第六讲 从SSL 1.0 到 TLS 1.3

汪　定

南开大学 网络空间安全学院

wangding@nankai.edu.cn

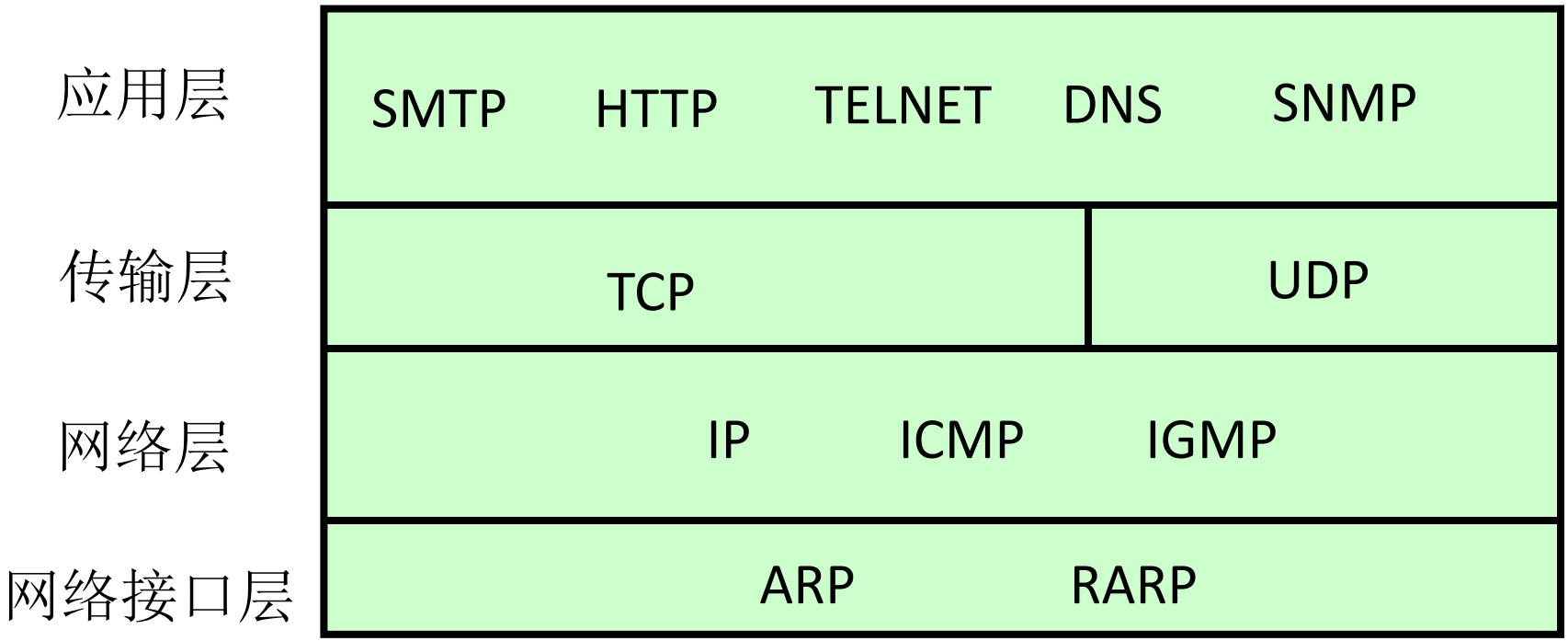
2021年12月11日

提 纲

CONTENTS

1. 传输层安全的发展历史
2. SSL3.0/TLS1.0
3. TLS1.2
4. TLS1.3
5. 总结

回顾：TCP/IP协议栈

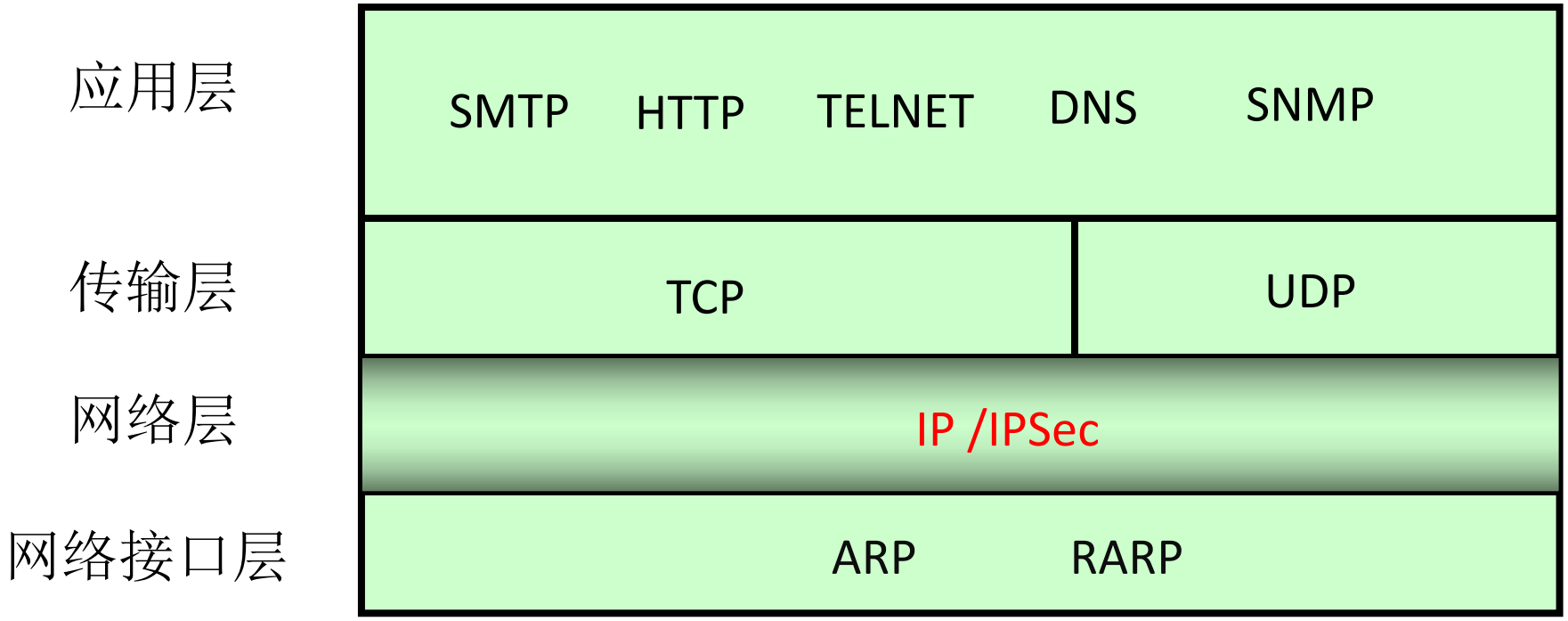

应用层
SMTP
HTTP
TELNET
DNS
SNMP
传输层
TCP
UDP
网络层
IP
ICMP
IGMP
网络接口层
ARP
RARP

网络安全协议栈

