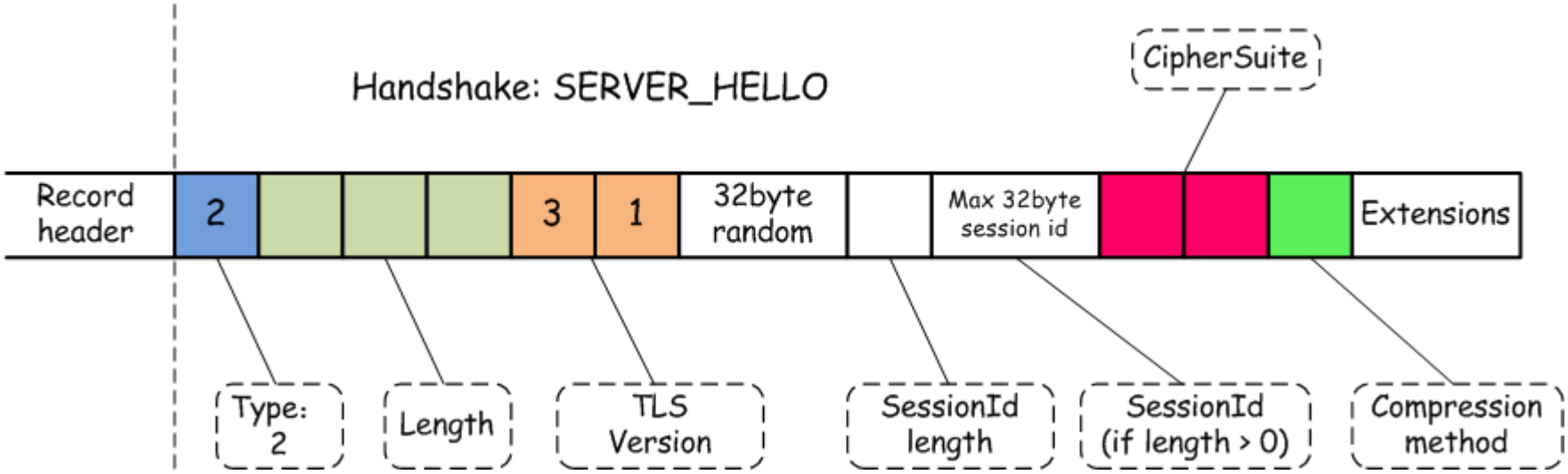
When:

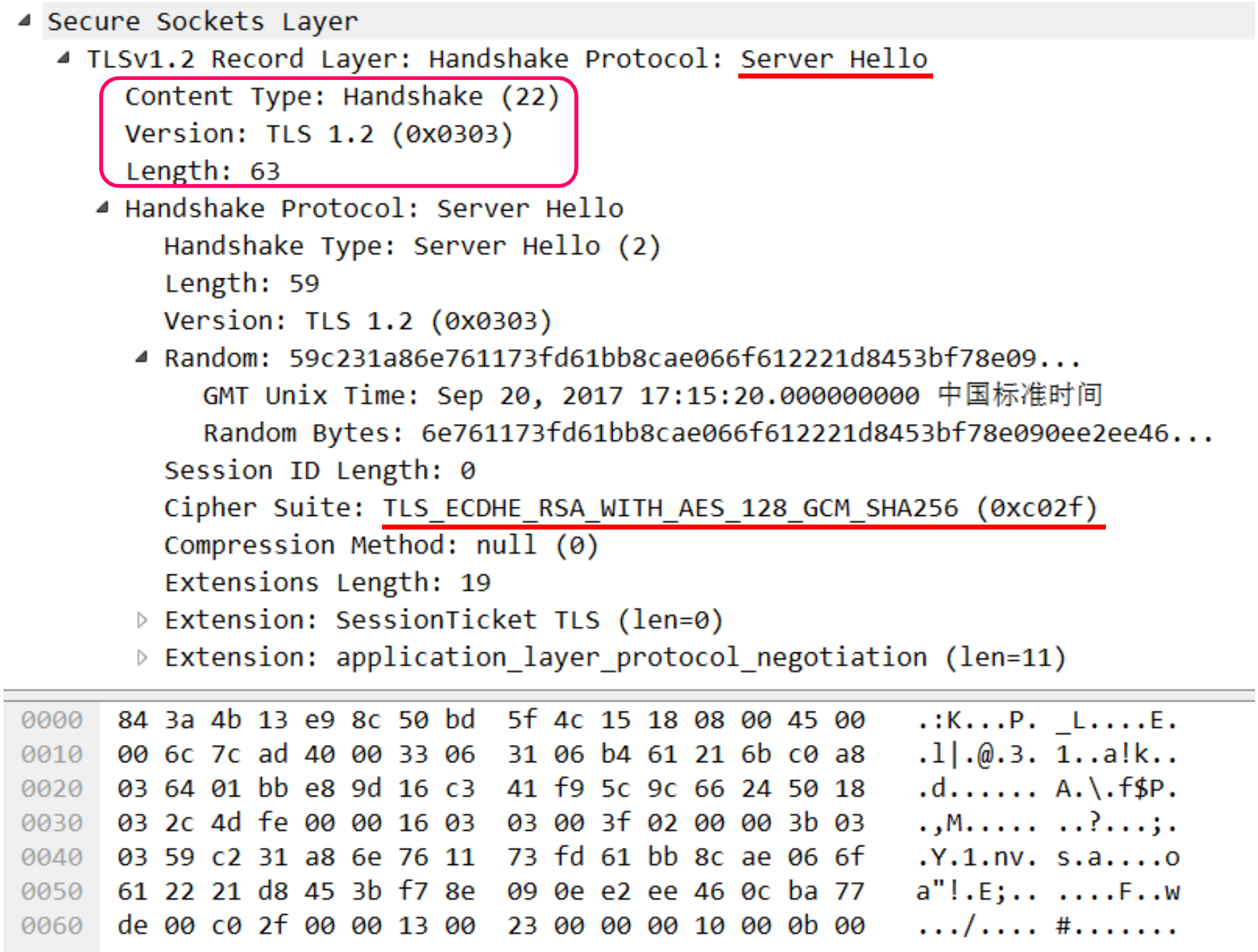
当服务器收到来自客户端的ClientHello消息后，如果它能够找到一套可以接受的算法（即可以就加密算法等取得协商一致），服务器将发送ServerHello消息来响应客户端的ClientHello消息。如果不能找到一套匹配的算法，则服务器将响应handshake failure alert。

HandShake: ServerHello

```
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;
```

ServerHello消息图示




Secure Sockets Layer
TLSv1.2 Record Layer: Handshake Protocol: Server Hello
Content Type: Handshake (22)
Version: TLS 1.2 (0x0303)
Length: 63
Handshake Protocol: Server Hello
Handshake Type: Server Hello (2)
Length: 59
Version: TLS 1.2 (0x0303)
Random: 59c231a86e761173fd61bb8cae066f612221d8453bf78e09...
GMT Unix Time: Sep 20, 2017 17:15:20.000000000 中国标准时间
Random Bytes: 6e761173fd61bb8cae066f612221d8453bf78e090ee2ee46...
Session ID Length: 0
Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f)
Compression Method: null (0)
Extensions Length: 19
Extension: SessionTicket TLS (len=0)
Extension: application_layer_protocol_negotiation (len=11)
0000 84 3a 4b 13 e9 8c 50 bd 5f 4c 15 18 08 00 45 00 .:K...P. _L....E.
0010 00 6c 7c ad 40 00 33 06 31 06 b4 61 21 6b c0 a8 .l|.@.3. 1..a!k..
0020 03 64 01 bb e8 9d 16 c3 41 f9 5c 9c 66 24 50 18 .d...... A.\.f$P.
0030 03 2c 4d fe 00 00 16 03 03 00 3f 02 00 00 3b 03 .,M..... ..?...;.
0040 03 59 c2 31 a8 6e 76 11 73 fd 61 bb 8c ae 06 6f .Y.1.nv. s.a....o
0050 61 22 21 d8 45 3b f7 8e 09 0e e2 ee 46 0c ba 77 a"!.E;.. ....F..w
0060 de 00 c0 2f 00 00 13 00 23 00 00 00 10 00 0b 00 .../.... #.......

Handshake: Certificate

服务器向客户端发送Certificate消息，使得客户端能够认证服务器的身份。

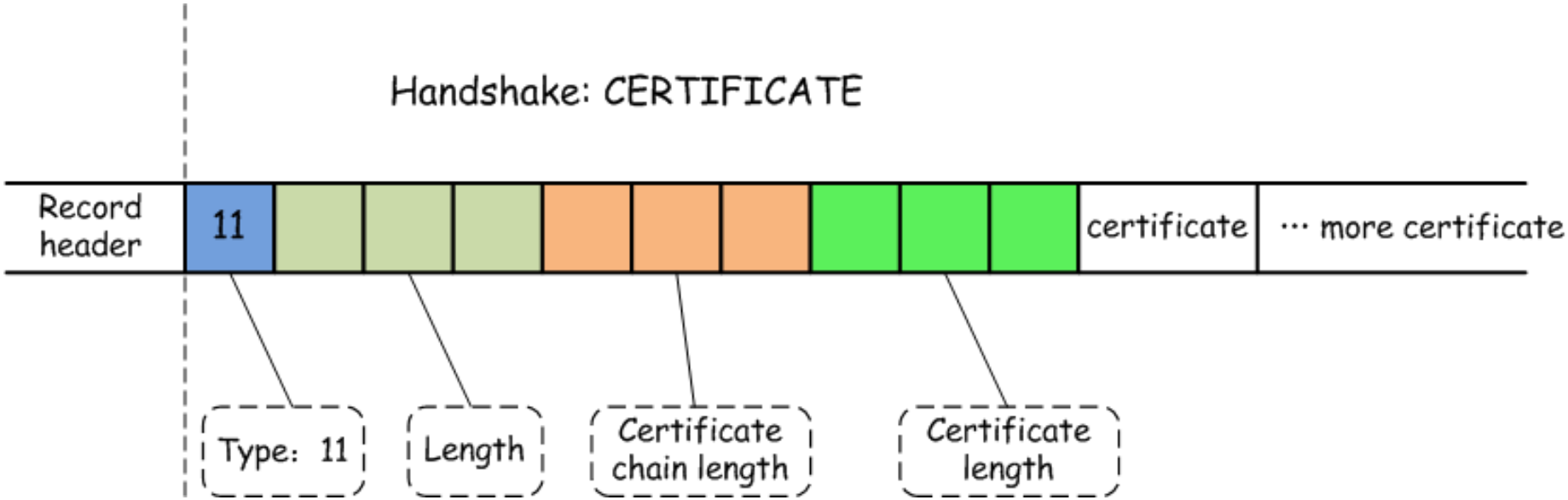
匿名通信的情况下，服务器不需要发送certificate消息。

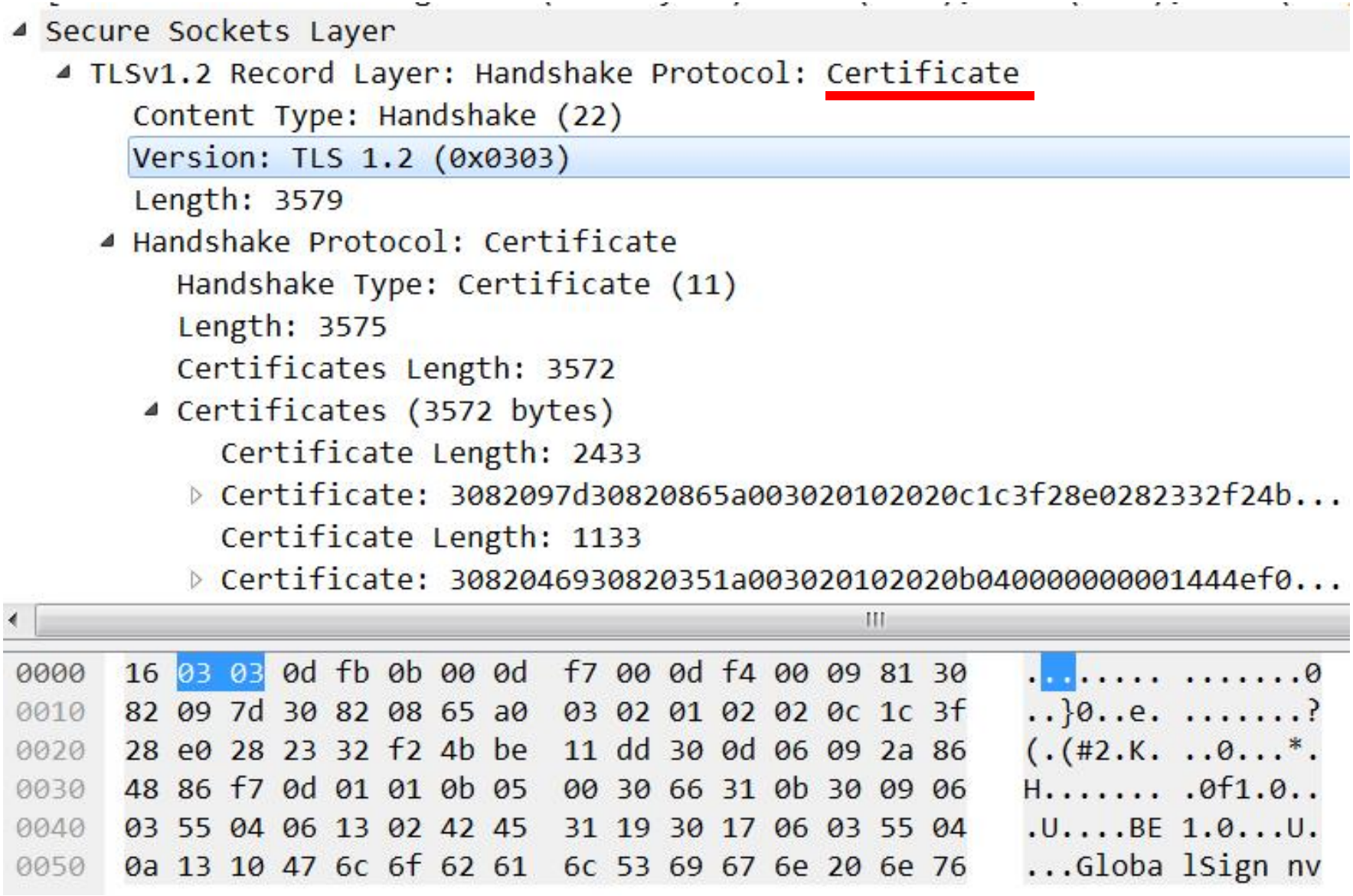
Certificate格式描述

```
opaque ASN.1Cert<1..2^24-1>;

struct {
        ASN.1Cert certificate_list<0..2^24-1>;
  } Certificate;
```

注:
ASN.1，即抽象语法表示法1（Abstract Syntax Notation one，ASN.1），是支持复杂数据结构和对象的定义、传输、交换的一系列规则。

Certificate消息图示

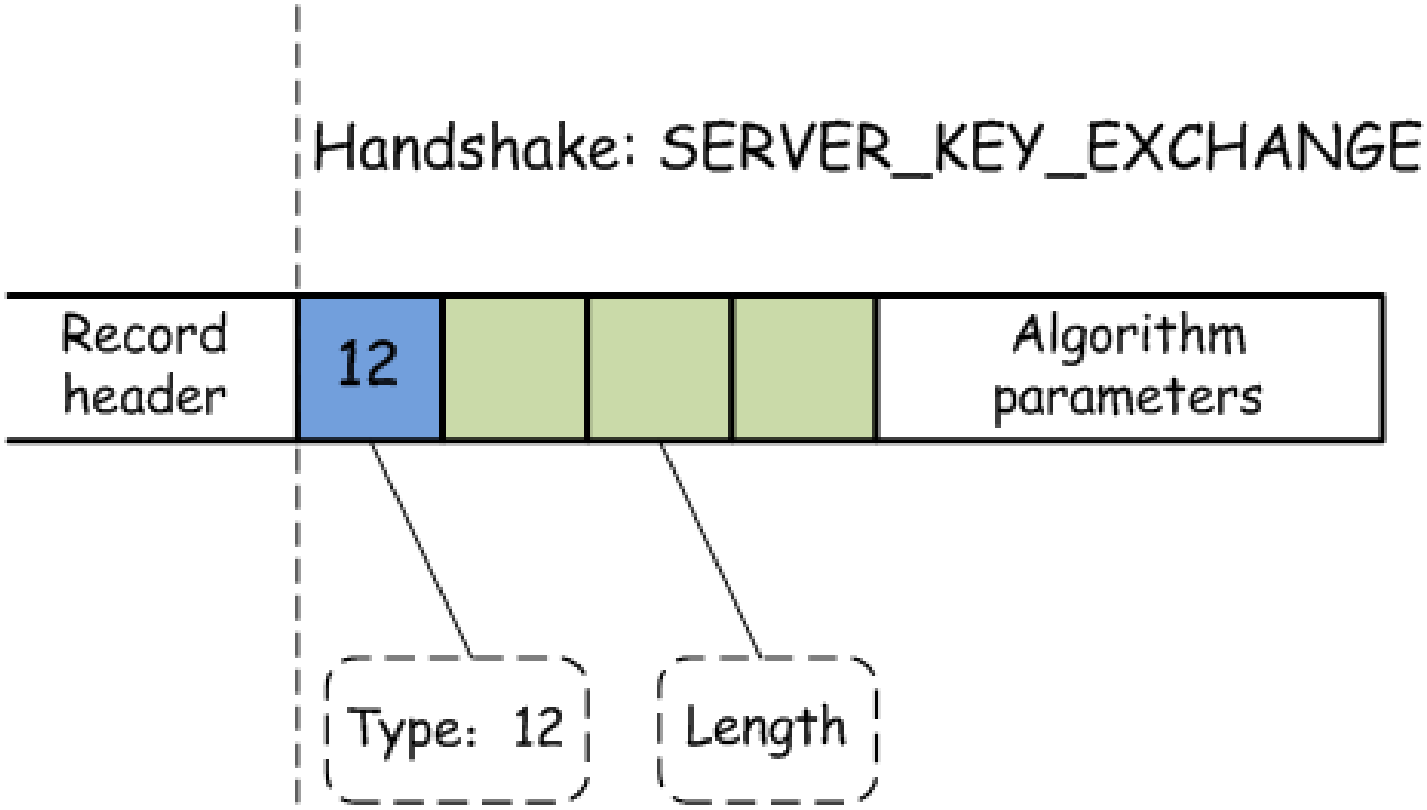



Secure Sockets Layer
TLSv1.2 Record Layer: Handshake Protocol: Certificate
Content Type: Handshake (22)
Version: TLS 1.2 (0x0303)
Length: 3579
Handshake Protocol: Certificate
Handshake Type: Certificate (11)
Length: 3575
Certificates Length: 3572
Certificates (3572 bytes)
Certificate Length: 2433
Certificate: 3082097d30820865a003020102020c1c3f28e0282332f24b...
Certificate Length: 1133
Certificate: 308204693082035 1a003020102020b0400000000001444ef0...
0000 16 03 03 0d fb 0b 00 0d f7 00 0d f4 00 09 81 30 0
0010 82 09 7d 30 82 08 65 a0 03 02 01 02 02 0c 1c 3f ..}0..e. .....?
0020 28 e0 28 23 32 f2 4b be 11 dd 30 0d 06 09 2a 86 (.(#2.K. ..0...*.
0030 48 86 f7 0d 01 01 0b 05 00 30 66 31 0b 30 09 06 H..... .0f1.0..
0040 03 55 04 06 13 02 42 45 31 19 30 17 06 03 55 04 .U....BE 1.0...U.
0050 0a 13 10 47 6c 6f 62 61 6c 53 69 67 6e 20 6e 76 ...Globa lSign nv