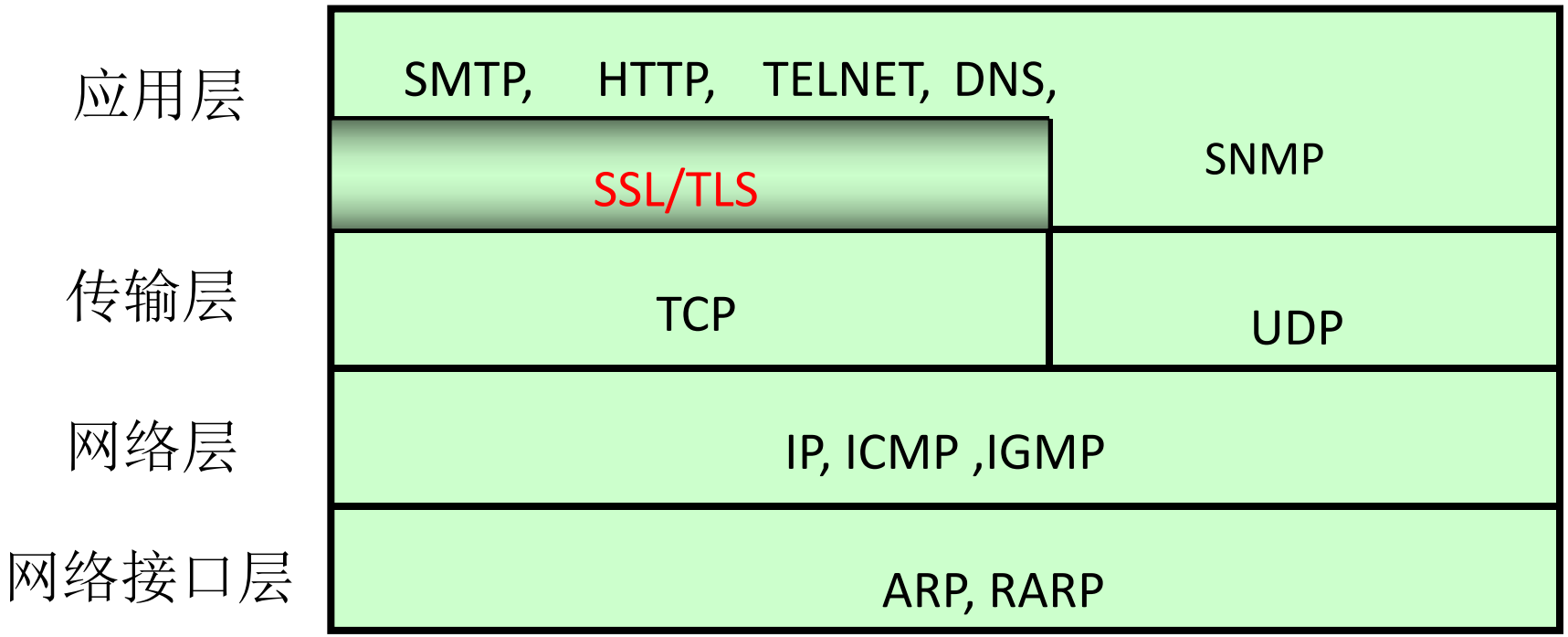
Communication layers	Security protocols
Application layer	HTTPS, SSH, S/MIME, PGP, Kerberos，SRP, EKE, SPEKE, HMQV, OPAQUE,…
Transport layer	TLS (1999-至今）, SSL（1994-1998）
Network layer	IPSec
Data Link layer	[PPTP, L2TP], IEEE 802.1X, IEEE 802.1AE, IEEE 802.11i (WPA2)
Physical layer	Quantum Cryptography