Handshake:ServerKeyExchange

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。

ServerKeyExchange消息图示



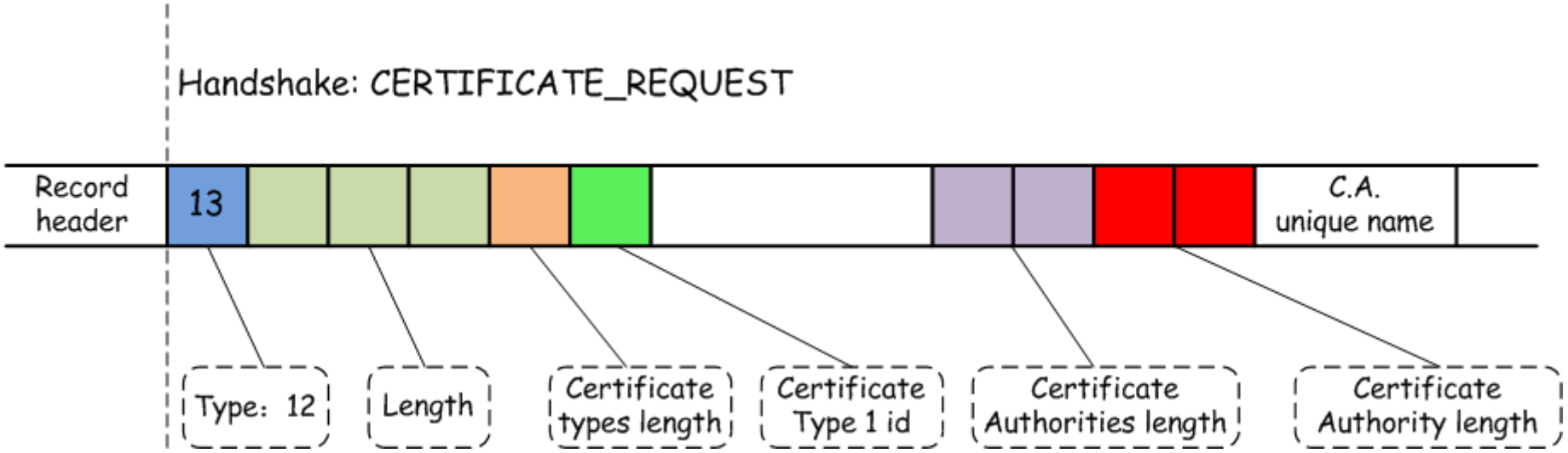
```
    TCP payload (338 bytes)
▲ Secure Sockets Layer
  ▲ TLSv1.2 Record Layer: Handshake Protocol: Server Key Exchange
      Content Type: Handshake (22)
      Version: TLS 1.2 (0x0303)
      Length: 333
    ▲ Handshake Protocol: Server Key Exchange
        Handshake Type: Server Key Exchange (12)
        Length: 329
      ▲ EC Diffie-Hellman Server Params
          Curve Type: named_curve (0x03)
          Named Curve: secp256r1 (0x0017)
          Pubkey Length: 65
          Pubkey: 046d975d136b8925e3c25092b2ef45f9008b8ce3aeff831e...
        ▷ Signature Hash Algorithm: 0x0401
          Signature Length: 256
          Signature: 0bb60306339417 24e483f3999f9a55f56a96d656df8e9a40...
```

```
0030  03 2c 7a ac 00 00 16 03  03 01 4d 0c 00 01 49 03   .,z..... ..M...I.
0040  00 17 41 04 6d 97 5d 13  6b 89 25 e3 c2 50 92 b2   ..A.m.]. k.%..P..
0050  ef 45 f9 00 8b 8c e3 ae  ff 83 1e b9 e5 d2 36 2e   .E...... ......6.
0060  cc 69 84 1a c3 95 ee a8  bd 39 6e cd 3a d1 91 d9   .i...... .9n.:...
0070  3c 33 d3 af 19 f6 0f 84  34 2d 43 4e 59 2f d9 f7   <3...... 4-CNY/..
0080  22 37 81 f8 04 01 01 00  0b b6 03 06 33 94 17 24   "7...... ....3..$
0090  e4 83 f3 99 9f 9a 55 f5  6a 96 d6 56 df 8e 9a 40   ......U. j..V...@
```

Handshake: CertificateRequest

服务器使用CertificateRequest消息请求对客户端进行身份验证，其中包含了服务器可以接受的证书类型列表，可接受的CA的列表。

CertificateRequest消息图示



CertificateRequest消息格式

```
struct {
      ClientCertificateType
certificate_types<1..2^8-1>;

      DistinguishedName
certificate_authorities<0..2^16-1>;
    } CertificateRequest;
```

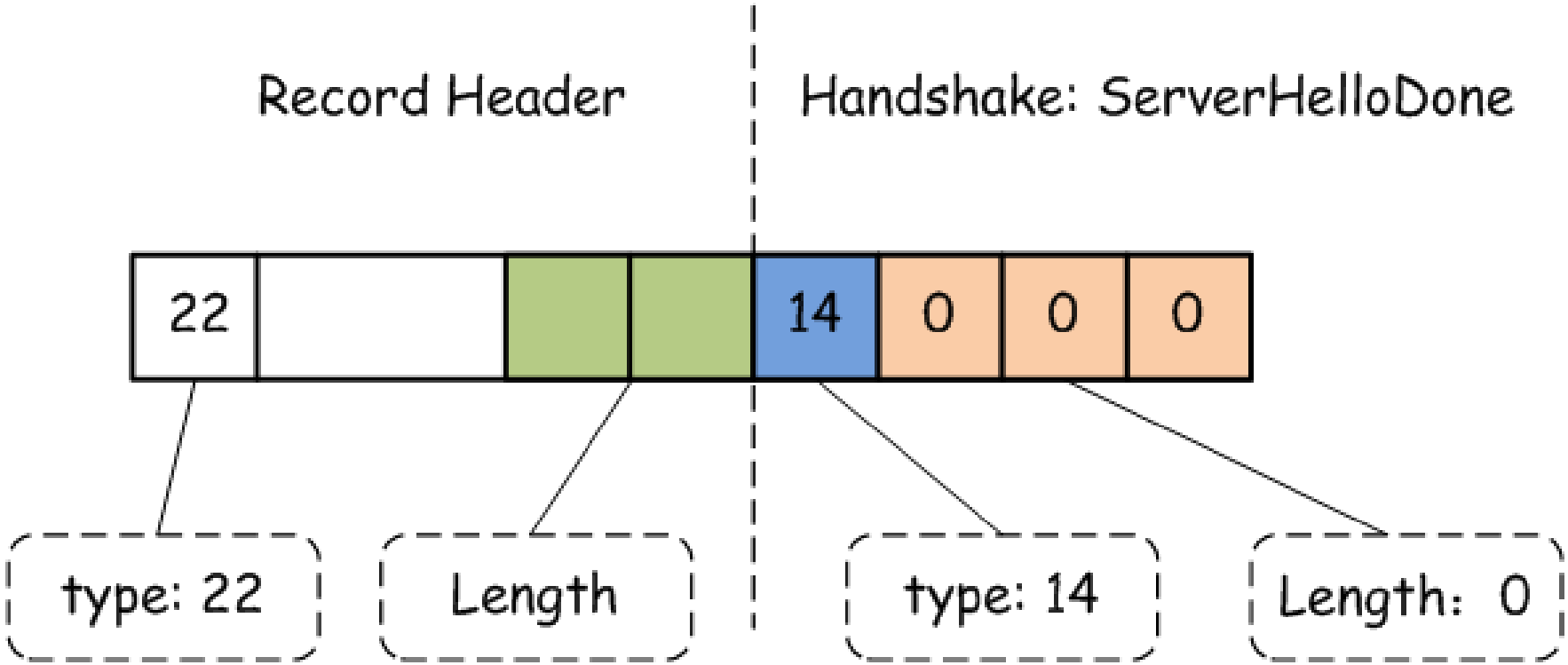
Handshake:ServerHelloDone

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。ServerHelloDone消息不包含任何内容。

ServerHelloDone消息

```
struct { } ServerHelloDone;
```



```
▷ Flags: 0x018 (PSH, ACK)
  Window size value: 812
  [Calculated window size: 25984]
  [Window size scaling factor: 32]
  Checksum: 0xe041 [unverified]
  [Checksum Status: Unverified]
  Urgent pointer: 0
▷ [SEQ/ACK analysis]
  TCP payload (9 bytes)
◢ 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
```

```
0000  84 3a 4b 13 e9 8c 50 bd  5f 4c 15 18 08 00 45 00   .:K...P. _L....E.
0010  00 31 7c b2 40 00 33 06  31 3c b4 61 21 6b c0 a8   .1|.@.3. 1<.a!k..
0020  03 64 01 bb e8 9d 16 c3  51 8f 5c 9c 66 24 50 18   .d...... Q.\.f$P.
0030  03 2c e0 41 00 00 16 03  03 00 04 0e 00 00 00      .,.A.... .......
```

Handshake:ClientKeyExchange

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。

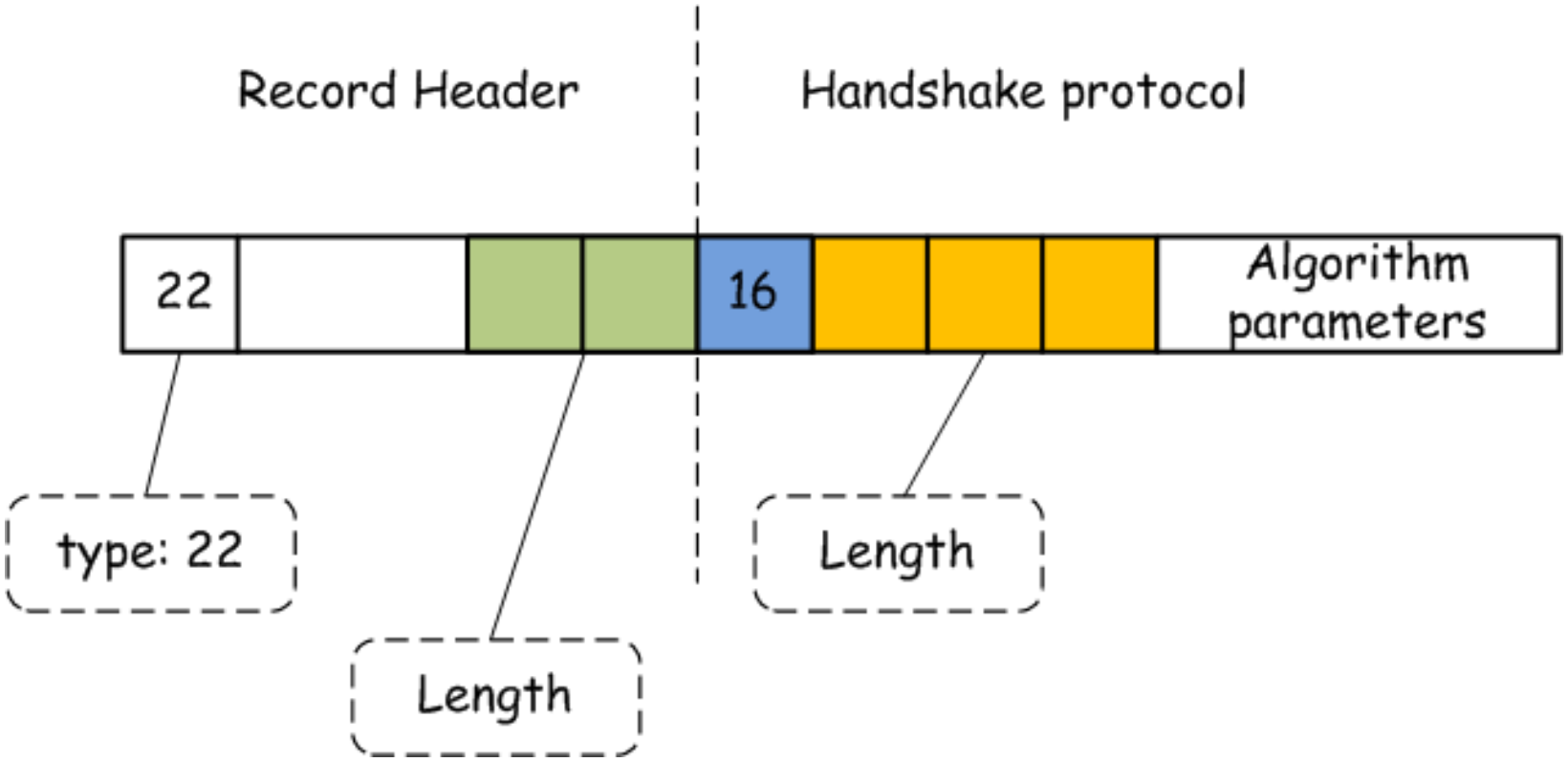
ClientKeyExchange

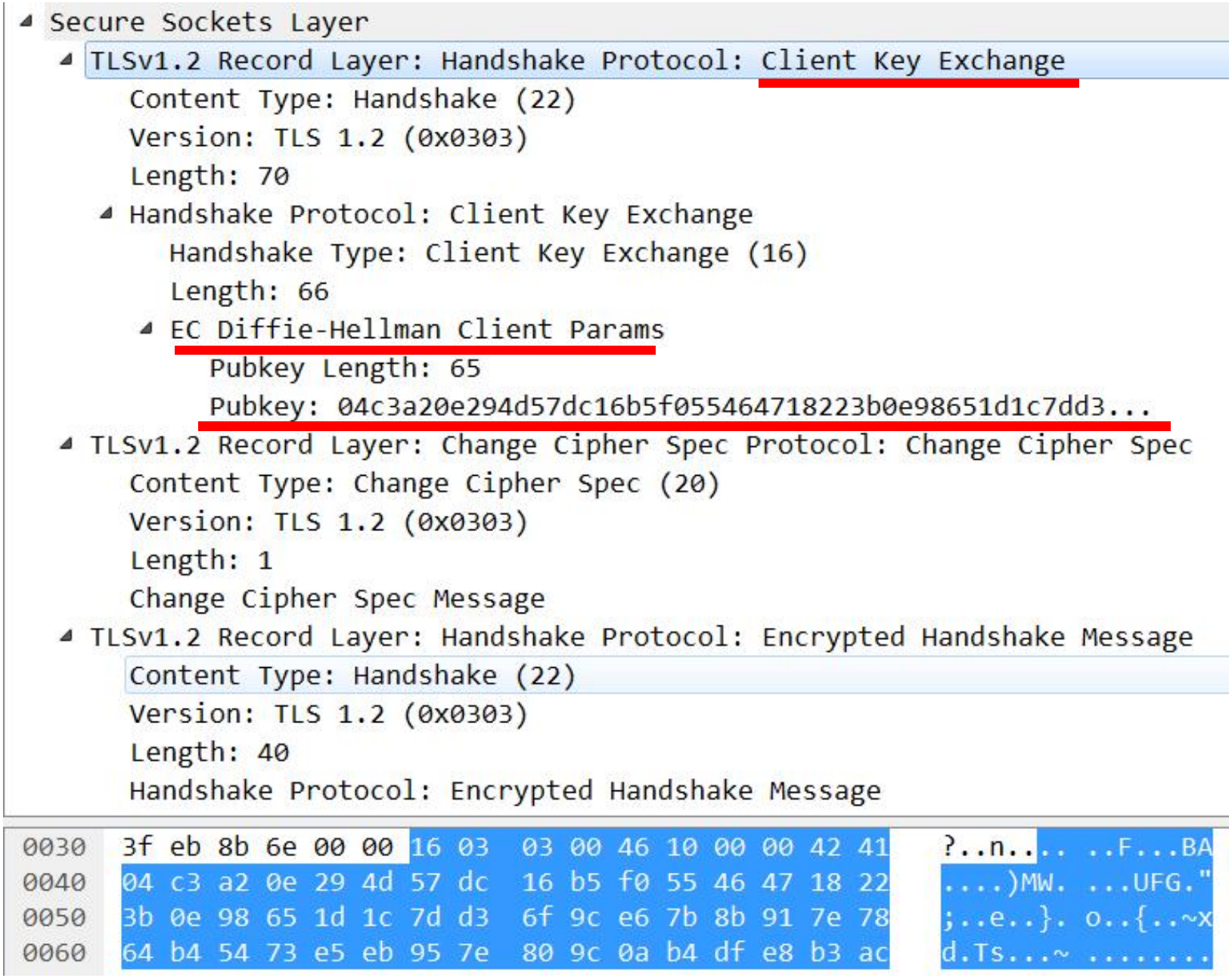
- 客户端发送ClientKeyExchange消息，向服务器提供用于生成对称加密密钥等所需的数据，相当于告诉服务器："这是经过加密的**Premaster secret**。"
- 当密码套件包含RSA时，会随ClientKeyExchange消息发送经过RSA-encrypted的premaster secret。
- 当密码套件包含Diffie-Hellman密钥交换时，会随ClientKeyExchange消息发送Diffie-Hellman的公开值。

ClientKeyExchange消息格式

```
struct {
    select (KeyExchangeAlgorithm) {
        case rsa:
            EncryptedPreMasterSecret;
        case dhe_dss:
        case dhe_rsa:
        case dh_dss:
        case dh_rsa:
        case dh_anon:
            ClientDiffieHellmanPublic;
    } exchange_keys;
} ClientKeyExchange;
```

ClientKeyExchange消息图示




Secure Sockets Layer
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: 04c3a20e294d57dc16b5f055464718223b0e98651d1c7dd3...
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
0030 3f eb 8b 6e 00 00 16 03 03 00 46 10 00 00 42 41 ?..n.. .. ..F...BA
0040 04 c3 a2 0e 29 4d 57 dc 16 b5 f0 55 46 47 18 22 ....)MW. ...UFG."
0050 3b 0e 98 65 1d 1c 7d d3 6f 9c e6 7b 8b 91 7e 78 ;..e..}. o..{..~x
0060 64 b4 54 73 e5 eb 95 7e 80 9c 0a b4 df e8 b3 ac d.Ts...~ ....