TCP/IP的安全性问题及其解决方法

网络层安全

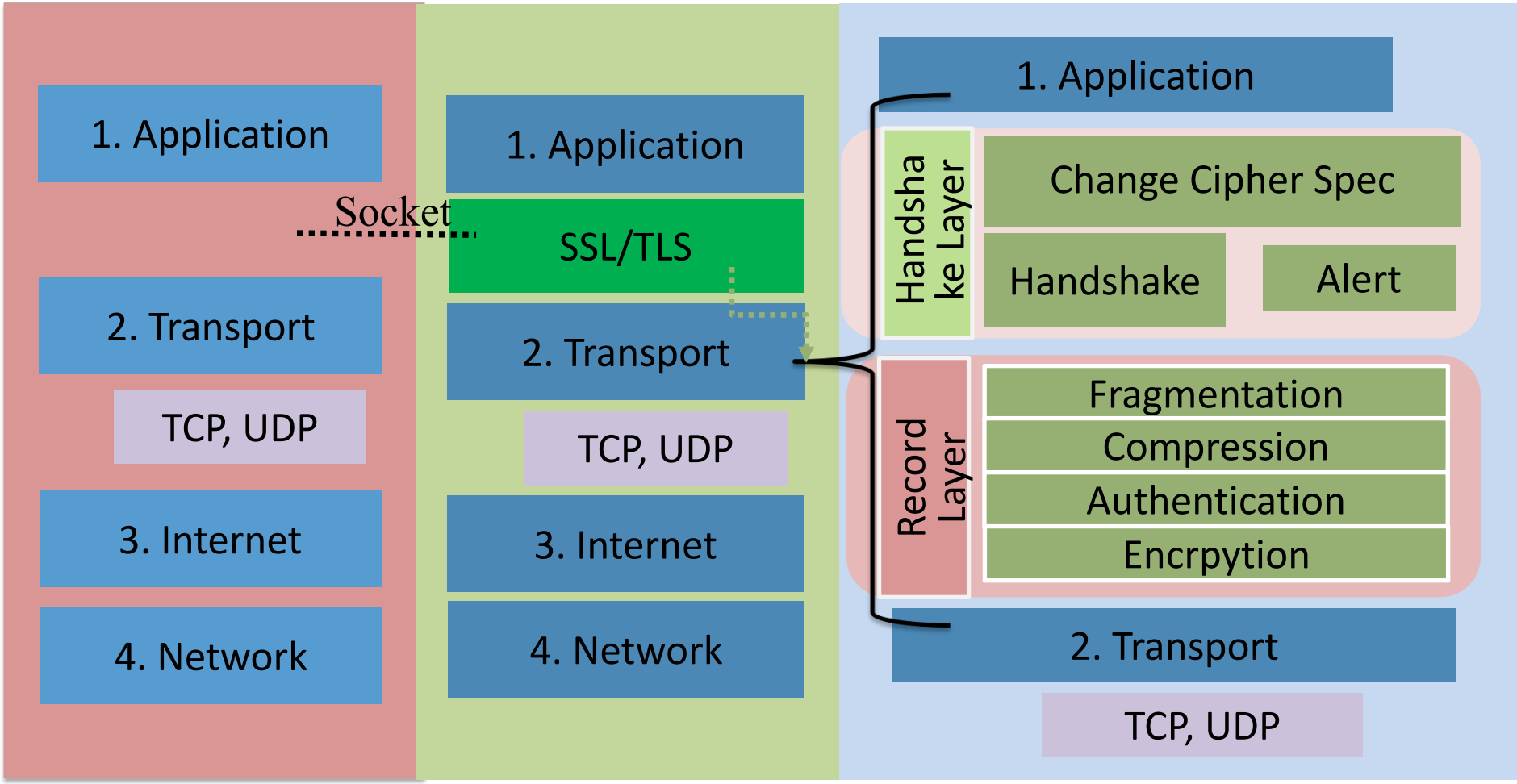


传输层安全

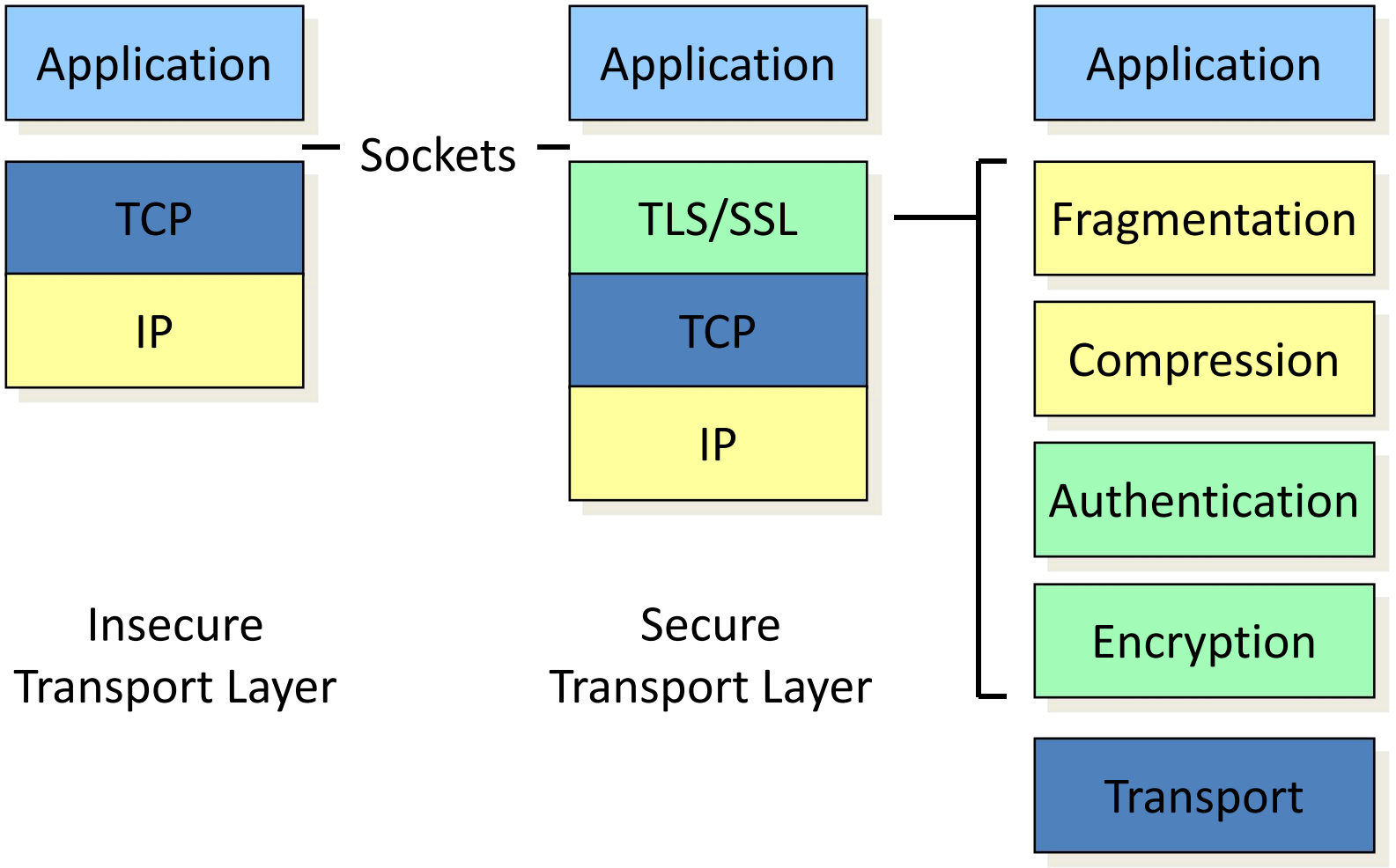


SSL: Secure Sockets Layer
TLS: Transport Layer Security

SSL/TLS Protocol



TLS/SSL Protocol Layers



传输层安全的历史

- **1994年初Netscape公司内部开发SSL 1.0**
  - Lost in the mists of time
- **1994年底Netscape 公司开发 SSL 2.0和3.0**
  - SSL v2 released in 1995
  - SSL v3 also released in 1995 due to bugs in v2
- **1996年IETF成立工作组**
  - Transport Layer Security (TLS) committee
- **1999年推出TLS 1.0**
  - Based on SSL 3.0, but not interoperable (uses different cryptographic algorithms)
  - http://www.ietf.org/rfc/rfc2246.txt
- **2003年推出TLS 扩展**