Handshake: CertificateVerify

When:

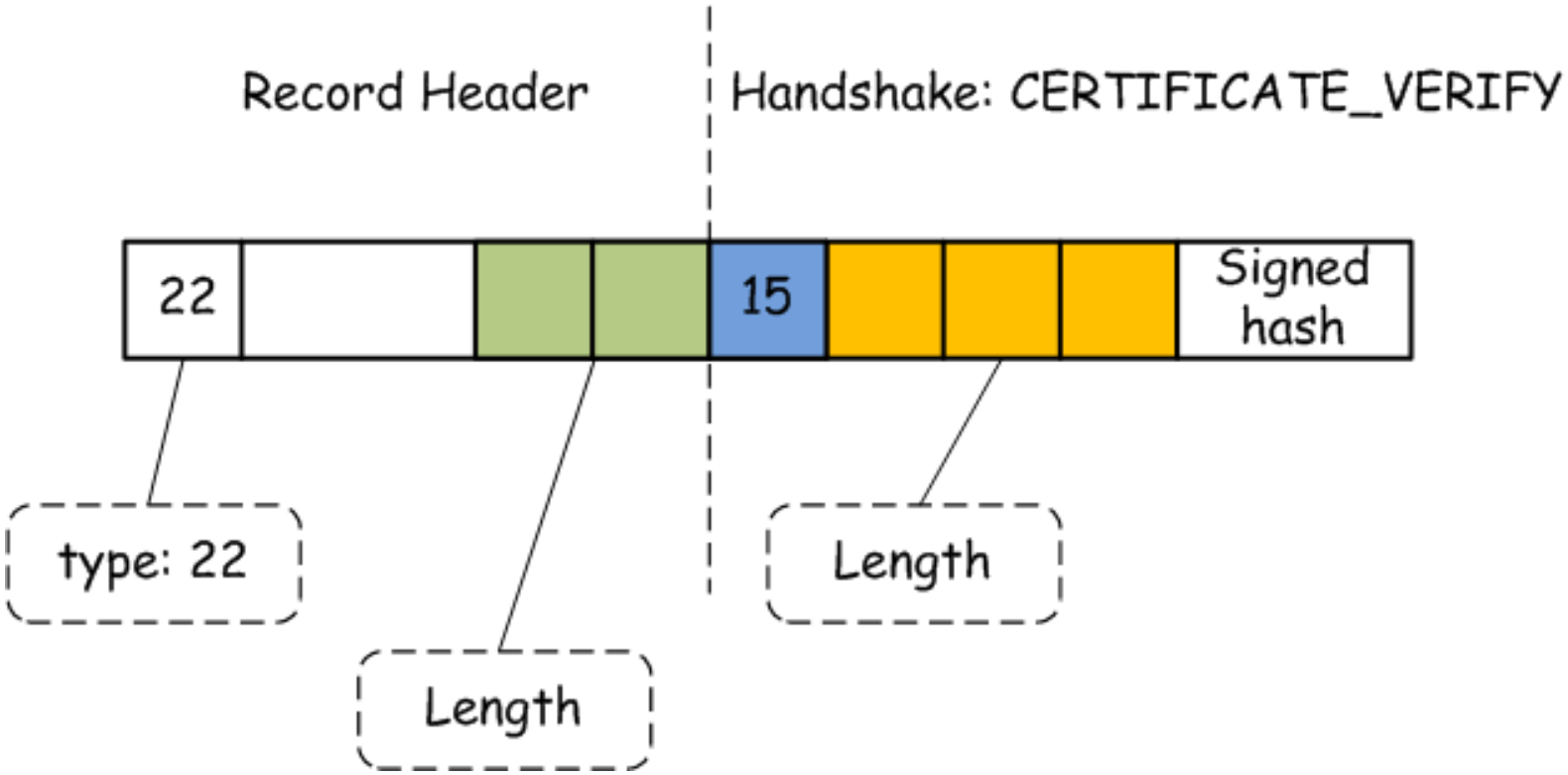
只有服务器向客户端发送CertificateRequest消息的情况下，客户端才会向服务器发送CertificateVerify消息，以向服务器证明自己的确持有客户端证书的私钥。

CertificateVerify消息格式

```
struct {
digitally-signed struct {
                opaque
handshake_messages[handshake_messages_length];
            }
      } CertificateVerify;
```

handshake_messages：迄今为止收到和发送的所有握手消息的拼接，从client hello开始，不包括当前这个消息。包括握手消息的type字段和length字段。用于证明，客户端确实就是证书的拥有者。

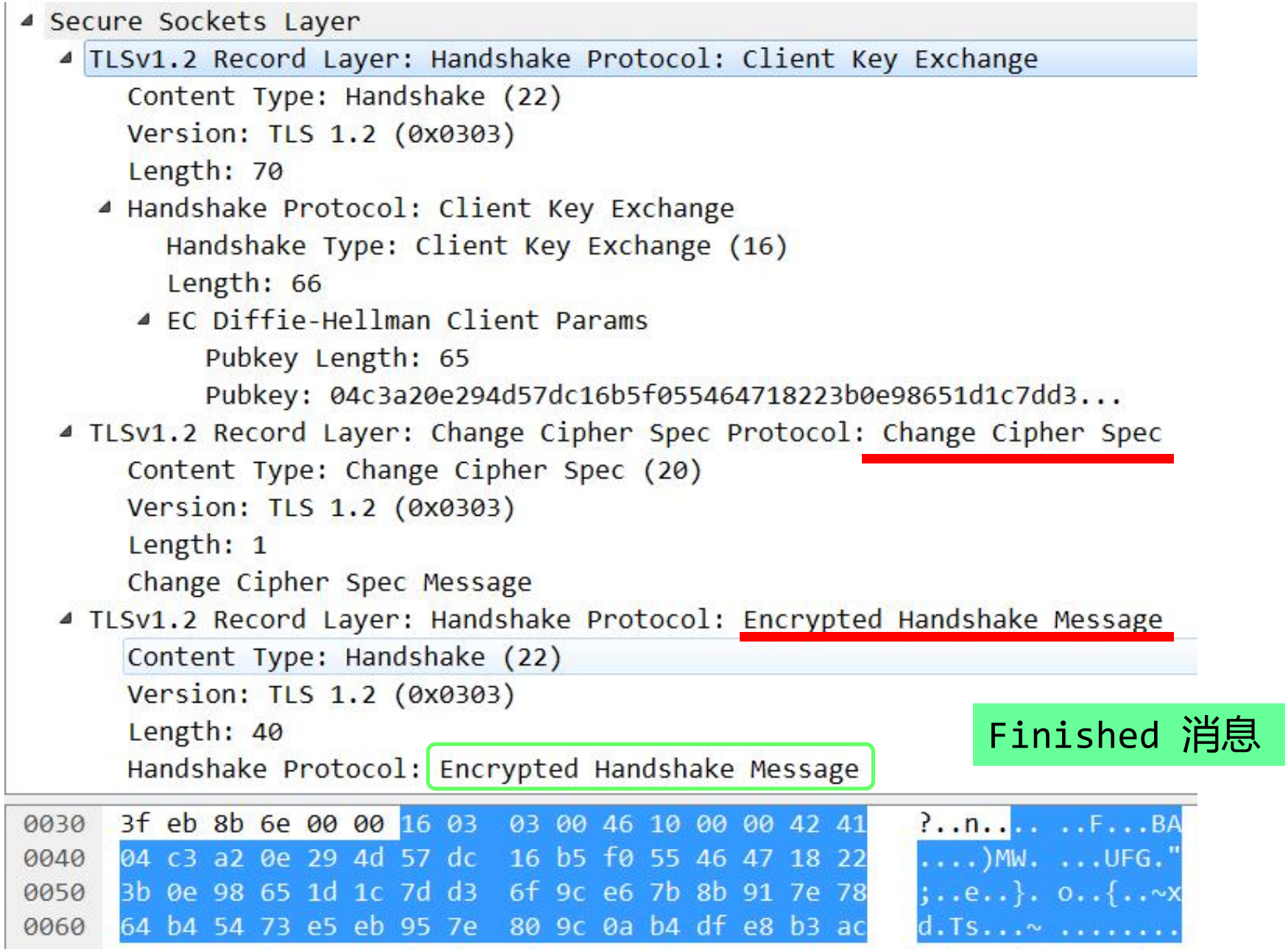
CertificateVerify消息图示



Handshake: Finished

Handshake: FINISHED

- When:
  - 发送ChangeCipherSpec来激活已经协商好的密码套件之后，客户端发送Finished消息，表明TLS握手协商完成，相当于告诉服务器“握手结束。”
- 密文？明文:
  - 由于已经完成了密码规格切换，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。


Secure Sockets Layer
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: 04c3a20e294d57dc16b5f055464718223b0e98651d1c7dd3...
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
0030 3f eb 8b 6e 00 00 16 03 03 00 46 10 00 00 42 41 ?..n.. .. ..F...BA
0040 04 c3 a2 0e 29 4d 57 dc 16 b5 f0 55 46 47 18 22 ....)MW. ...UFG."
0050 3b 0e 98 65 1d 1c 7d d3 6f 9c e6 7b 8b 91 7e 78 ;..e..}. o..{..~x
0060 64 b4 54 73 e5 eb 95 7e 80 9c 0a b4 df e8 b3 ac d.Ts...~ ....
Finished 消息

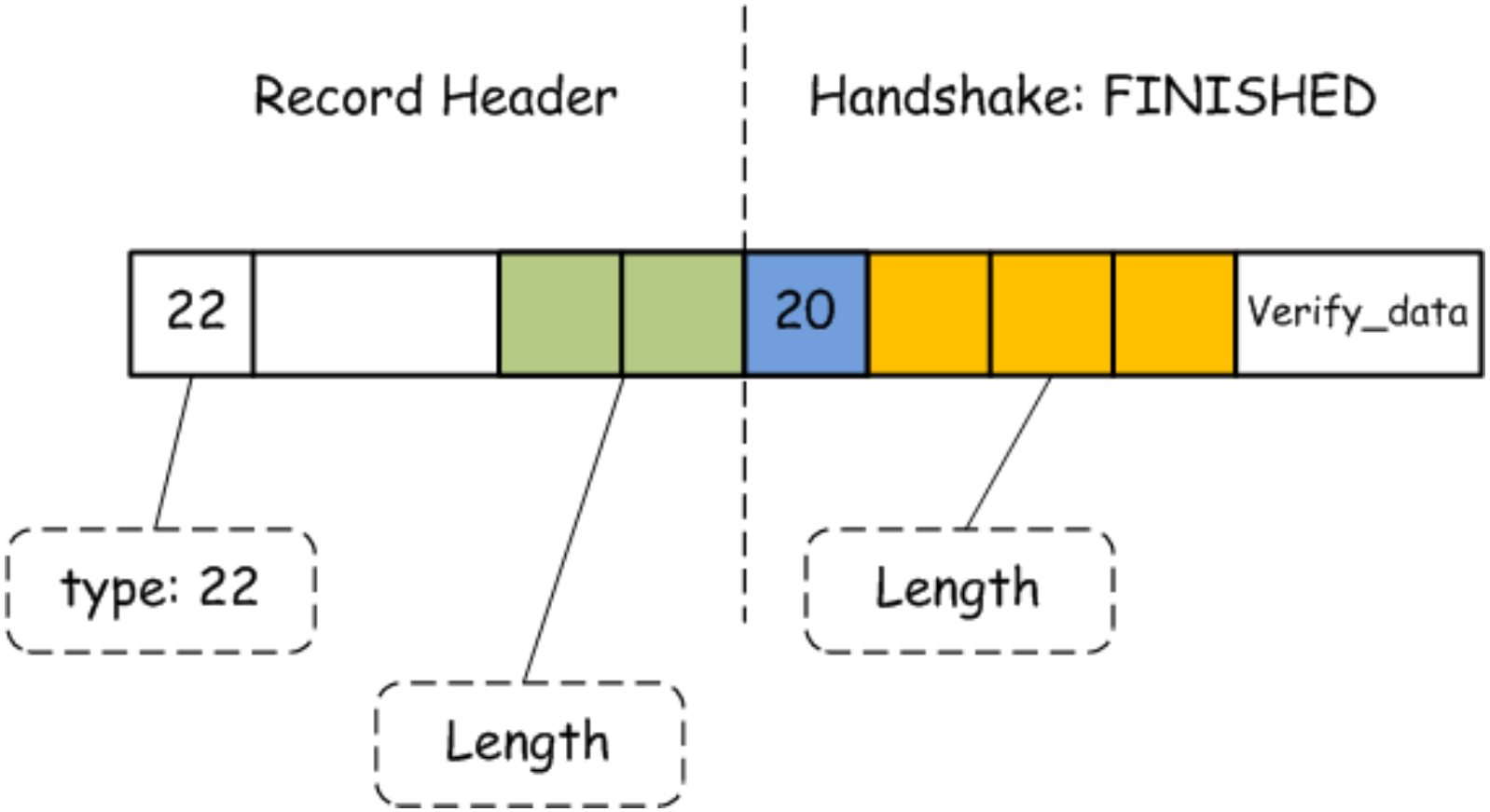
Handshake: FINISHED

- Finished消息的内容采用PRF函数生成，其输入包括： master_secret，finished_label,之前所有handshake消息组合的hash值。
- 服务器可以通过对收到的消息进行验证来确认收到的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。

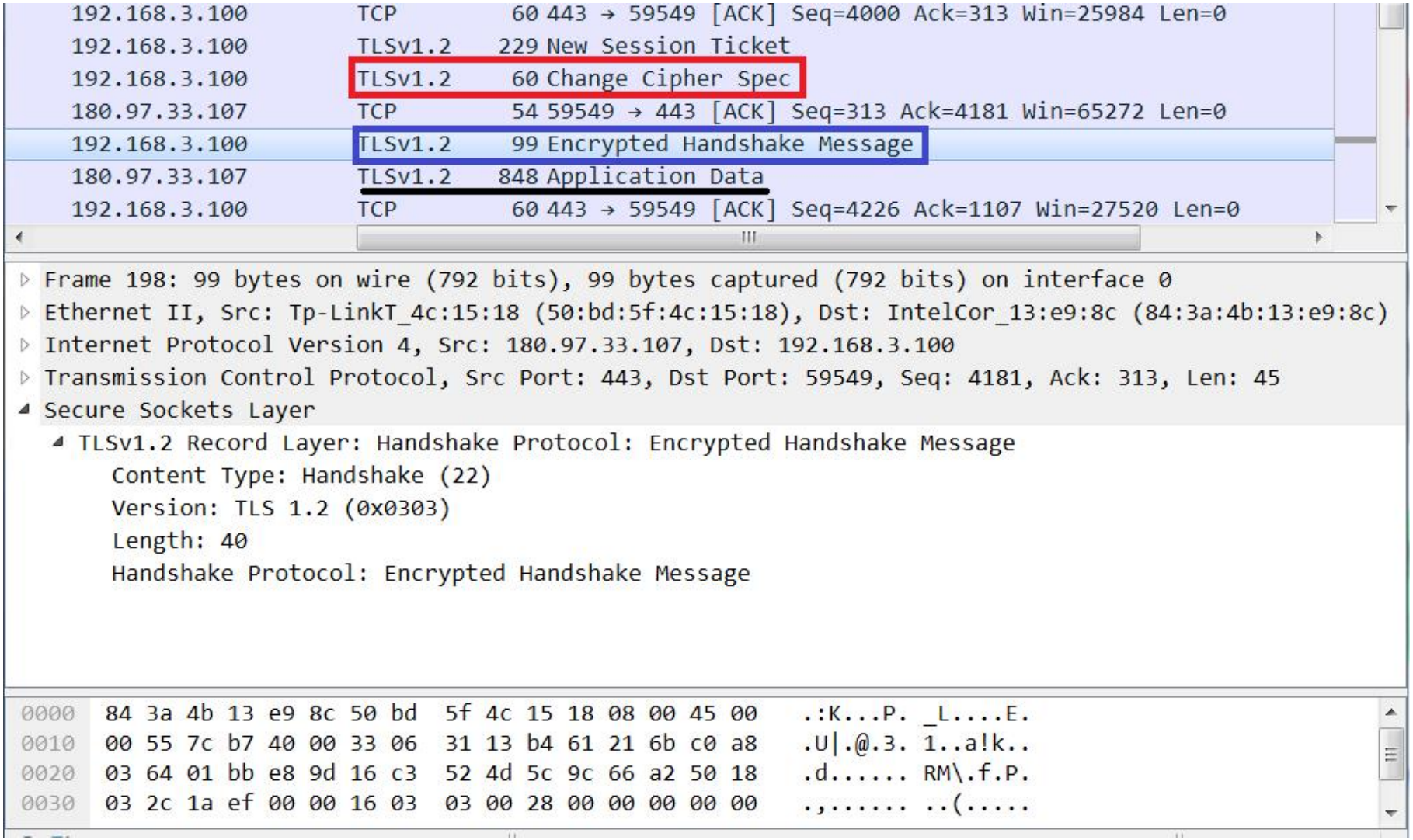
Handshake: FINISHED内容

```
struct {
      opaque verify_data[verify_data_length];
       } Finished;


verify_data PRF(master_secret, finished_label,
Hash(handshake_messages))[0..verify_data_length-1];


finished_label:
Client: “client finished”
Server: “server finished”
```

Handshake: FINISHED



Finished消息截图



Finished消息：防止降级攻击

- 什么是降级攻击？
  - MiTM攻击者修改ClientHello消息里面的密码套件列表，用较弱的密码套件替换较强的密码套件，从而降低所建立的通信信道的安全性。

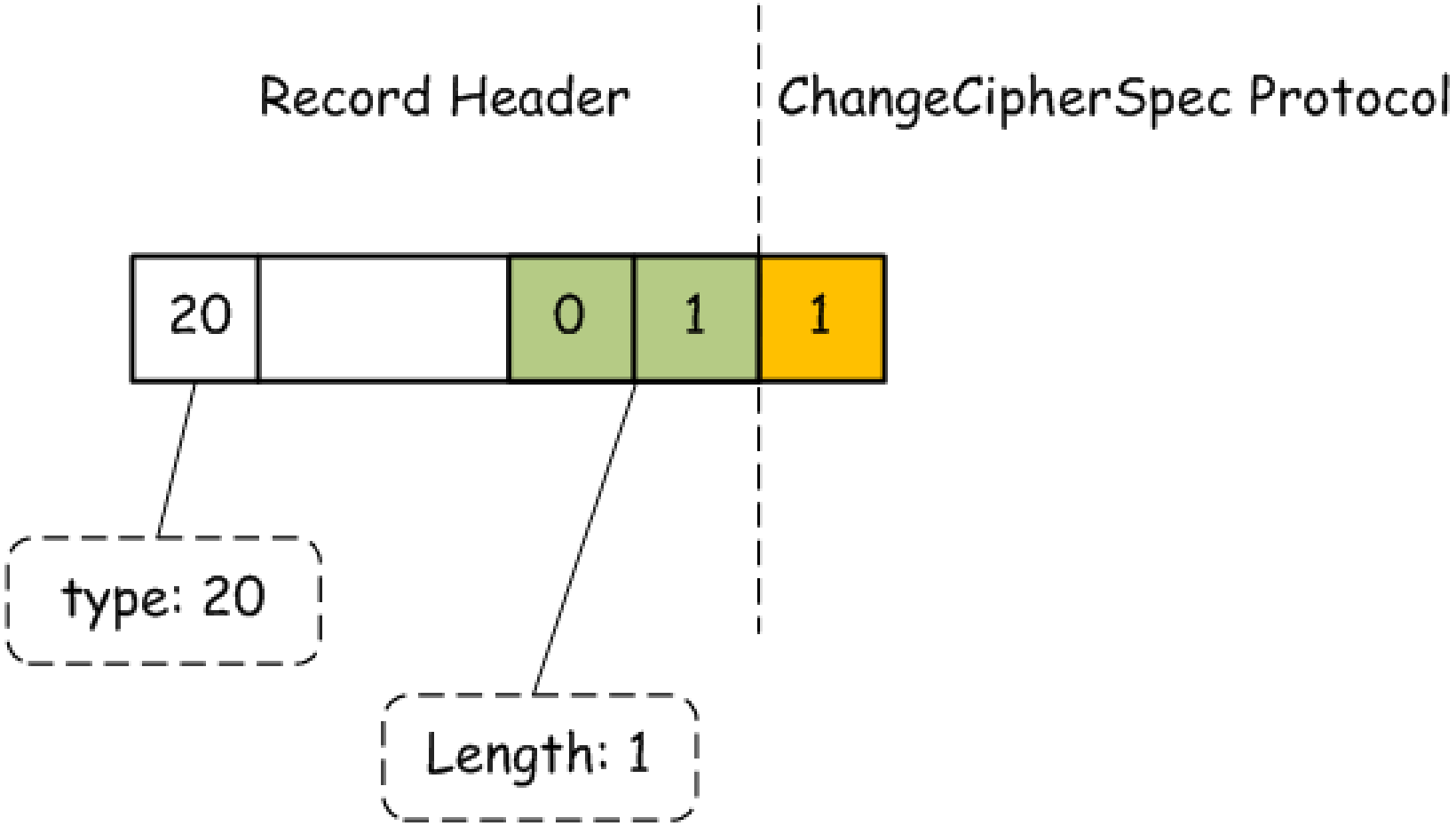
Finished消息，包含了所有握手消息的hash，如果攻击者篡改前面的握手消息，则无法通过验证。

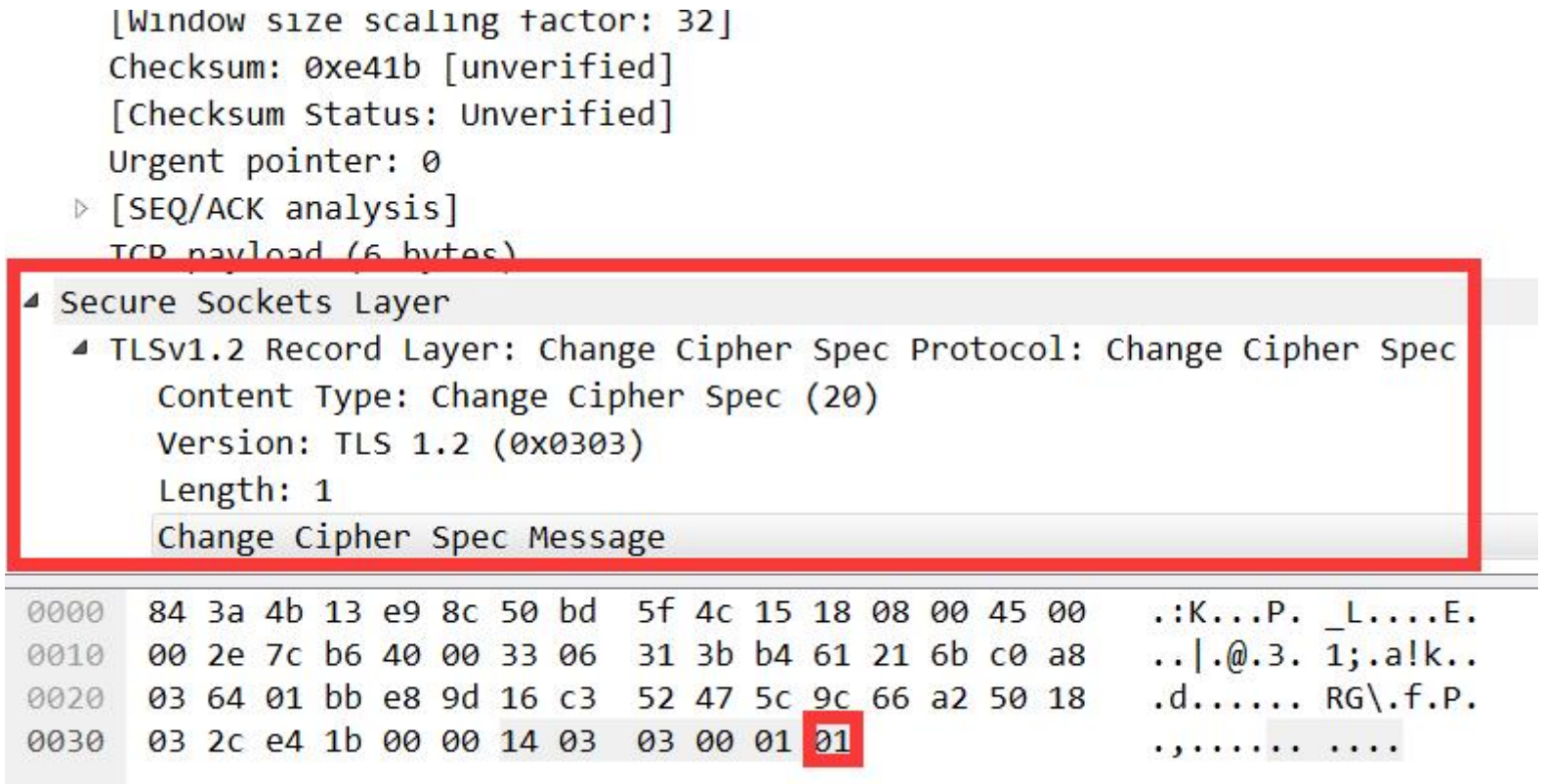
Changecipherspec protocol

ChangeCipherSpec

```
struct {
      enum { change_cipher_spec(1), (255) } type;
  } ChangeCipherSpec;
```

ChangeCipherSpec




[Window size scaling factor: 32]
Checksum: 0xe41b [unverified]
[Checksum Status: Unverified]
Urgent pointer: 0
[SEQ/ACK analysis]
TCP payload (6 bytes)
Secure Sockets Layer
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
0000 84 3a 4b 13 e9 8c 50 bd 5f 4c 15 18 08 00 45 00 .:K...P. _L....E.
0010 00 2e 7c b6 40 00 33 06 31 3b b4 61 21 6b c0 a8 ..|.@.3. 1;.a!k..
0020 03 64 01 bb e8 9d 16 c3 52 47 5c 9c 66 a2 50 18 .d...... RG\.f.P.
0030 03 2c e4 1b 00 00 14 03 03 00 01 01 .,...... ....

Alert Protocol

Alert protocol

- 以简单的通知机制告知通信对端出现异常状况
  - 报告错误
  - 通知连接关闭

Alert Protocol

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

AlertLevel

```
enum {

        warning(1), fatal(2), (255)

} AlertLevel;
```

AlertLevel

- Alert消息分两类：警告消息（warning ）和致命消息（fatal ）。
- 致命消息将导致连接被立即中止，并将与这个连接相关的会话（SessionID）作废，以免这个会话被继续用来建立新的连接。
- 警告消息仅仅是通告对方有关报警信息，不会导致连接的关闭。

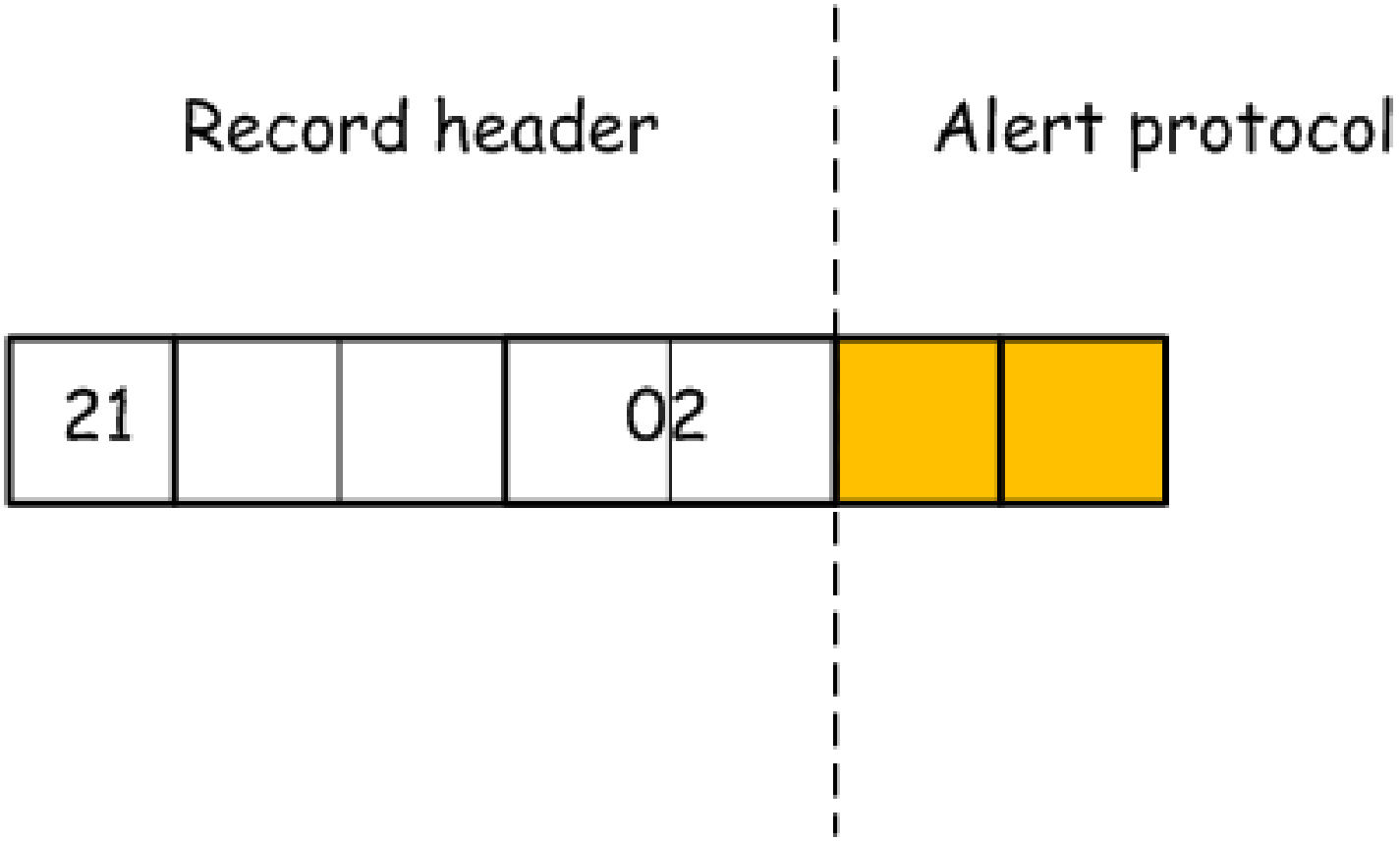
Alert Description

```
enum {
      close_notify(0),
      unexpected_message(10),
      bad_record_mac(20),
      decryption_failed_RESERVED(21),
      record_overflow(22),
      decompression_failure(30),
      handshake_failure(40),
      no_certificate_RESERVED(41),
      bad_certificate(42),
      unsupported_certificate(43),
      certificate_revoked(44),
      certificate_expired(45),
      certificate_unknown(46),
      illegal_parameter(47),
      unknown_ca(48),
      access_denied(49),
      decode_error(50),
      decrypt_error(51),
      export_restriction_RESERVED(60),
      protocol_version(70),
      insufficient_security(71),
      internal_error(80),
      user_canceled(90),
      no_renegotiation(100),
      unsupported_extension(110),  (255)
        } AlertDescription;
```

TLS报警消息类型与触发原因

报警消息	触发原因	类型
Bad_certificate	证书验证失败	致命
Bad_record_mac	收到不正确的消息验证码	致命
Certificate_expired	证书过期	致命
Certificate_revoked	证书已撤销	致命
Certificate_unknown	无法验证证书	致命
Close_notify	主动中断会话	致命
Decompression_failure	解压错误	致命
Handshake_failure	无法协商公用参数	致命
Illegal_parameter	握手过程中参数不一致或超限制	致命
No_certificate	无法回应证书请求	警告/致命
Unexpected_message	到达消息非预期	致命
Unsupported_certficate	收到不支持类型的证书	警告/致命

Alert消息 封装

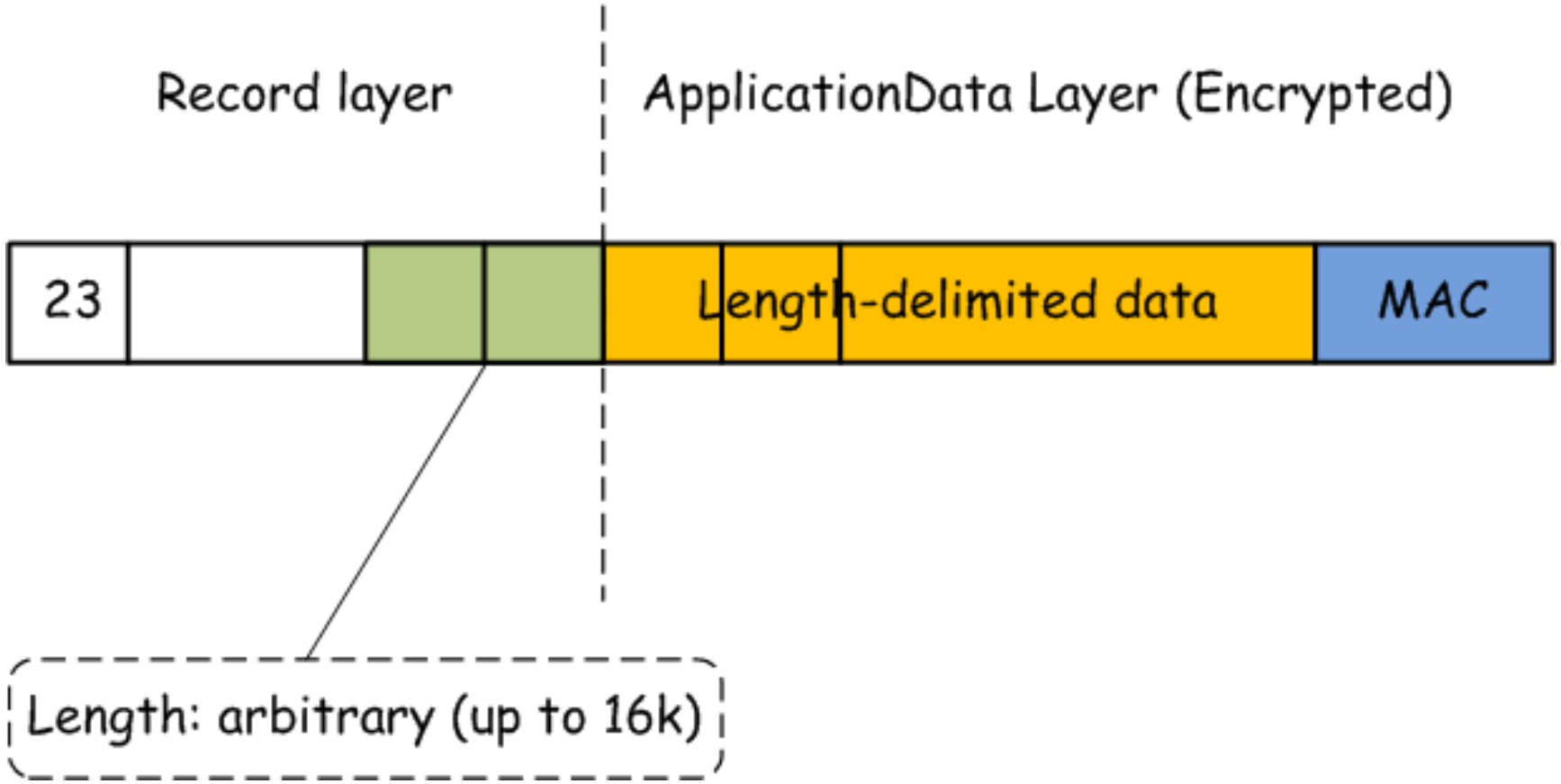


Closure alert（关闭连接警报）

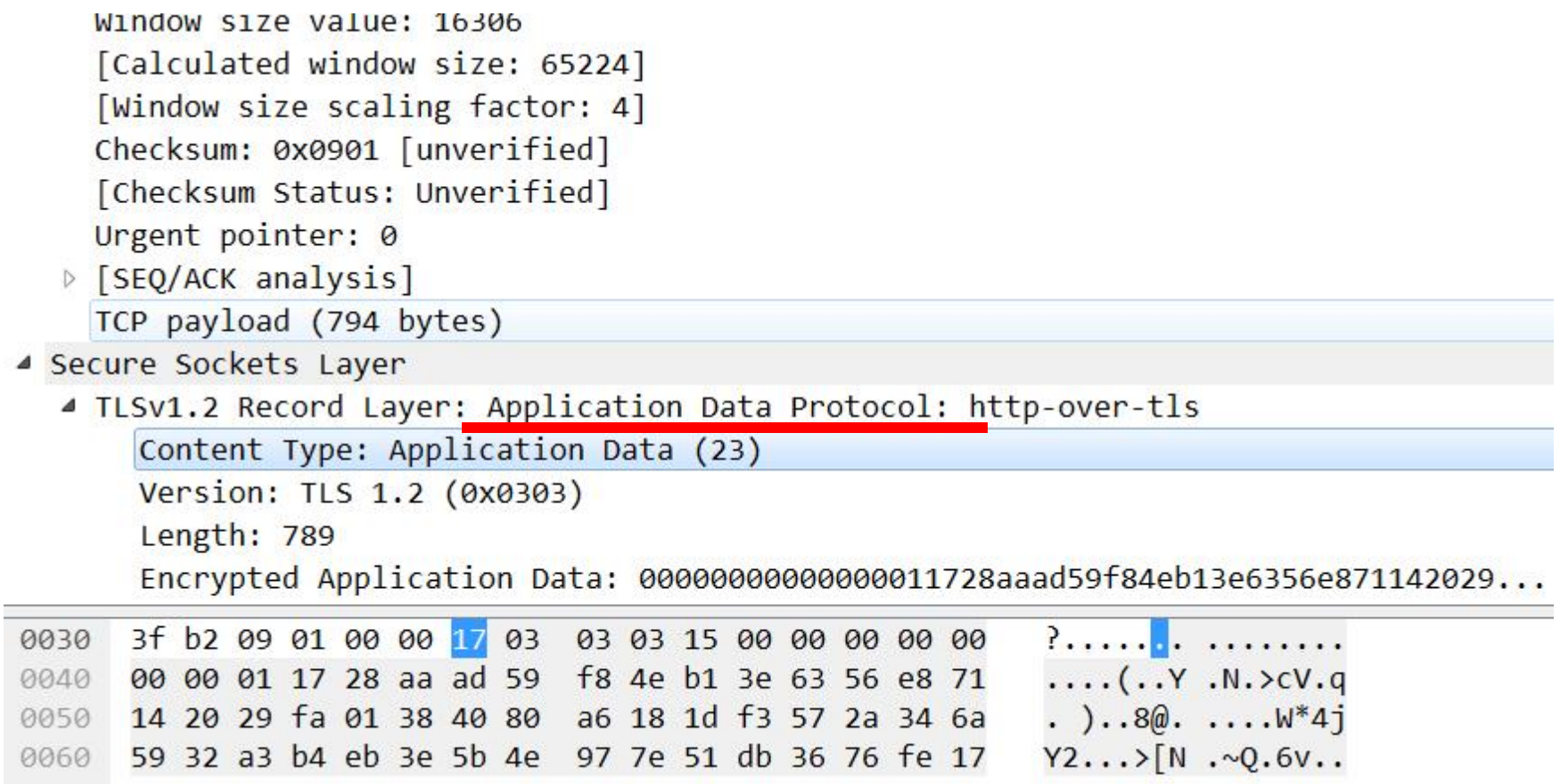
- 一旦一端确定无数据发送，决定关闭连接，就会发送一个close_notify警报。另一端收到这个警报后，会丢弃任何还未写出的数据，并发送自己的close_notify警报。警报之后的任何消息都会被忽略。
- 目的：防止truncation attack（截断攻击）
  - 攻击者主动发送TCP FIN包，提前结束通信，阻断后续消息的发送。
  - 如果没有close_notify警报功能，通信双方无法确定是被攻击了还是通信真正结束了。