传输层安全的历史

- **2006年推出TLS 1.1**
  - Some protection against CBC-mode attacks: explicit IV, better padding
  - http://www.ietf.org/rfc/rfc4346.txt
- **2008年推出TLS 1.2**
  - First collusion attacks on MD5 in 2005; Certificate forging in 2008: finally, TLS changes to SHA 1 and SHA-256; Adds AES-GCM
  - http://www.ietf.org/rfc/rfc5246.txt
- **2018年推出TLS 1.3**
  - http://www.ietf.org/rfc/rfc8446.txt

为什么需要TLS1.3

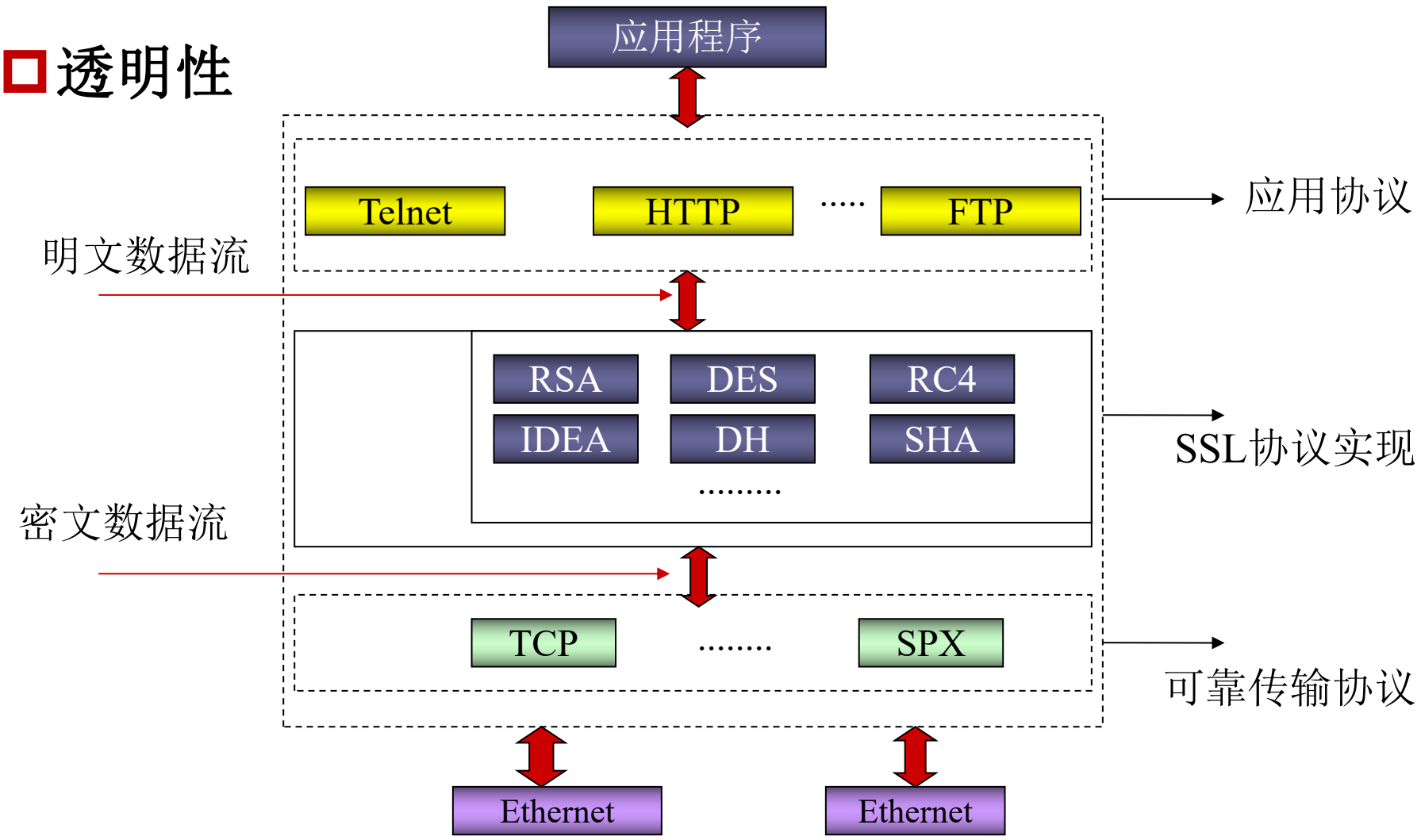
- 2014年启动修订，2018年8月正式推出TLS1.3
- 安全考量
  - RSA 密钥传输 —— 不支持前向安全性
  - CBC 模式密码 —— 易受 BEAST 和 Lucky 13 攻击
  - RC4 流密码 —— 在 HTTPS 中使用并不安全
  - SHA-1 哈希函数 —— 建议以 SHA-2 取而代之
  - 任意 Diffie-Hellman 组—— CVE-2016-0701 漏洞
  - 输出密码 —— 易受 FREAK 和 LogJam 攻击
- 效率考量
  - 模幂运算慢 → 仅支持ECC
  - 握手需要2轮（四次交互） →握手需要1轮

SSL的目标

- 加密功能
  - SSL被设计用来使用TCP提供一个可靠的端到端安全服务
  - 为两个通信实体之间提供
    - 保密性
    - 完整性
    - 身份认证
    - 密钥交换
- 可扩展性
  - 新的密钥算法可以容易的加入
- 高效性
  - 减少对CPU的使用
- 透明性
  - 对应用层透明

SSL的透明性

- 透明性

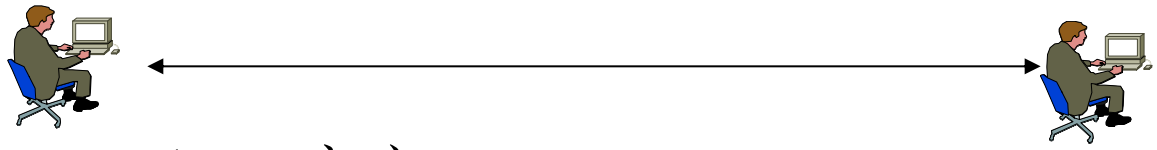

应用程序
Telnet
HTTP
.....
FTP
应用协议
明文数据流
RSA
DES
RC4
IDEA
DH
SHA
.....
SSL协议实现
密文数据流
TCP
.....
SPX
可靠传输协议
Ethernet
Ethernet

SSL的安全功能

- **Data Encryption:**
  - RC2-40
  - RC4-128
  - DES
  - DES 40
  - 3DES
  - IDEA
  - AES 128
  - Fortezza
- **Key Exchange:**
  - RSA
  - Fixed Diffie-Hellman
  - Ephemeral Diffie-Hellman
  - Anonymous Diffie-Hellman
  - Fortezza
- **Message Digest:**
  - MD5
  - SHA-1
- **Data Compression:**
  - PKZip
  - WinZip
  - gzip
  - ....