Application Data Protocol

ApplicationData Layer



Application data截图



The Illustrated TLS Connection

https://tls.ulfheim.net/

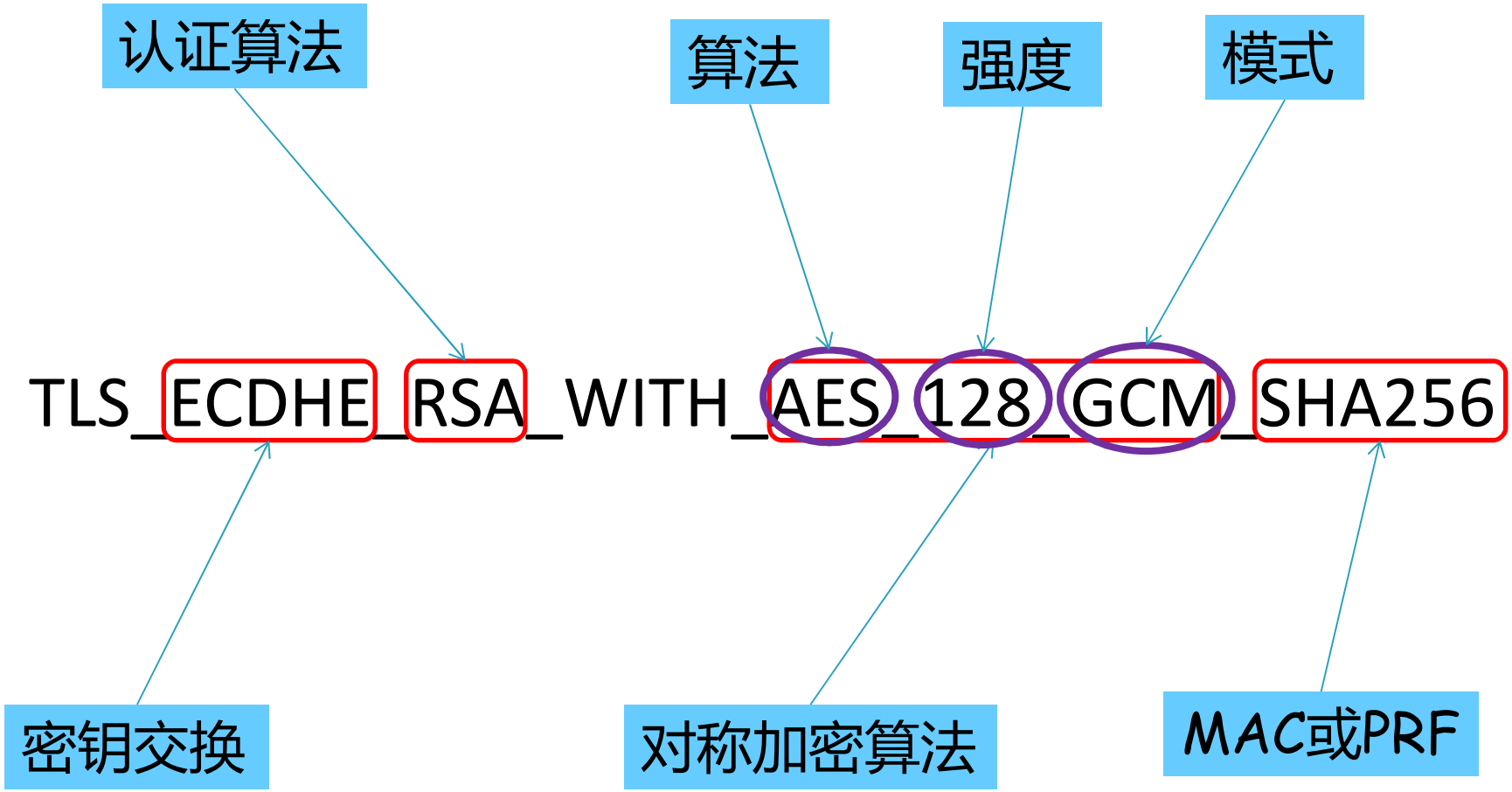
可以参考该网站进一步理解TLS的原理和流程

TLS Cipher Suite

密码套件：属性

- 身份认证算法
- 密钥交换算法
- 加密算法
- 加密密钥大小
- 加密算法模式（可应用时）
- MAC算法（可应用时）
- PRF（TLS1.2）
- 用于Finished消息的散列函数（TLS1.2）

密码套件格式


认证算法
算法
强度
模式
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
密钥交换
对称加密算法
MAC或PRF

密码套件名称及解析

密码套件名称	签　名 算　法	密　钥 交换	对称加密算法	MAC	PRF
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	RSA	ECDHE	AES-128-GCM	-	SHA256
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	ECDSA	ECDHE	AES-256-GCM	-	SHA384

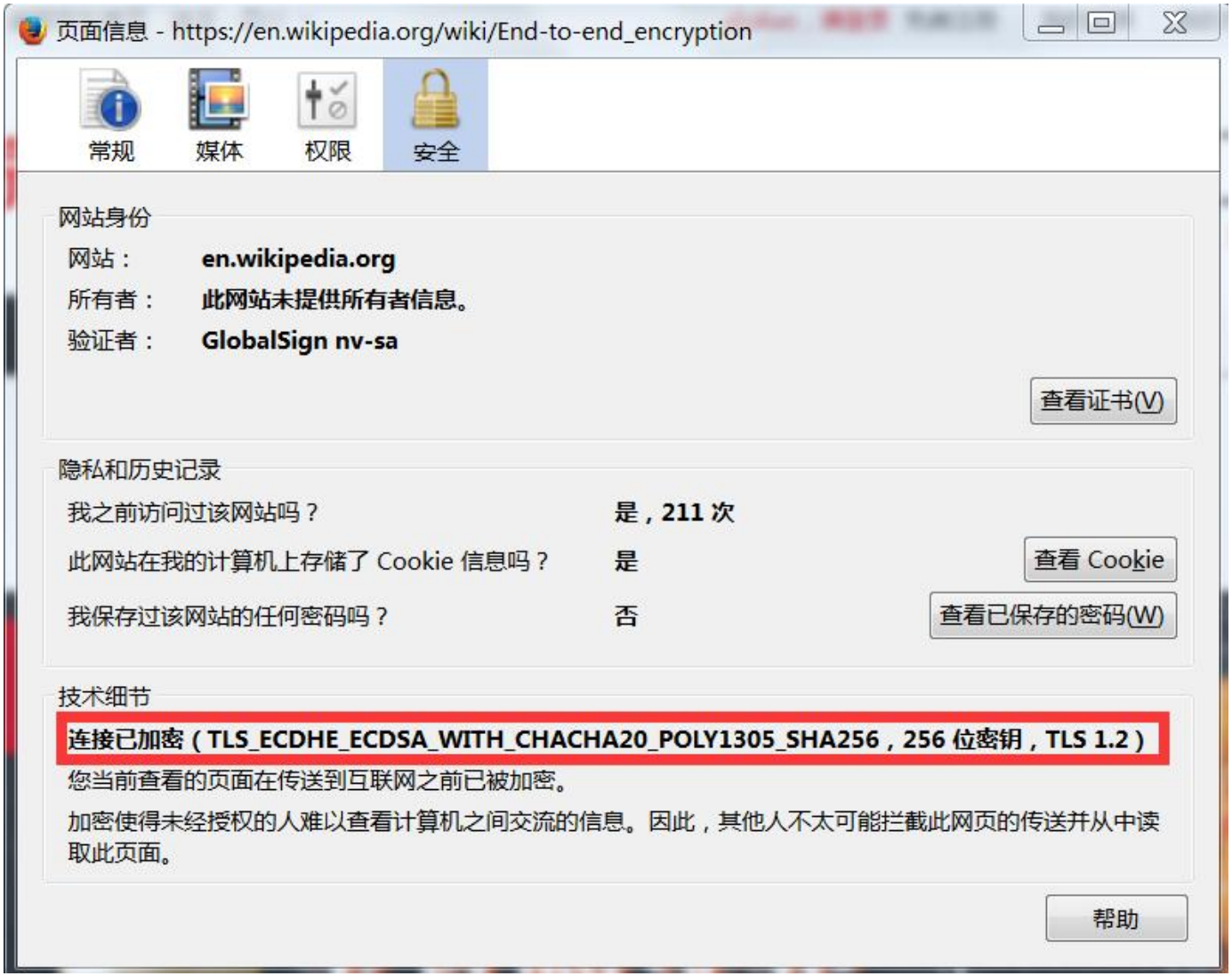
密码套件：IANA列表

- 满足要求的组合很多，因此IANA给每一个密码套件分配一个2字节编号来标识密码套件。

Value	Description	DTLS-OK	Reference
0xCC, 0xAE	TLS_RSA_PSK_WITH_CHACHA20_POLY1305_SHA256	Y	[RFC7905]
0xCC, 0xAD	TLS_DHE_PSK_WITH_CHACHA20_POLY1305_SHA256	Y	[RFC7905]
0xCC, 0xAC	TLS_ECDHE_PSK_WITH_CHACHA20_POLY1305_SHA256	Y	[RFC7905]
0xCC, 0xAB	TLS_PSK_WITH_CHACHA20_POLY1305_SHA256	Y	[RFC7905]
0xCC, 0xAA	TLS_DHE_RSA_WITH_CHACHA20_POLY1305_SHA256	Y	[RFC7905]
0xCC, 0xA9	TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256	Y	[RFC7905]
0xCC, 0xA8	TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256	Y	[RFC7905]

注：IANA(The Internet Assigned Numbers Authority,互联网数字分配机构)

查看网站的密码套件


页面信息 - https://en.wikipedia.org/wiki/End-to-end_encryption
常规
媒体
权限
安全
网站身份
网站： en.wikipedia.org
所有者： 此网站未提供所有者信息。
验证者： GlobalSign nv-sa
查看证书(V)
隐私和历史记录
我之前访问过该网站吗？ 是，211 次
此网站在我的计算机上存储了 Cookie 信息吗？ 是
查看 Cookie
我保存过该网站的任何密码吗？ 否
查看已保存的密码(W)
技术细节
连接已加密（TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256，256 位密钥，TLS 1.2）
您当前查看的页面在传送到互联网之前已被加密。
加密使得未经授权的人难以查看计算机之间交流的信息。因此，其他人不太可能拦截此网页的传送并从中读取此页面。
帮助

特殊密码套件

- 在TLS的初始连接时，使用什么CipherSuite?
- CipherSuite TLS_NULL_WITH_NULL_NULL = { 0x00,0x00 };
- 该密码套件不可协商，而且不提供任何安全保护。

问题?

如何确定一个网站配置了哪些密码套件？有没有不安全的密码套件？或者如何扫描一个https网站的是否有配置弱点呢?

命令：

```
openssl s_client –connect www.baidu.com:443 –tls1_2
-cipher ECDHE-RSA-AES128-GCM-SHA256
```

命令结果

```
6 - G2
---
No client certificate CA names sent
Peer signing digest: SHA256
Server Temp Key: ECDH, P-256, 256 bits
---
SSL handshake has read 4137 bytes and written 250 bytes
Verification error: unable to get local issuer certificate
---
New, TLSv1.2, Cipher is ECDHE-RSA-AES128-GCM-SHA256
Server public key is 2048 bit
Secure Renegotiation IS supported
Compression: NONE
Expansion: NONE
No ALPN negotiated
SSL-Session:
    Protocol  : TLSv1.2
    Cipher    : ECDHE-RSA-AES128-GCM-SHA256
    Session-ID: 7B7186435D01142F914FC5A076E372946CC08FD41A426921E75FD6548FADD190

    Session-ID-ctx:
    Master-Key: 7FCEBFAF62A66ABFA73C25F809C5A0BEFBB84BE36B9F5076E6D05E98595C662F
34587F992CADE700DC207EBDF6B0A4F2
    PSK identity: None
    PSK identity hint: None
```

sslyze工具

命令：

```
sslyze --regular www.baidu.com:443
```

```
* OpenSSL CCS Injection:
                                  OK - Not vulnerable to OpenSSL CCS injection

* Resumption Support:
     With Session IDs:            OK - Supported (5 successful, 0 failed, 0 errors, 5 total a
ttempts).
     With TLS Tickets:            OK - Supported

* TLSV1_2 Cipher Suites:
      Forward Secrecy             OK - Supported
      RC4                         INSECURE - Supported

    Preferred:
       TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256                  128 bits      HTTP 200 OK

    Accepted:
       TLS_RSA_WITH_RC4_128_SHA                               128 bits      HTTP 200 OK

       TLS_RSA_WITH_AES_256_CBC_SHA                           256 bits      HTTP 200 OK

       TLS_RSA_WITH_AES_128_CBC_SHA                           128 bits      HTTP 200 OK

       TLS_ECDHE_RSA_WITH_RC4_128_SHA                         128 bits      HTTP 200 OK

       TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA                     256 bits      HTTP 200 OK

       TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256                  128 bits      HTTP 200 OK
```

密钥导出

TLS的密钥

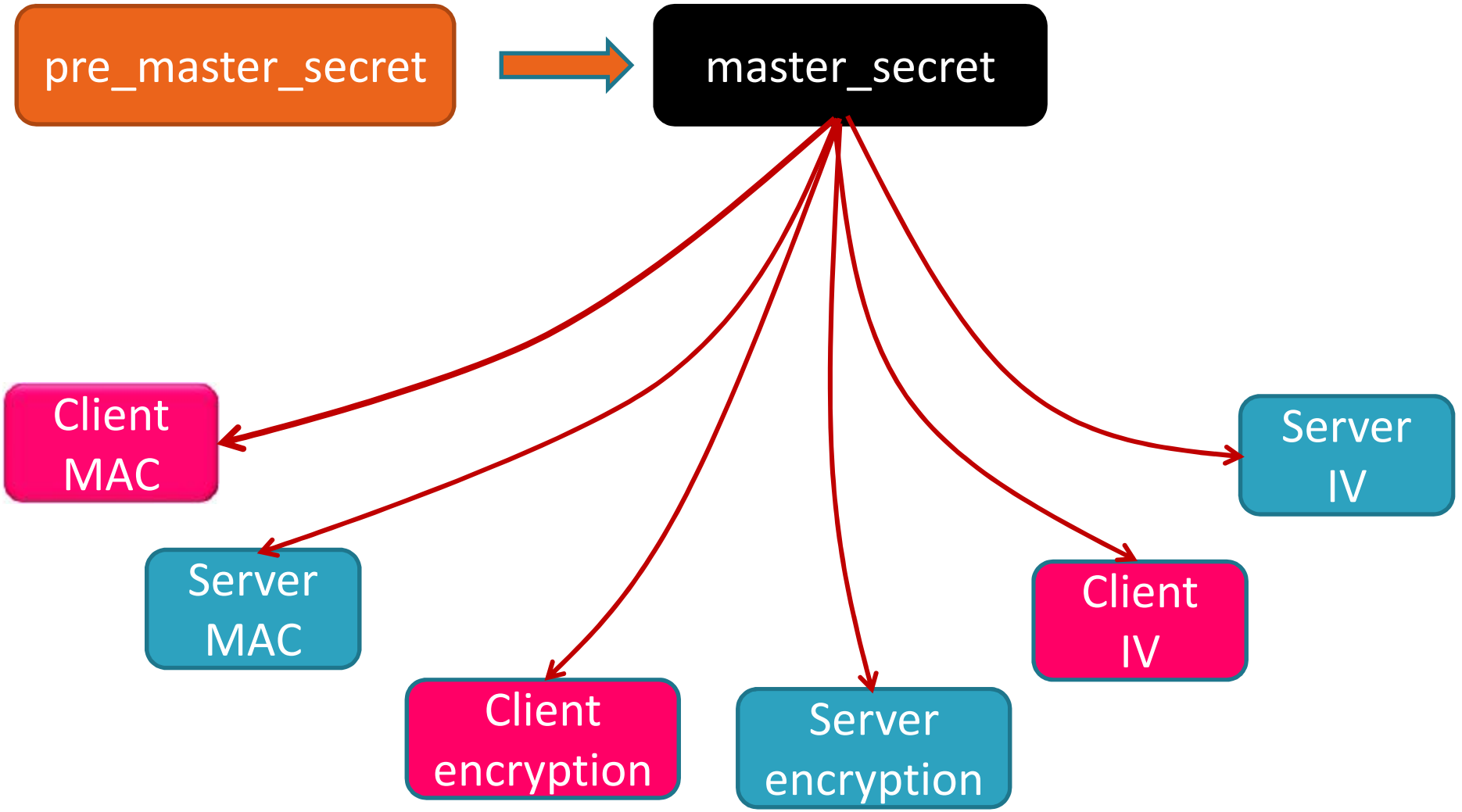
- 密码学的实践原则之一：对于不同的使用目的，应该使用不同的密钥。其主要目的是提高整个系统的安全性，即便某个密钥泄漏，也不至于整个系统都被攻破。
- TLS中，需要用到三种密钥：
  - 用于消息加密的对称密码密钥
  - CBC模式的初始化向量
  - 用于消息认证码的密钥

对于新的AEAD型算法，不需要MAC key，这里暂不讨论AEAD

TLS密钥

- client write MAC key
- server write MAC key
- client write encryption key
- server write encryption key
- client write IV
- server write IV

TLS密钥生成流程


pre_master_secret
master_secret
Client MAC
Server MAC
Client encryption
Server encryption
Client IV
Server IV

pre_master_secret?

- 如果采用RSA进行密钥协商
  - client 生成一个随机值，在发送ClientKeyExchange消息时用server的公钥加密，发送给server，最后双方均具有这个秘密，即为**pre_master_secret**。

代码说明如下：

ClientKeyExchange消息

```
struct {
      select (KeyExchangeAlgorithm) {
          case rsa:
              EncryptedPreMasterSecret;
          case dhe_dss:
          case dhe_rsa:
          case dh_dss:
          case dh_rsa:
          case dh_anon:
              ClientDiffieHellmanPublic;
      } exchange_keys;
  } ClientKeyExchange;
```

PreMasterSecret

```
struct {
    ProtocolVersion client_version;
    opaque random[46];
   } PreMasterSecret;

struct {
  public-key-encrypted
PreMasterSecret pre_master_secret;
   } EncryptedPreMasterSecret;
```

pre_master_secret?

- 如果采用DH进行密钥协商
  - Client在发送ClientKeyExchange消息时，将DH的公开值发送给server。server也会发送一个DH公开值给Client。根据DH算法，最终通信双方获得一个相同的秘密值，即为**pre_master_secret**。

pre_master_secret→master_secret?

- master_secret: 48字节数值

```
master_secret = PRF(pre_master_secret, "master secret",

ClientHello.random + ServerHello.random) [0..47];
```

- pre_master_secret：来自于handshake阶段，client和server协商出的一个秘密数。
- "master secret"：一个常量标签字符串，用于表示所生成数据的使用目的。
- ClientHello.random：客户端随机数。
- ServerHello.random：服务器随机数。

TLS PRF定义

PRF(secret, label, seed) = P_<hash>(secret, label + seed)

P_hash(secret, seed) = HMAC_hash(secret, A(1) + seed) +

HMAC_hash(secret, A(2) + seed) +

HMAC_hash(secret, A(3) + seed) + ...

TLS PRF定义

A( ) 定义如下:

A(0) = seed

A(i) = HMAC_hash(secret, A(i-1))

TLS PRF定义

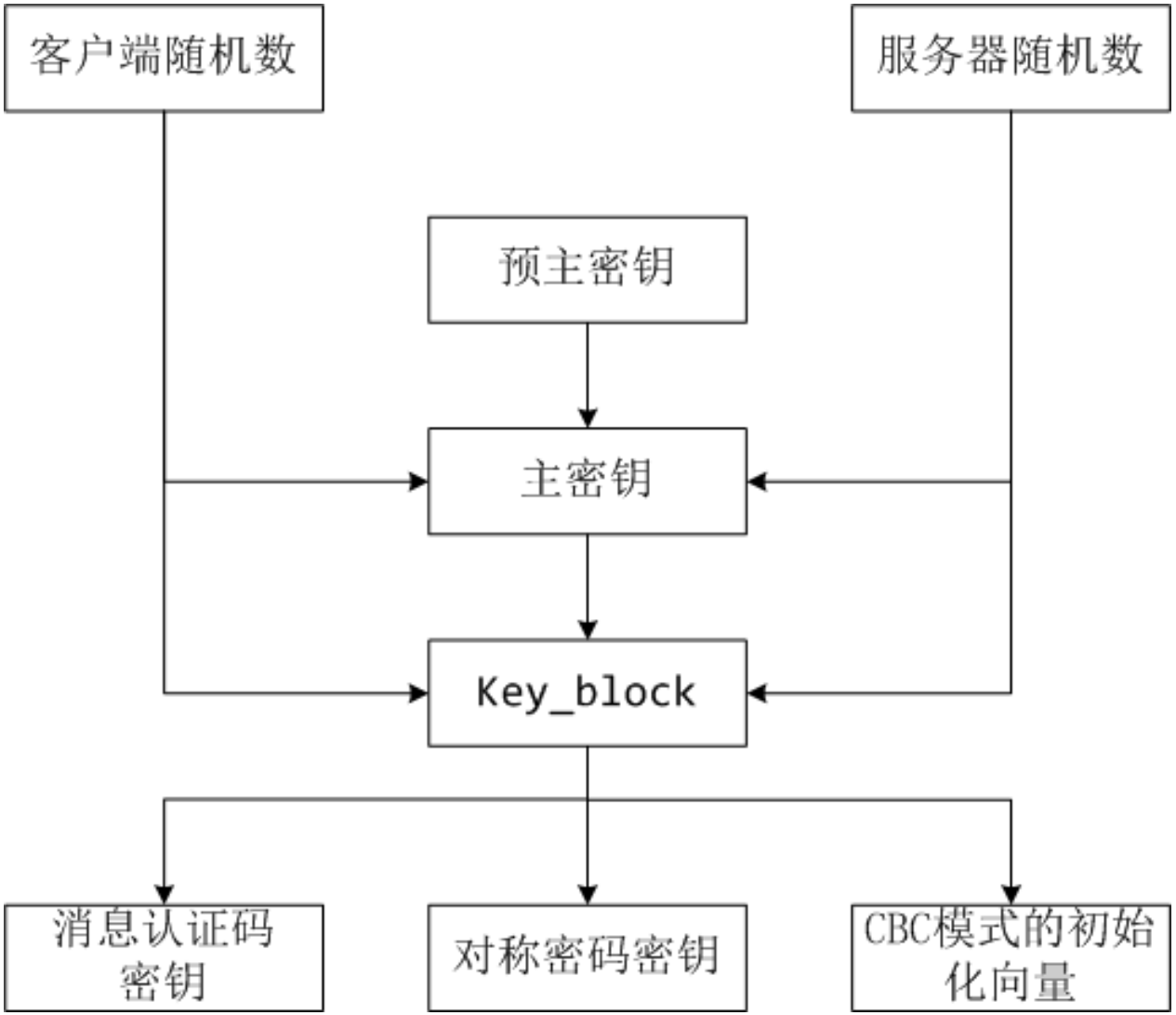
- P_hash是一个数据扩展函数，使用一个hash函数把一个秘密数（secret）和种子（seed）扩展成任意长度的输出。
- 如果需要，P_hash可以迭代足够多次，从而产生足够数量的数据。
  - 比如，如果P_SHA256用于产生80字节数据，则需要迭代3次（直至A(3)），产生96字节输出数据；最后一次迭代生成数据的最后16个字节是多余的，将被丢弃，而保留80字节数据。

密钥导出

- 最终通信双方产生相同的master secret，且进一步产生相同的keying material，通过相同顺序的切割，最后得到一致的6个密钥。

```
key_block =
PRF(SecurityParameters.master_secret,
                        "key expansion",

SecurityParameters.server_random +

SecurityParameters.client_random);
```

密钥导出


客户端随机数
服务器随机数
预主密钥
主密钥
Key_block
消息认证码
密钥
对称密码密钥
CBC模式的初始
化向量

恢复以前的session
Session Resumption

Reuse session information

- 完整的握手过程需要的开销较大
  - 消息传递（带来额外的时延）
  - 计算开销
- 能否在稍后的连接中重用已有的会话信息呢?
  - Ciphersuite
  - master secret
- opaque SessionID<0..32>;/*标识一个session*/
  - ClientHello
  - ServerHello

ClientHello消息格式

```
 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;
```

ServerHello消息格式

```
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;
```

握手过程: case 1

- 当客户端发送ClientHello消息时，其中的SessionID值可能是什么?
  - 为空（即：Session ID Length为0）
- 服务器端响应ServerHello消息，其中的SessionID值是什么?
  - 为空（即：Session ID Length为0）
    - 表示将来没有重用该会话信息的意愿
  - 为一个服务器生成值（一般为一个随机数）
    - 标识当前会话，表示将来可以重用该会话信息
    - 客户端会保存该SessionID及与该会话相关的信息

后续，走完整的握手流程!

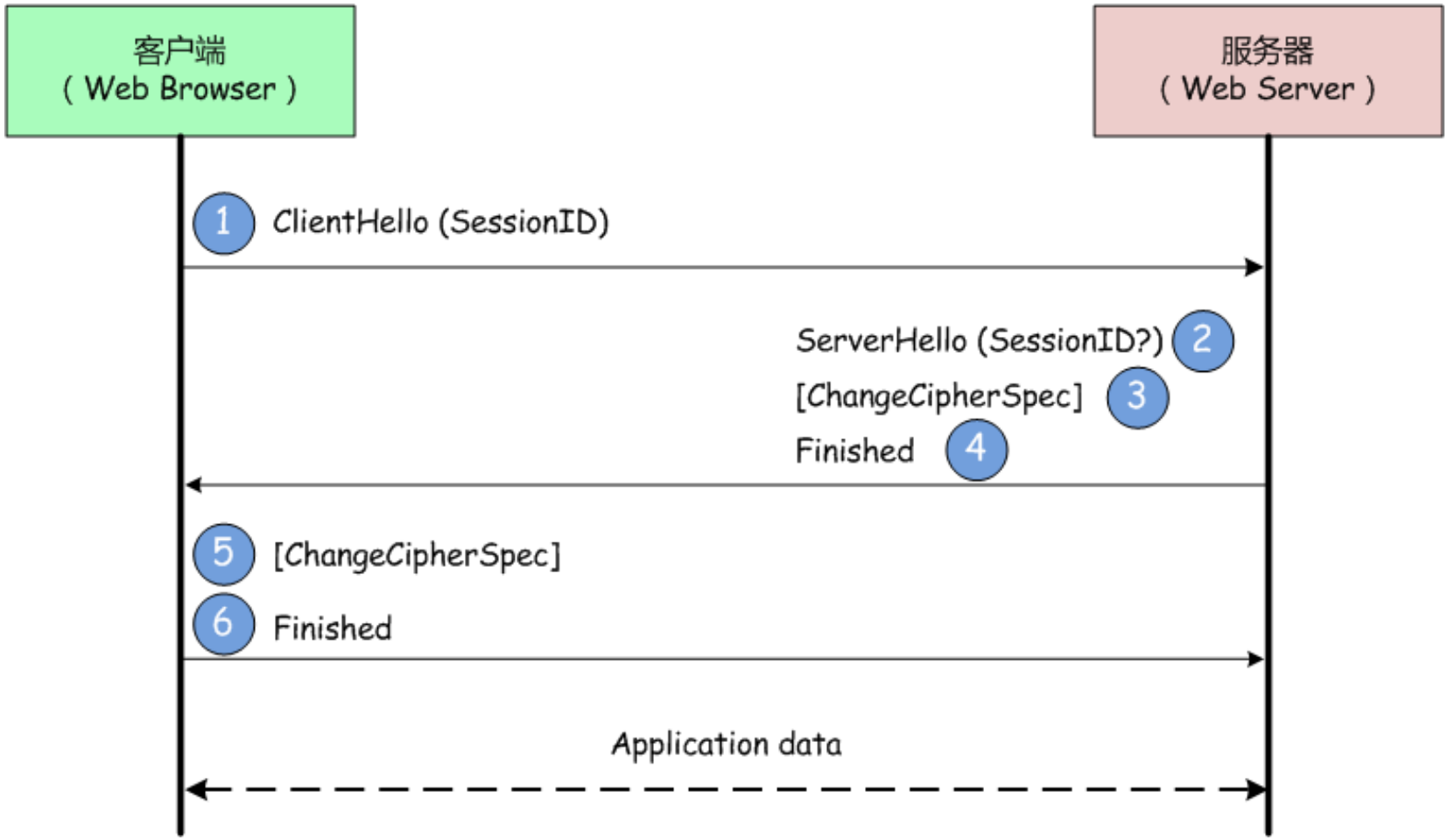
握手过程: case 2

- 当客户端发送ClientHello消息时，其中的SessionID值可能是什么?
  - 不为空（即：以前保存的SessionID）
- 服务器端响应ServerHello消息，其中的SessionID值是什么?
  - 为空（即：Session ID Length为0）
    - 表示没有找到匹配的项或者没有重用该会话信息的意愿
    - 后续，走完整的握手流程
  - 不为空（与ClientHello相同的SessionID）
    - 服务器收到ClientHello消息后，检查Session cache中是否有与ClientHello中的SessionID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。

恢复以前的session

- 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。

恢复以前session的消息流（abbreviated）



Weaknesses of SessionID

- 为每个客户端创建和维护一个session cache
  - 服务器需要大量的存储空间
- 服务器负载均衡
  - 多个服务器之间共享session cache是个难题

SessionID机制给服务器带来过多的开销！

解决方案：Session Ticket机制

Session ticket

- 什么是Session Ticket?
- ticket指的是一个由服务器创建和使用的且受密码学保护的数据结构，服务器使用ticket来重建与特定会话相关的状态。

参考RFC5077

Session ticket格式

```
struct {
        opaque key_name[16];
        opaque iv[16];
        opaque encrypted_state<0..2^16-1>;
        opaque mac[32];
    } ticket;
```

Session ticket：字段

- key_name
  - serves to identify a particular set of keys used to protect the ticket.
- encrypted_state
  - 128-bit AES in CBC mode with the given IV
- mac
  - HMAC-SHA-256
  - key_name (16 octets)
  - IV (16 octets)
  - Length of encrypted_state field
  - encrypted_state

Session ticket: StatePlaintext

```
struct {
      ProtocolVersion protocol_version;
      CipherSuite cipher_suite;
      CompressionMethod compression_method;
      opaque master_secret[48];
      ClientIdentity client_identity;
      uint32 timestamp;
    } StatePlaintext;
```

Session ticket的创建

- TLS服务器把会话相关的信息构造为一个session ticket并发送给客户端存储。
- 如果客户端要恢复一个session，它就发送session ticket给服务器，然后服务器解密验证ticket并根据里面存储的session状态信息来恢复一个会话。
- session ticket是服务器加密且认证的，因此在使用session ticket之前，服务器可以验证session ticket的有效性。

问题讨论

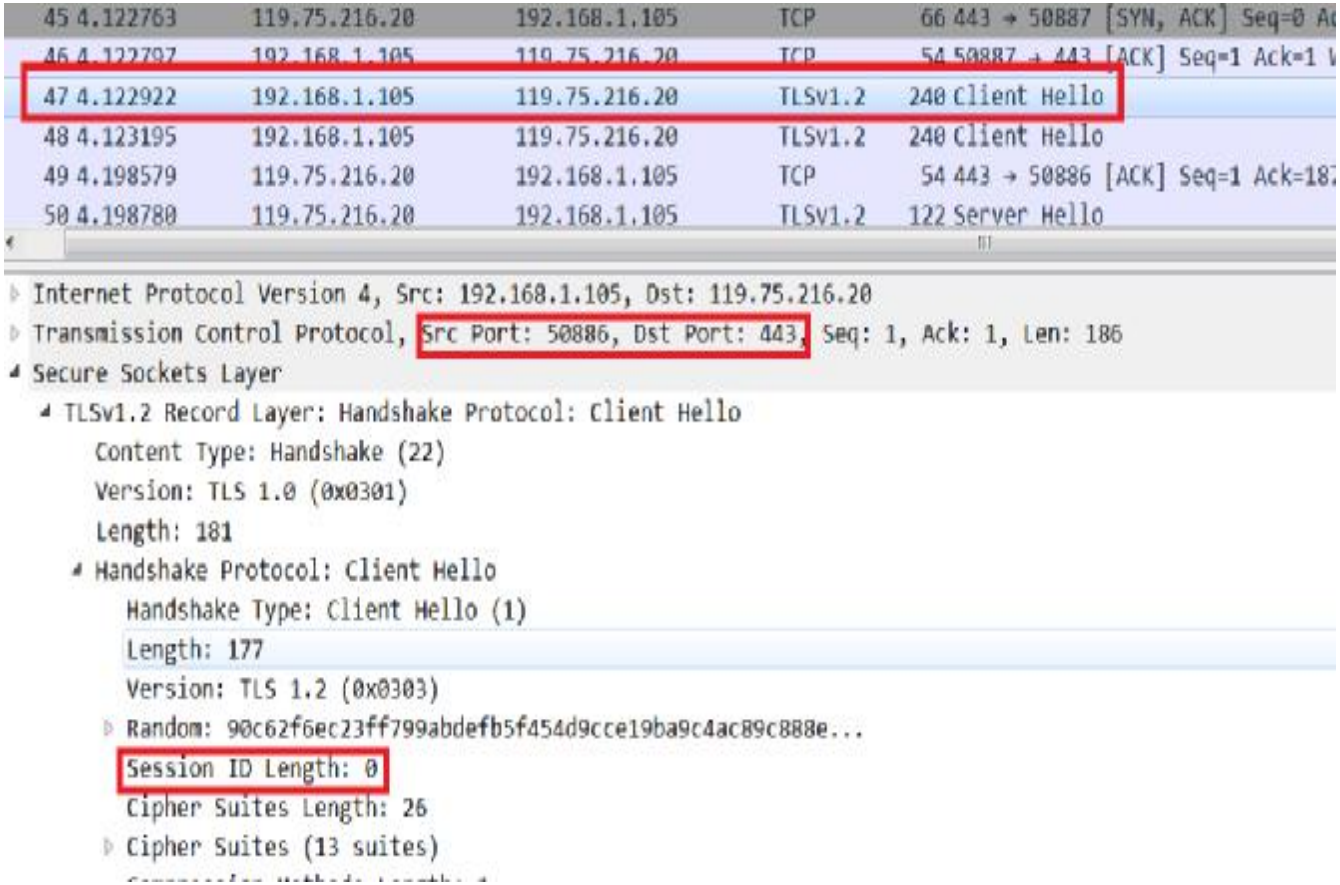
1. 攻击者（MiTM）能否利用偷取的session ticket?
2. 攻击者能否伪造session ticket?

Session ticket使用

- 客户端通过在ClientHello消息里面包含一个空的SessionTicket扩展来表示它支持session ticket机制。
- 服务器发送一个空的SessionTicket扩展给客户端，表示它支持Session Ticket机制，后面会使用NewSessionTicket握手消息发送一个Session Ticket给客户端（保存）。
- 客户端可以使用存储的session ticket来恢复会话。