SSL中的通信实体

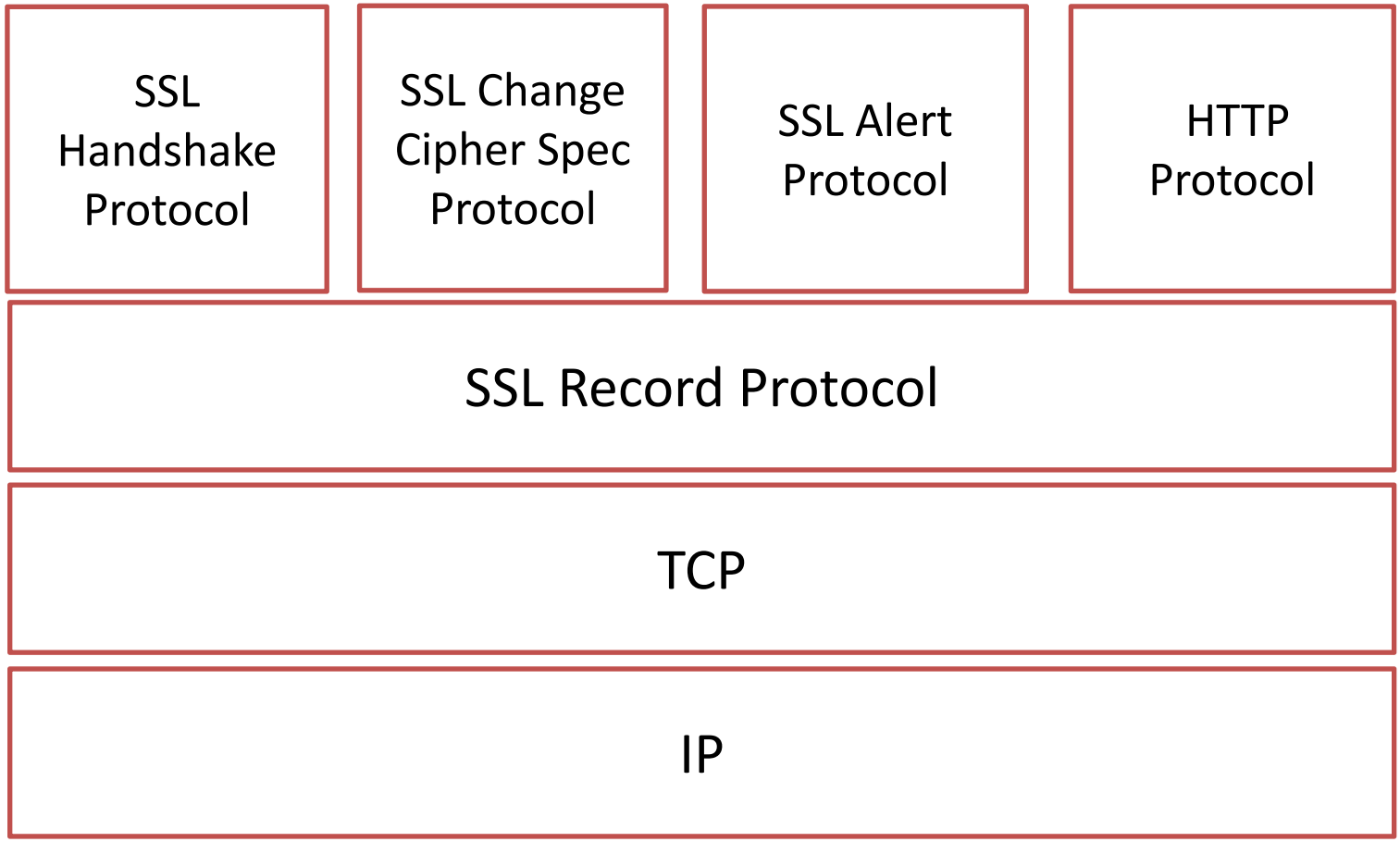
- 通信实体（entity）是指SSL的参与者
- 在SSL v3中定义了两个通信实体
  - 客户
    - 客户是协议的发起者
  - 服务器
    - 服务器是协议的响应者



SSL Client（SSL客户）

SSL Server（SSL服务器）

SSL 的协议分层



握手协议：用来实现密钥交换和认证　　加密规约修改协议：启用新的密参数

记录层协议：用来安全传输数据　　报警协议：报警和错误

一个问题

- Question：

  为什么在SSL中存在两层：握手层和记录层？

Key Terms of SSL

- 在SSL中，有几个关键概念：
  - SSL Session（SSL会话）
  - SSL Connection（SSL连接）
  - SSL Session State（SSL会话状态）
  - SSL Connection State（SSL连接状态）

SSL会话

- SSL session（会话）
  - An association between client & server
    （是客户和服务器之间的一个关联）
  - Created by the Handshake Protocol
    （通过握手协议来创建）
  - Define a set of cryptographic parameters
    （定义了一套加密参数）
  - May be shared by multiple SSL connections
    （可以被多个SSL连接共享）

SSL连接

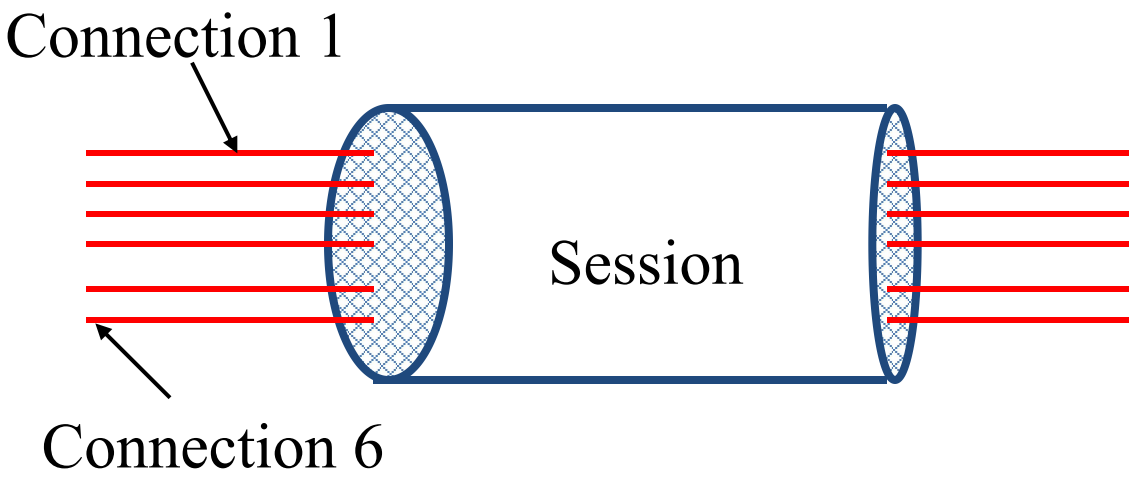
- SSL connection（连接）
  - A transient, peer-to-peer, communications link（是一种通信实体具有对等关系的通信连接）
  - Associated with one SSL session
    （与一个SSL Session关联）
  - 连接是瞬时的，用后即消失