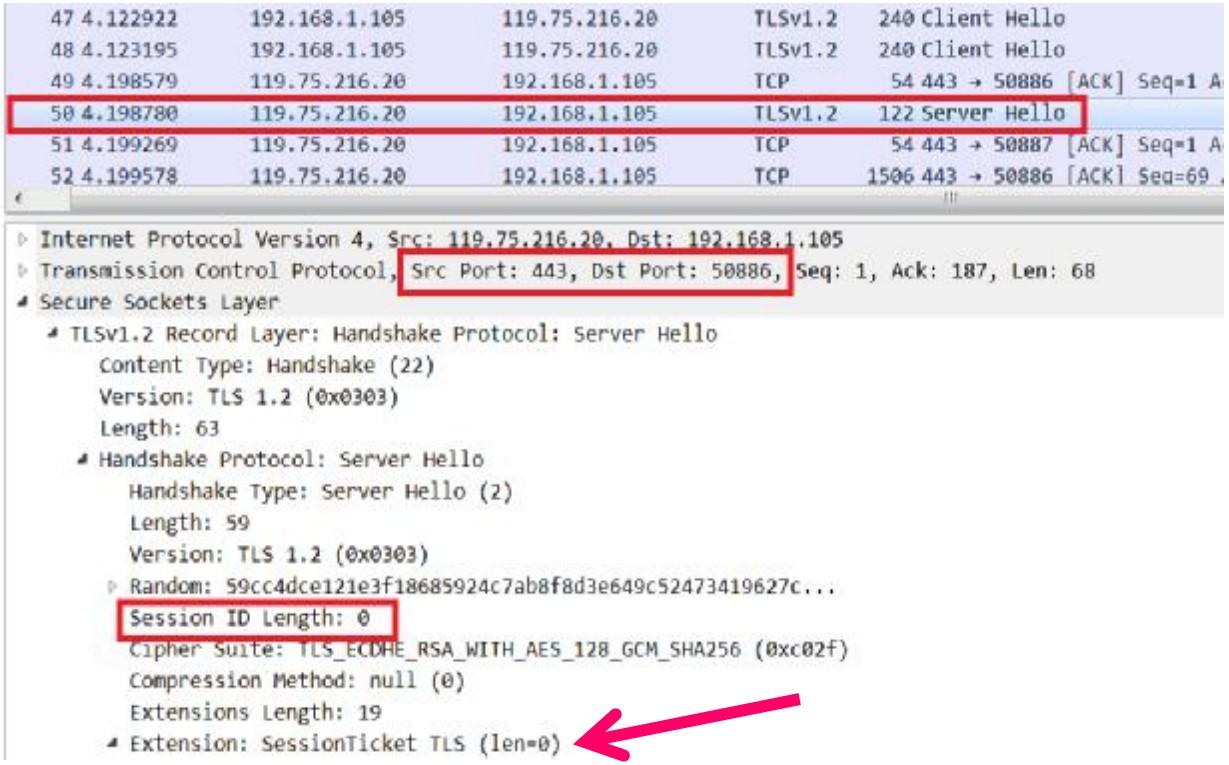
Session Ticket恢复会话过程

Session ticket使用



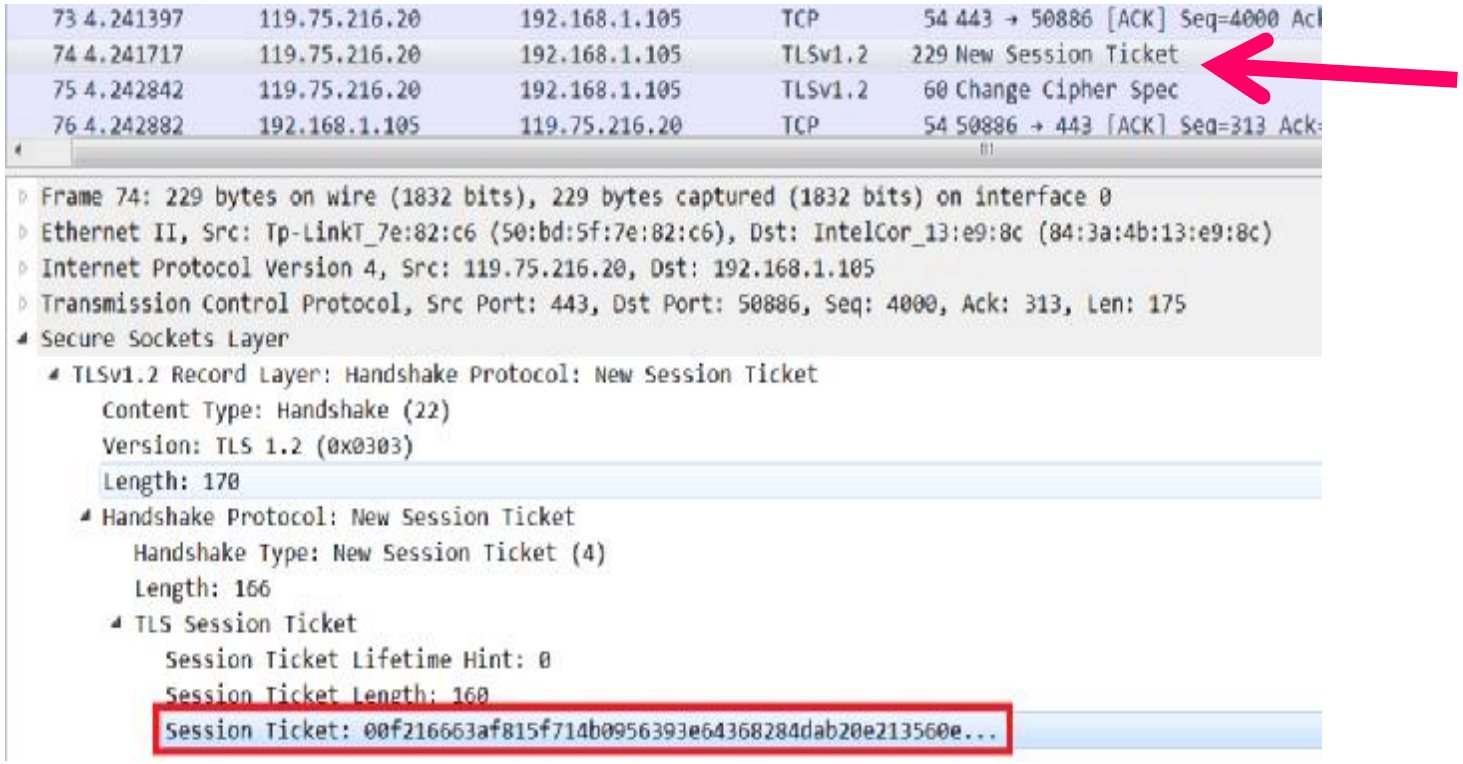
ClientHello： Session ID为空； 有SessionTicket extension，但是为空，表示支持SessionTicket机制。

Session ticket使用



ServerHello： Session ID为空；有SessionTicket extension，但是为空，表示支持SessionTicket机制。后面会发送NewSessionTicket握手消息

Session ticket使用

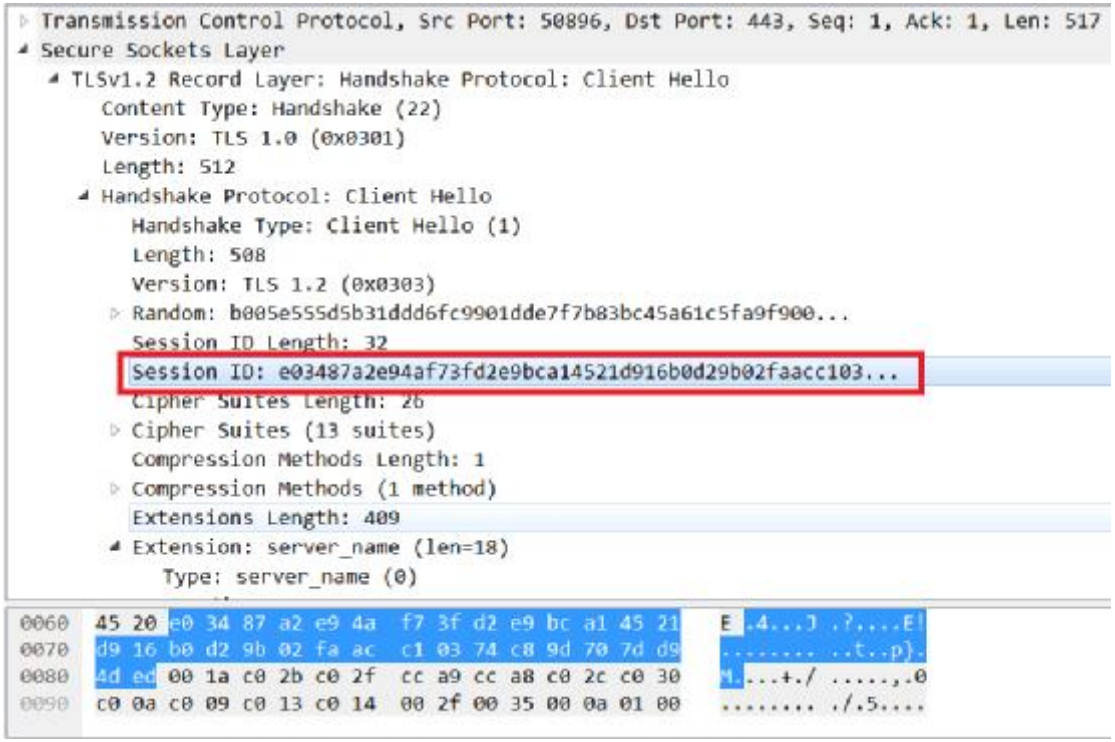


服务器发送的New Session Ticket，通过该握手消息给客户端发送一个ticket，即客户端存储ticket。

注意ticket的值，后面客户端是发送同样的ticket给服务器来恢复会话的。

随后不久，再次
访问同一网站

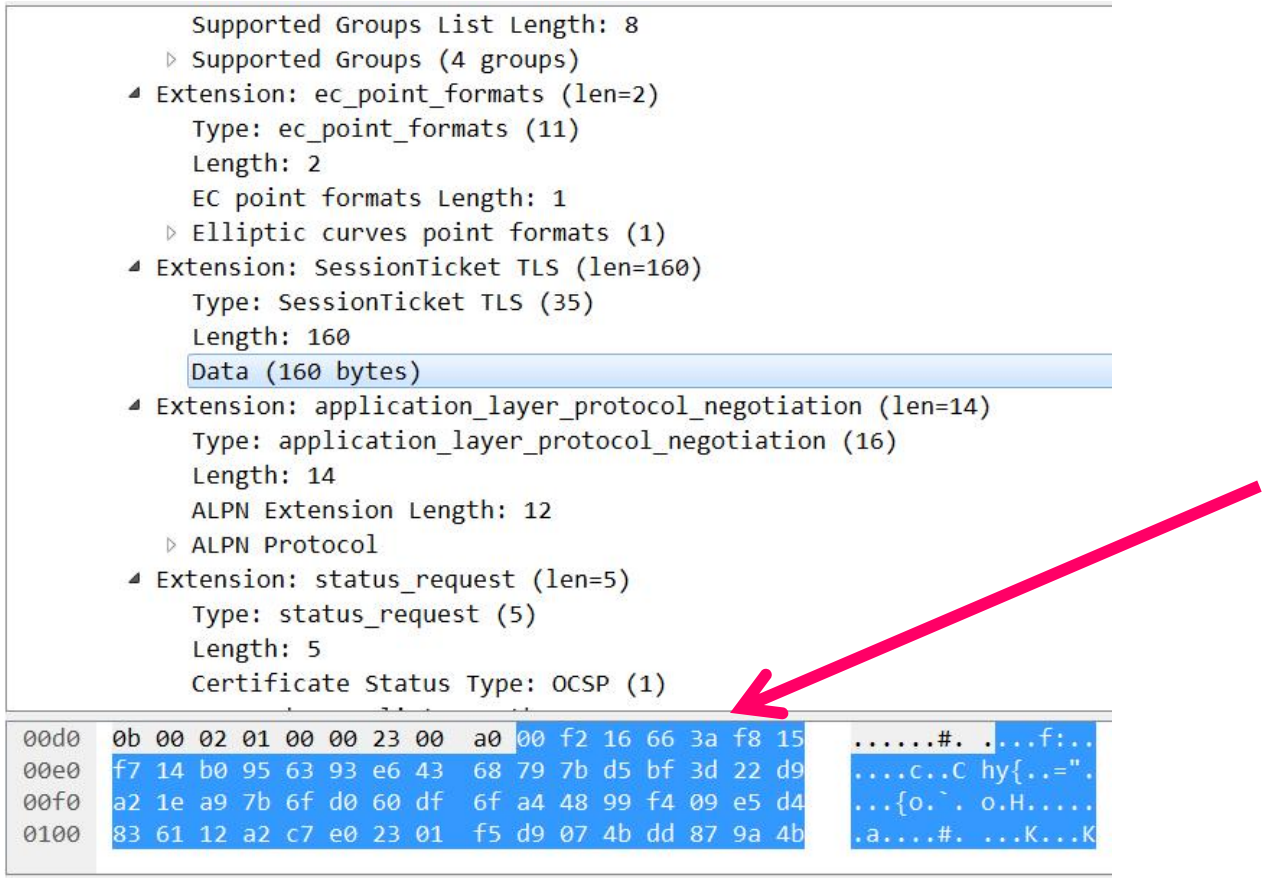
Session ticket使用：第二次ClientHello



ClientHello：其中的SessionID不为空！ServerHello包含同样的SessionID，客户端通过这种方式区分服务器是在恢复一个会话还是在进行一次完整的握手。

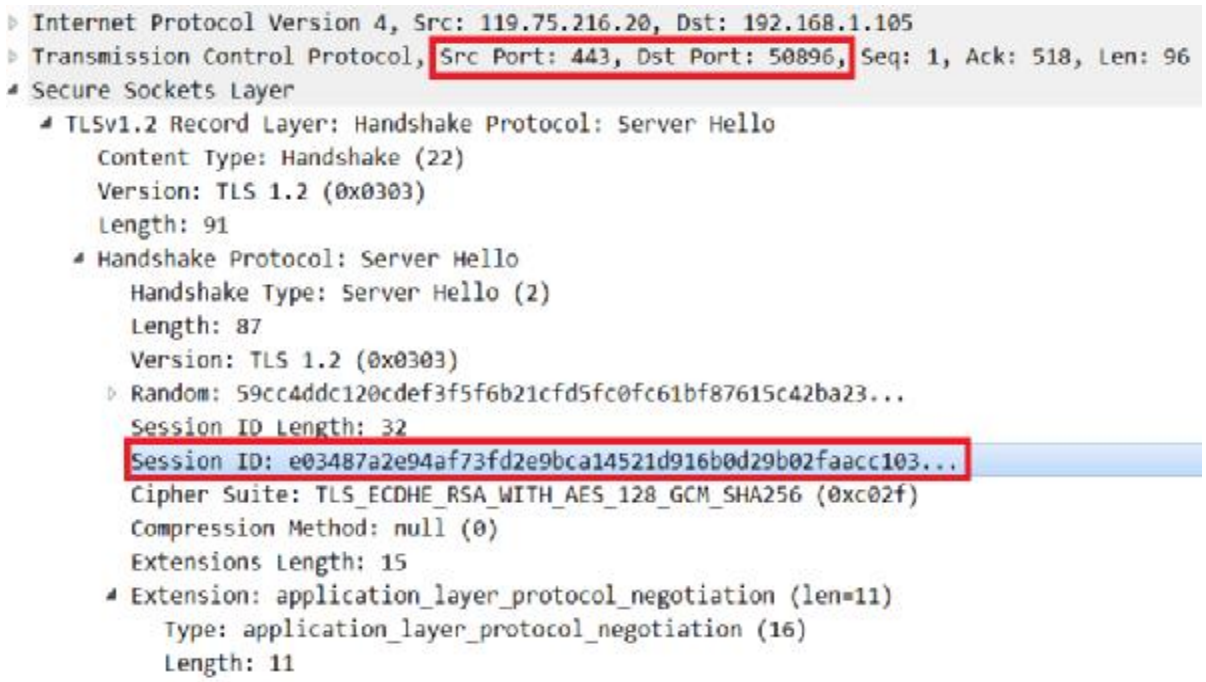
下图是ClientHello里面的SessionTicket extension

Session ticket使用



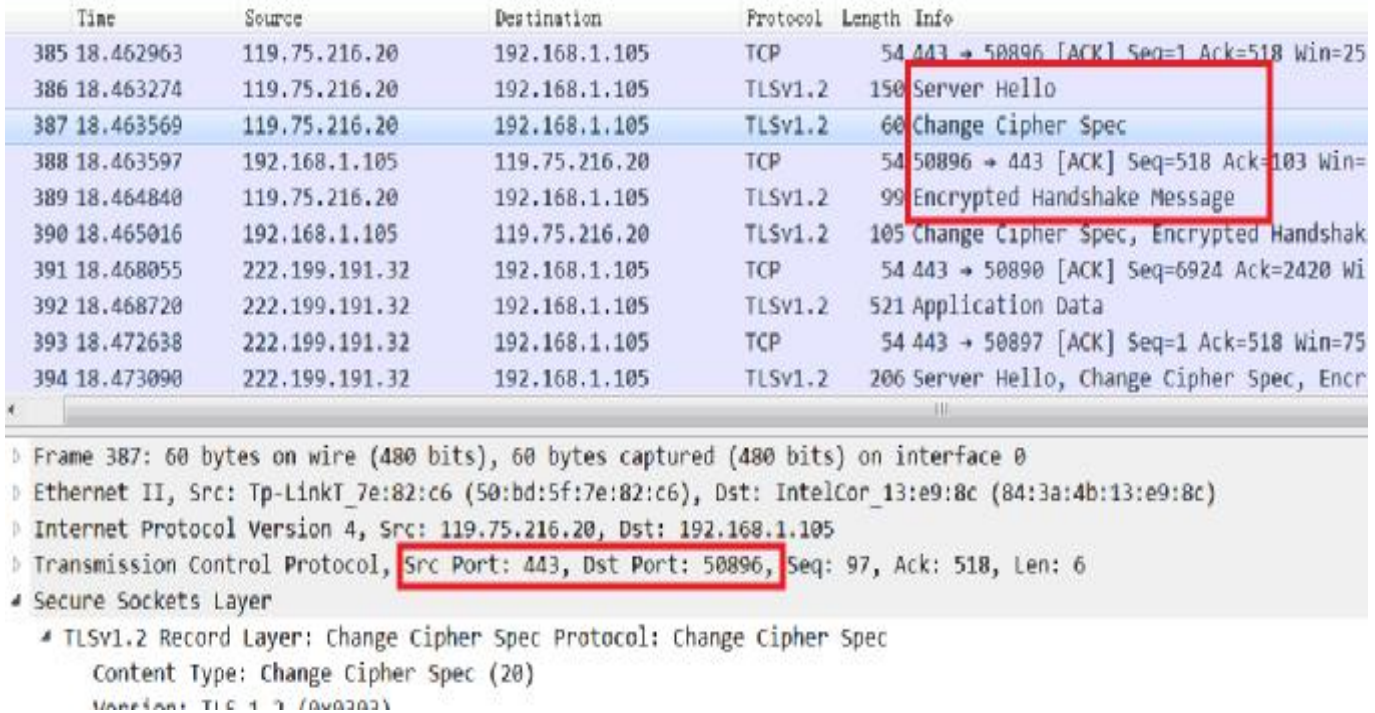
ClientHello：Session Ticket extension，注意其中的data，即为加密的ticket，对比前面服务器发送的ticket

Session ticket使用



ServerHello： Session ID和ClientHello里面的一样。

Session ticket使用



	Time	Source	Destination	Protocol	Length	Info
385	18.462963	119.75.216.20	192.168.1.105	TCP	54	443 → 50896 [ACK] Seq=1 Ack=518 Win=25
386	18.463274	119.75.216.20	192.168.1.105	TLSv1.2	150	Server Hello
387	18.463569	119.75.216.20	192.168.1.105	TLSv1.2	60	Change Cipher Spec
388	18.463597	192.168.1.105	119.75.216.20	TCP	54	50896 → 443 [ACK] Seq=518 Ack=103 Win=
389	18.464840	119.75.216.20	192.168.1.105	TLSv1.2	99	Encrypted Handshake Message
390	18.465016	192.168.1.105	119.75.216.20	TLSv1.2	105	Change Cipher Spec, Encrypted Handshak
391	18.468055	222.199.191.32	192.168.1.105	TCP	54	443 → 50890 [ACK] Seq=6924 Ack=2420 Wi
392	18.468720	222.199.191.32	192.168.1.105	TLSv1.2	521	Application Data
393	18.472638	222.199.191.32	192.168.1.105	TCP	54	443 → 50897 [ACK] Seq=1 Ack=518 Win=75
394	18.473090	222.199.191.32	192.168.1.105	TLSv1.2	206	Server Hello, Change Cipher Spec, Encr

```
▷ Frame 387: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0
▷ Ethernet II, Src: Tp-LinkT_7e:82:c6 (50:bd:5f:7e:82:c6), Dst: IntelCor_13:e9:8c (84:3a:4b:13:e9:8c)
▷ Internet Protocol Version 4, Src: 119.75.216.20, Dst: 192.168.1.105
▷ Transmission Control Protocol, Src Port: 443, Dst Port: 50896, Seq: 97, Ack: 518, Len: 6
◢ Secure Sockets Layer
  ◢ TLSv1.2 Record Layer: Change Cipher Spec Protocol: Change Cipher Spec
      Content Type: Change Cipher Spec (20)
```