SSL会话 vs. SSL连接

- 连接&会话的异同
  - 会话是用来协商安全参数的（如加密算法，hash函数等）
  - 连接是用来安全传输应用程序数据的（如加密传送消息）
  - 理论上，双方可以存在多个同时会话，但在实践中并未用到这个特性
  - 在任意一对通信双方之间，也许会有多个安全连接
  - 每个连接都只和一个会话相关
  - 一个会话可能包含多个安全连接
  - 会话定义了一组可以被多个连接共用的密码安全参数，对于每个连接，可以利用会话来避免对新的安全参数进行代价昂贵的协商

会话与连接的关系

- 连接&会话



SSL 会话与会话状态

- Session States(会话状态）：
  - 标识一个具体的SSL会话的信息
- 客户和服务器必须存储已经建立的会话及其相应会话状态的信息
- 会话状态信息供握手协议使用
  - 特别是恢复一个会话时（避免重新建立会话）

连接状态

- Connection States (连接状态）contains all the information in one specific connection；
- Only when the connection exists, the connection states will be remembered

SSL 会话和会话状态

- Information of Session States（内容）
  - Session Identifier (会话标识符）
  - The X.509 certificates of Peer Certificate Server (Client)（客户或服务器的X.509证书，如果不需要验证，则该信息为空）
  - Compression Methods （压缩算法）
  - Cipher Spec （握手协议已经协商的一套加密参数）
    - 对称加密算法
    - MAC算法
    - 加密属性（包括Hash长度等）

SSL Session & Session State（Cont.)

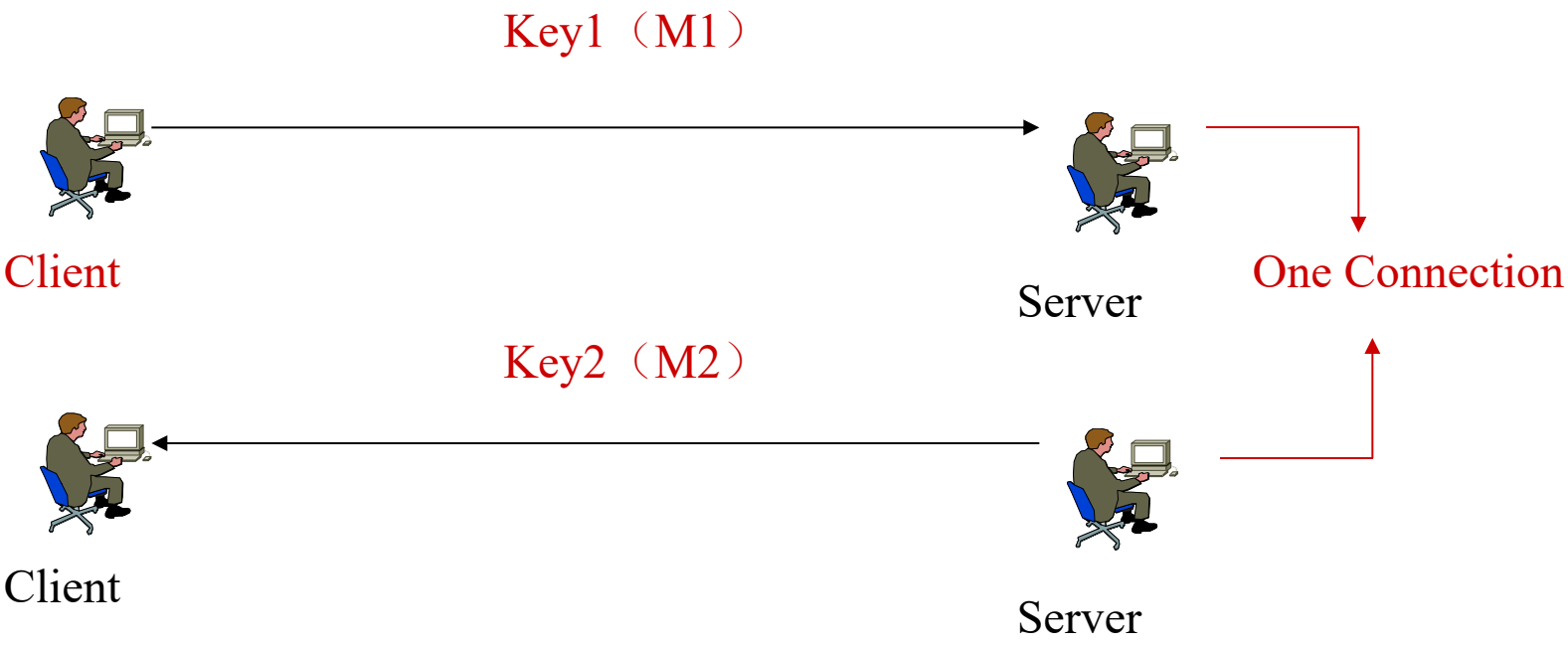
- Connection States includes（内容包括）：
- Server and client randoms: 本次连接的随机数（包括客户和服务器的）
- Server write MAC secrete(服务器MAC计算密钥）：服务器对要发送的数据进行hash运算的秘密值，它也是客户接收时进行Hash运行的秘密值
- Client write MAC secrete（客户端MAC计算密钥）：客户对要发送的数据进行Hash运算的秘密值,它也是服务器接收数据时进行hash运算的秘密值

SSL Session & Session State（Cont.）

- Server write key（服务器发送数据的加密密钥）：服务器用来加密数据、客户用来解密数据的对称加密密钥
- Client write key（客户端发送数据的加密密钥）：客户用来加密、服务器用来解密的对称加密密钥
- Initialization Vectors (IV)：采用密码反馈模式是的初始化向量
- Sequence Vectors：客户/服务器为其在一个连接中发送和接收消息分配的序列号

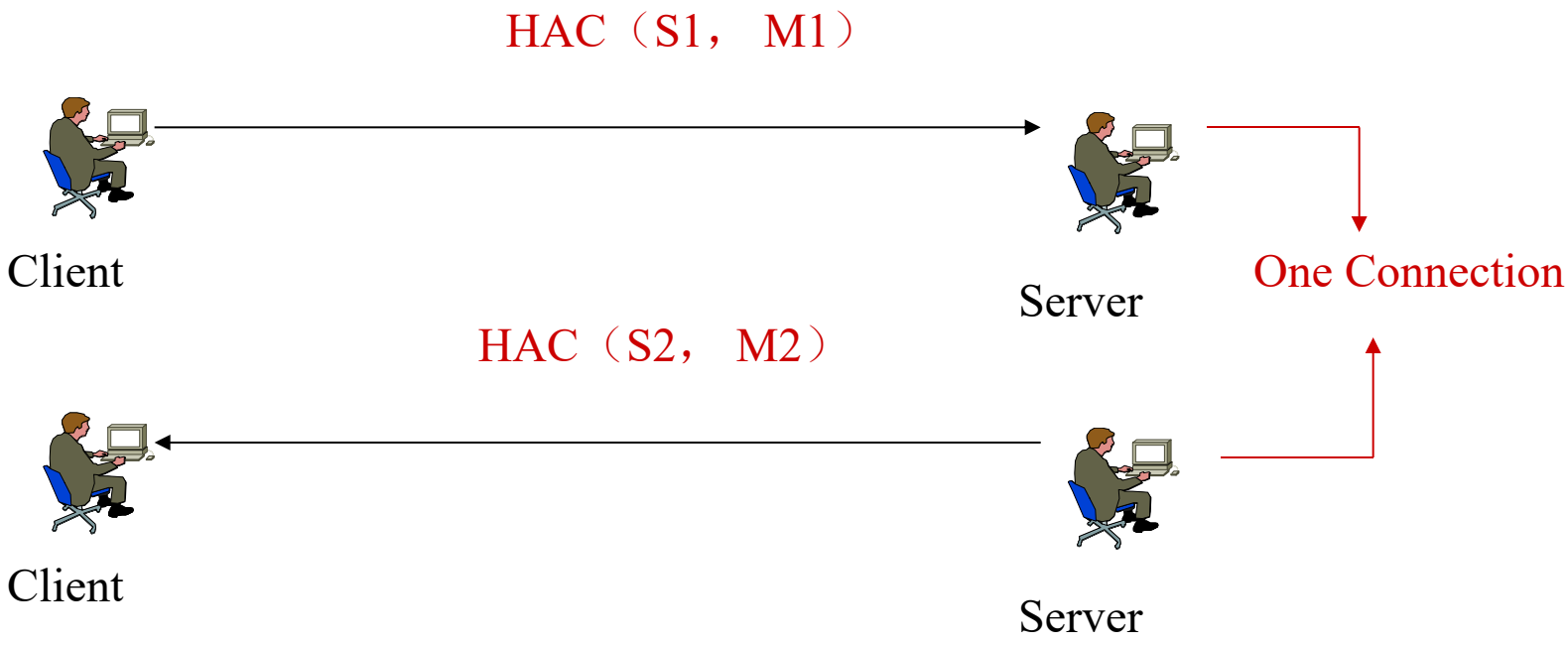
SSL Session & Session State（Cont.）

- Write Key（实际就是会话密钥）



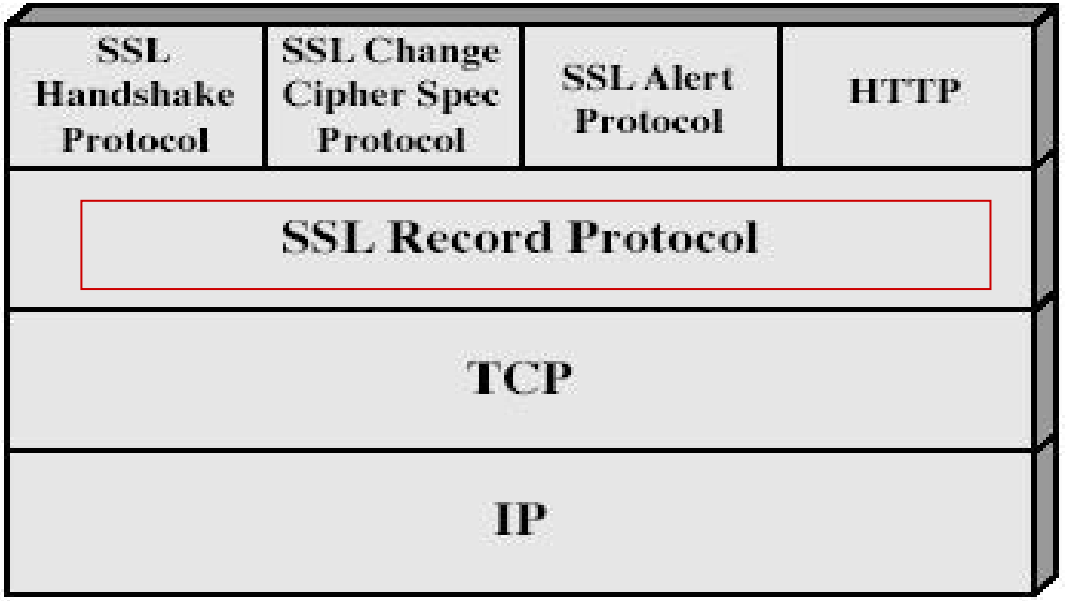
SSL Session & Session State（Cont.）

- MAC Secrete（实际就是MAC运算的秘密值）



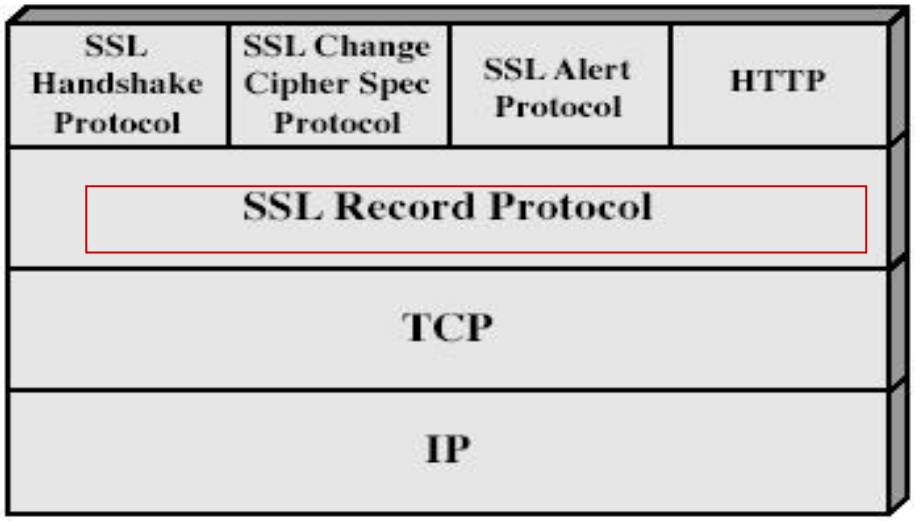
SSL 记录协议

- 记录协议的位置
- 建立在可靠的传输协议（如TCP）基础上
- 提供连接安全性
  - 保密性：使用对称加密算法
  - 完整性：使用HMAC算法



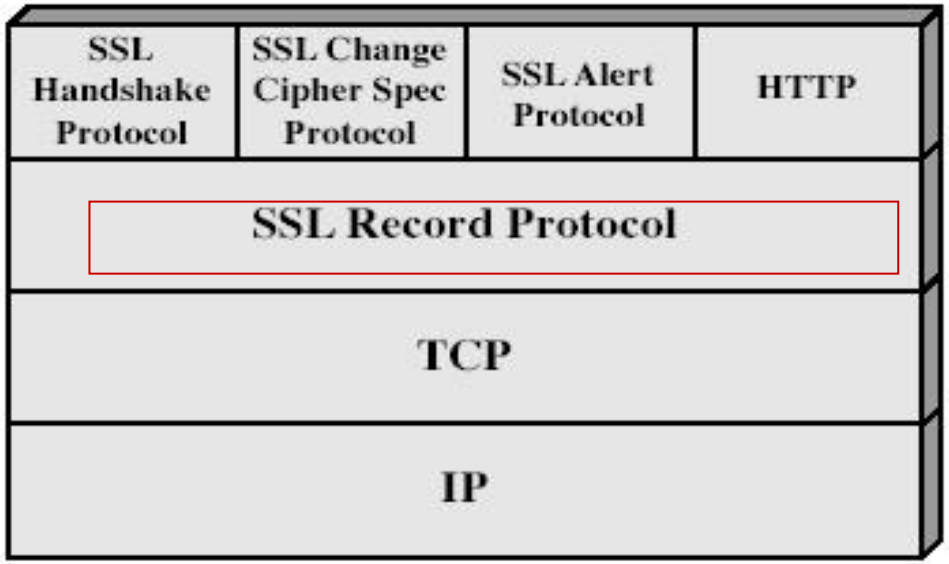
SSL 记录协议

- 根据当前会话状态，给出压缩算法，Cipher Spec给出对称加密算法、MAC算法、密钥长度、Hash长度、IV长度，以及连接状态中给出的Client和Server的随机数、加密密钥、MAC秘密值、IV，消息序列号等，对将要传送的数据实施以下操作：