ServerHello消息后是ChangeCipherSpec消息，然后是加密的Finished消息。

表明：上述是一个简单的握手过程，恢复以前的会话。

TLS的应用

TLS 的应用

- 分设端口

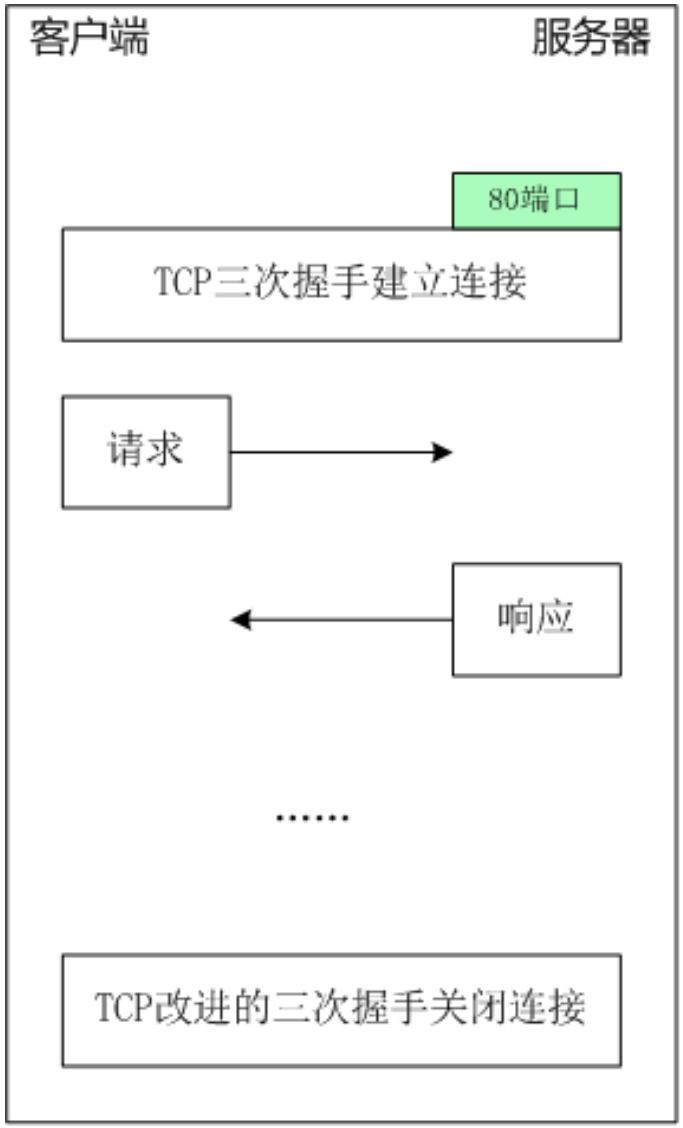
普通协议	普通协议端口	安全协议	安全协议端口	描述
HTTP	80	HTTPS	443	超文本传输协议
FTP	21	FTPS	990	文件传输协议
Telnet	23	Telnets	992	远程登录协议

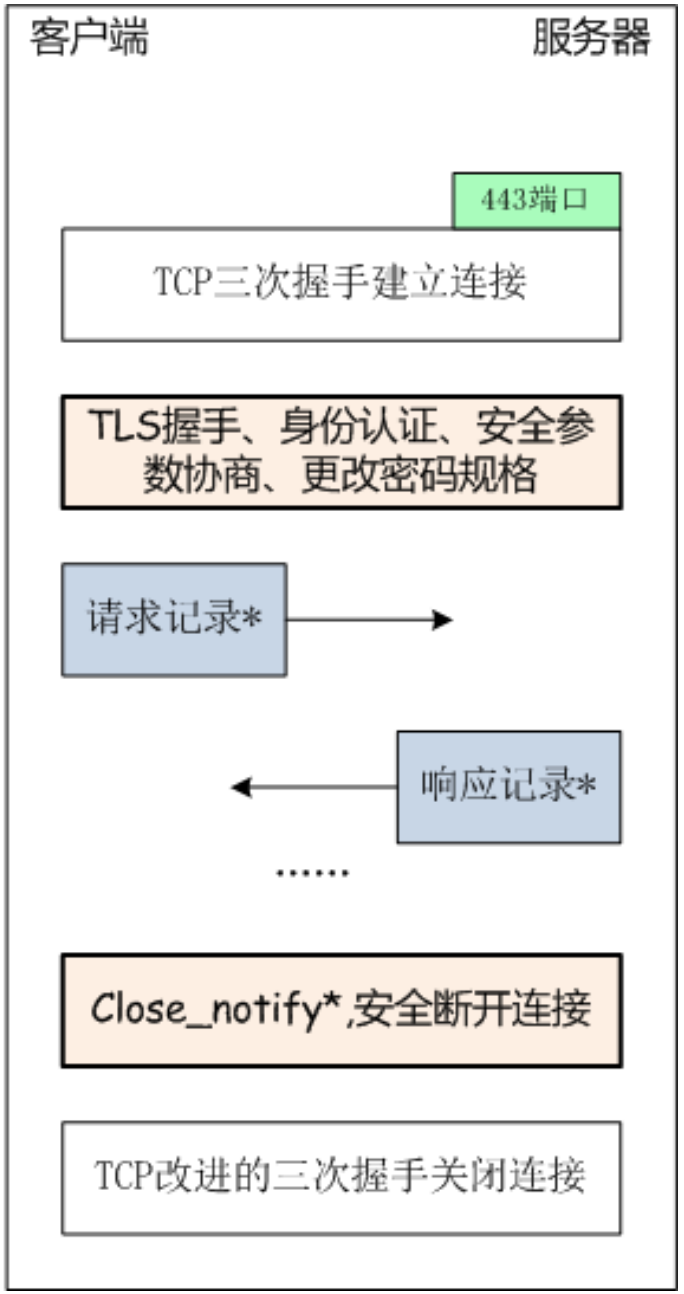
TLS 的应用

- 向上协商
  - 通过修改应用层协议，基于应用层协议来协商是否采用TLS协议而不是采用端口分设来决定是否采用TLS协议

HTTPS
(HTTP over TLS)

HTTP 与 HTTPS 的流程对比

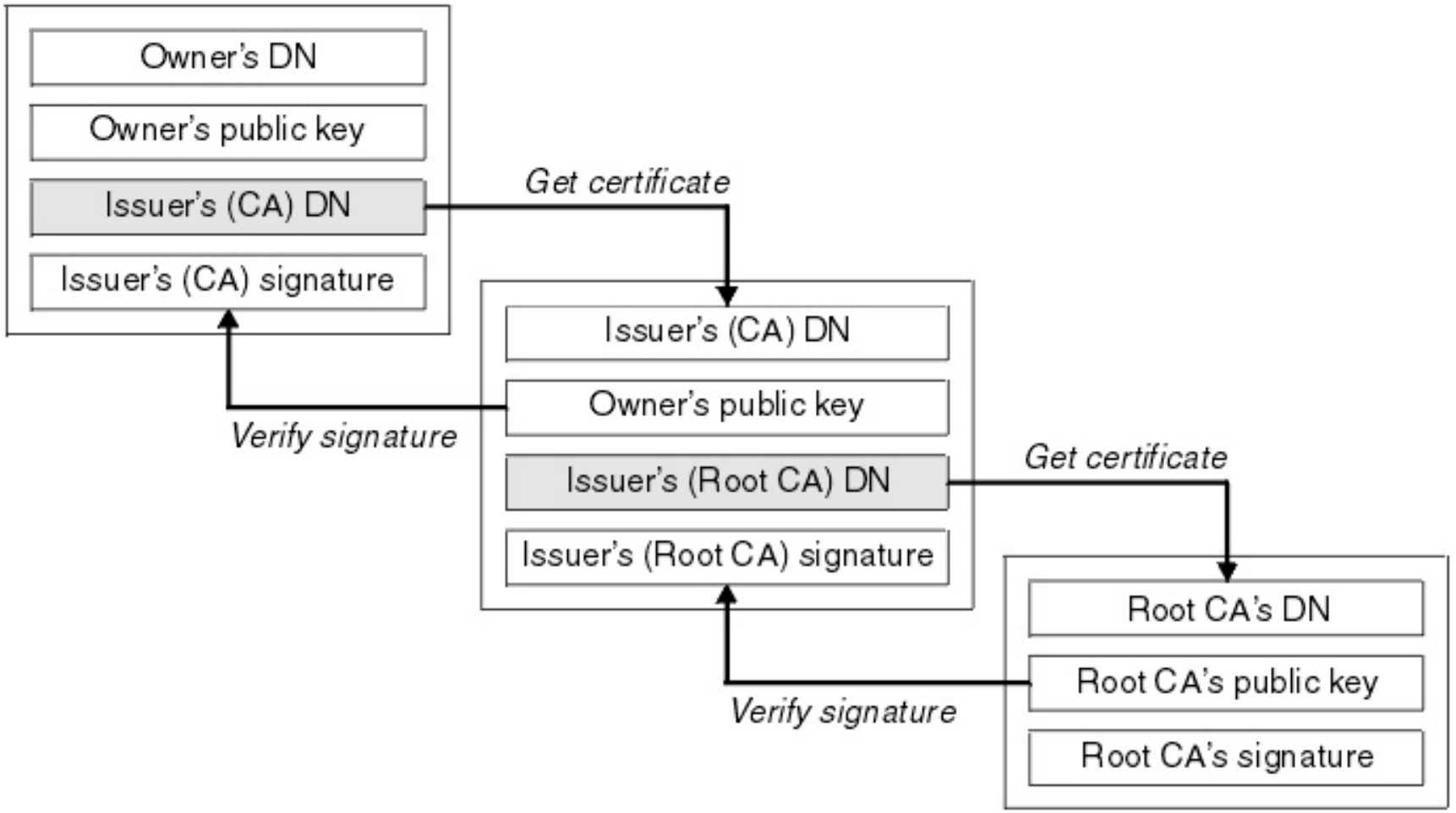




问题讨论?

- 在企业网络的入口节点（middlebox），如何能够对https流量进行DPI？

Certificate Chain



可信任证书库

浏览器平台	Windows	Unix
Chrome	使用 windows 可信任根证书库	使用 NSS 可信任根证书库
Firefox	使用 NSS 可信任根证书库	使用 NSS 可信任根证书库

根证书库存放位置

微软Windows系统：

- CERT_SYSTEM_STORE_CURRENT_USER，代表当前登陆用户，对应的证书存储在 HKEY_CURRENT_USER\Software\Microsoft\SystemCertificates 目录。
- CERT_SYSTEM_STORE_LOCAL_MACHINE，代表本机，对应的证书存储在 HKEY_LOCAL_MACHINE\Software\Microsoft\SystemCertificates 目录。
- CERT_SYSTEM_CURRENT_USER_GROUP_POLICY，代表当前用户组策略，对应的证书存储在 HKEY_CURRENT_USER\Software\Policies\Microsoft\SystemCertificates 目录。

NSS根证书库：

Cert9.db-----SQLite数据库

Cert8.db

如何测试分析基于TLS的应用?

- Charles Proxy
  - https://www.charlesproxy.com/
- mitmproxy
  - https://mitmproxy.org/
- Fiddler, Burpsuite……

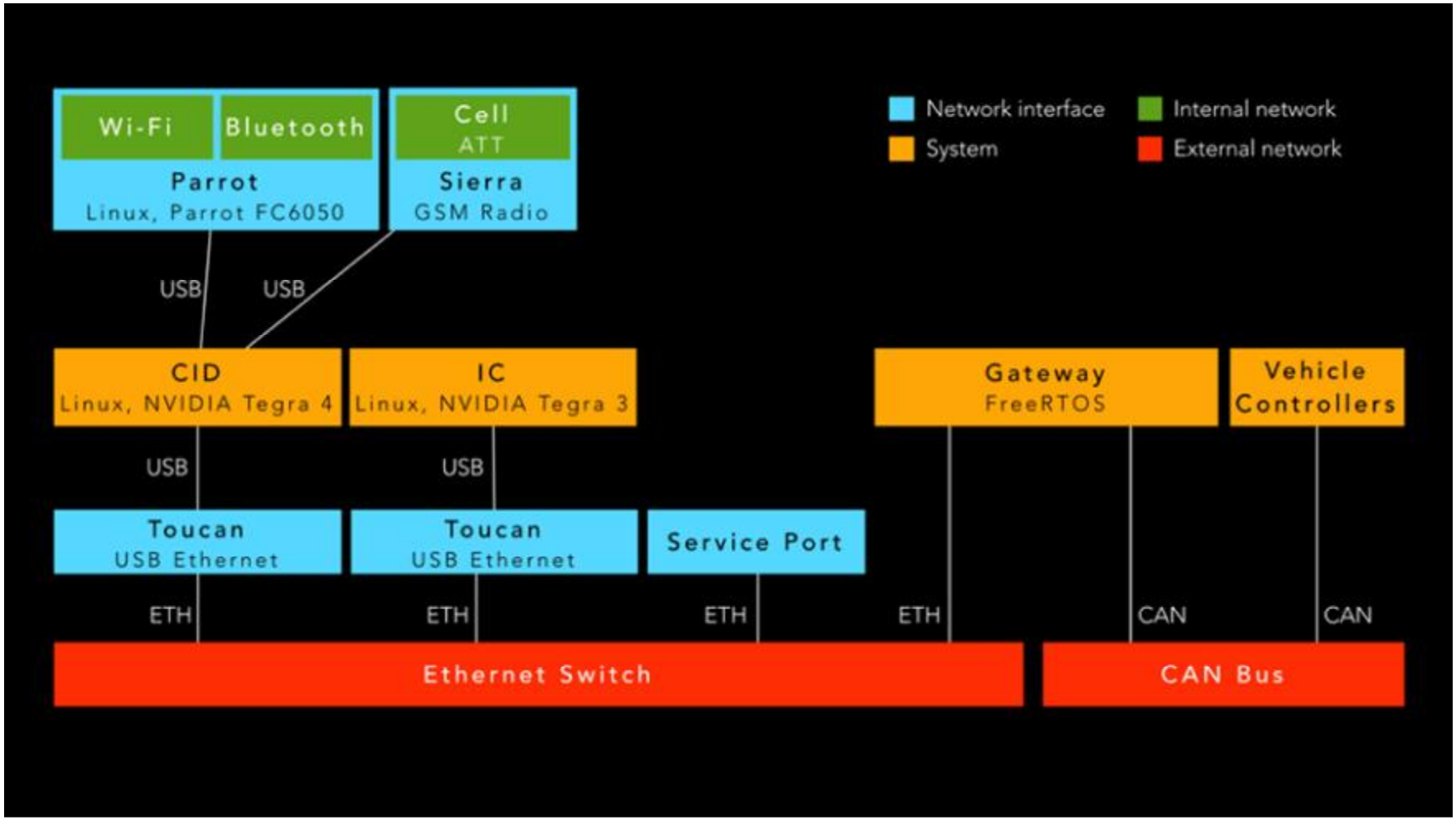
课后作业

- 编写程序来获取https服务器的密码套件和证书等配置信息，可以进一步考虑大规模扫描https服务器的程序架构和实现。
- 可参考：https://github.com/prbinu/tls-scan

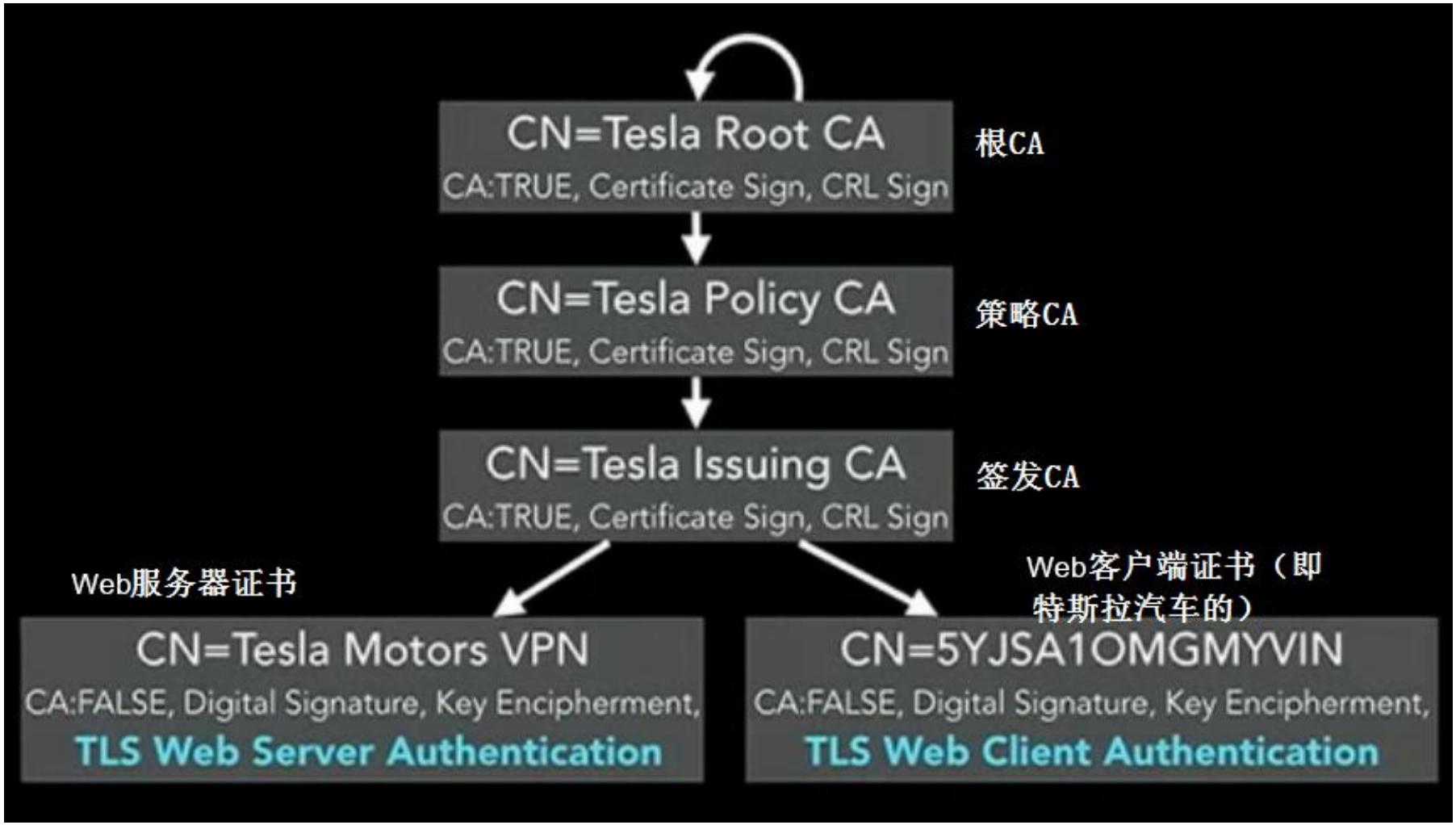
OSINT

HACKING TESLA

内部物理连接


Wi-Fi
Bluetooth
Parrot
Linux, Parrot FC6050
Cell
ATT
Sierra
GSM Radio
Network interface
Internal network
System
External network
USB
USB
CID
Linux, NVIDIA Tegra 4
IC
Linux, NVIDIA Tegra 3
Gateway
FreeRTOS
Vehicle
Controllers
USB
USB
Toucan
USB Ethernet
Toucan
USB Ethernet
Service Port
ETH
ETH
ETH
ETH
CAN
CAN
Ethernet Switch
CAN Bus

TESLA数字证书架构



双向身份认证

证书，私钥

攻击思路


CN=5YJSA1OMGMYVIN
CA:FALSE, Digital Signature, Key Encipherment,
TLS Web Client Authentication
正常通信链路
CN=Tesla Motors VPN
CA:FALSE, Digital Signature, Key Encipherment,
TLS Web Server Authentication
服务器
CN=5YJSA1OMGMYVIN
CA:FALSE, Digital Signature, Key Encipherment,
TLS Web Client Authentication
攻击者伪造服务器

攻击思路


Car 汽车
FauxpenVPN
冒牌服务器
Client Hello
Server Hello
Certificate Request
Certificate=5YJSA1OMGMYVIN
返回从车里提取的客户端证书

攻击失败

- TESLA采用x509v3证书，其中EKU（Extended Key Usage是个扩展标准，指定了公钥用途）规定了一个证书中的公钥的用途，即：一个证书只能被用于特定的目的。比如VPN服务器的证书就只能用于服务器认证，而CarKey的证书只能被用于客户端认证。OpenVPN中可以设定是否要对EKU中指定的密钥用途进行验证。