  - 压缩/解压
  - 加密/解密
  - MAC计算/MAC校验



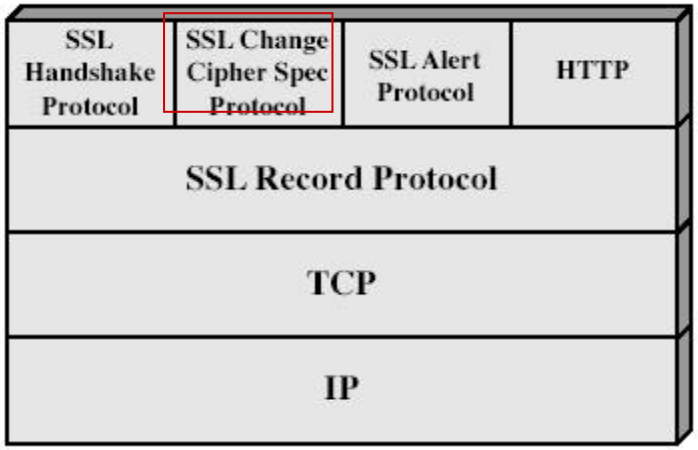
SSL记录协议

- 用来封装高层的协议
  - Change Cipher Spec protocol
  - Alert protocol
  - Handshake protocol
  - Application protocol (HTTP, FTP, TELNET, et al)



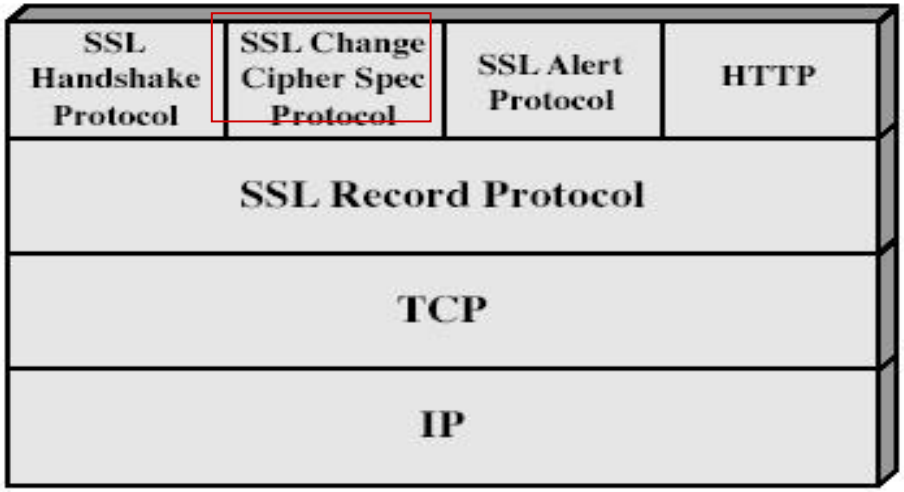
Change Cipher Spec (Cont.)

- Question：在Change Cipher Spec中，提到如果发送该消息后，将使用新的加密、压缩和MAC算法，请问：它和以前的这些信息是一个什么关系(换句话说，握手层协商的加密参数等何时对记录层生效）？
- Answer: SSL使用另外两个状态来管理这种关系
  - 预备状态
  - 当前操作状态

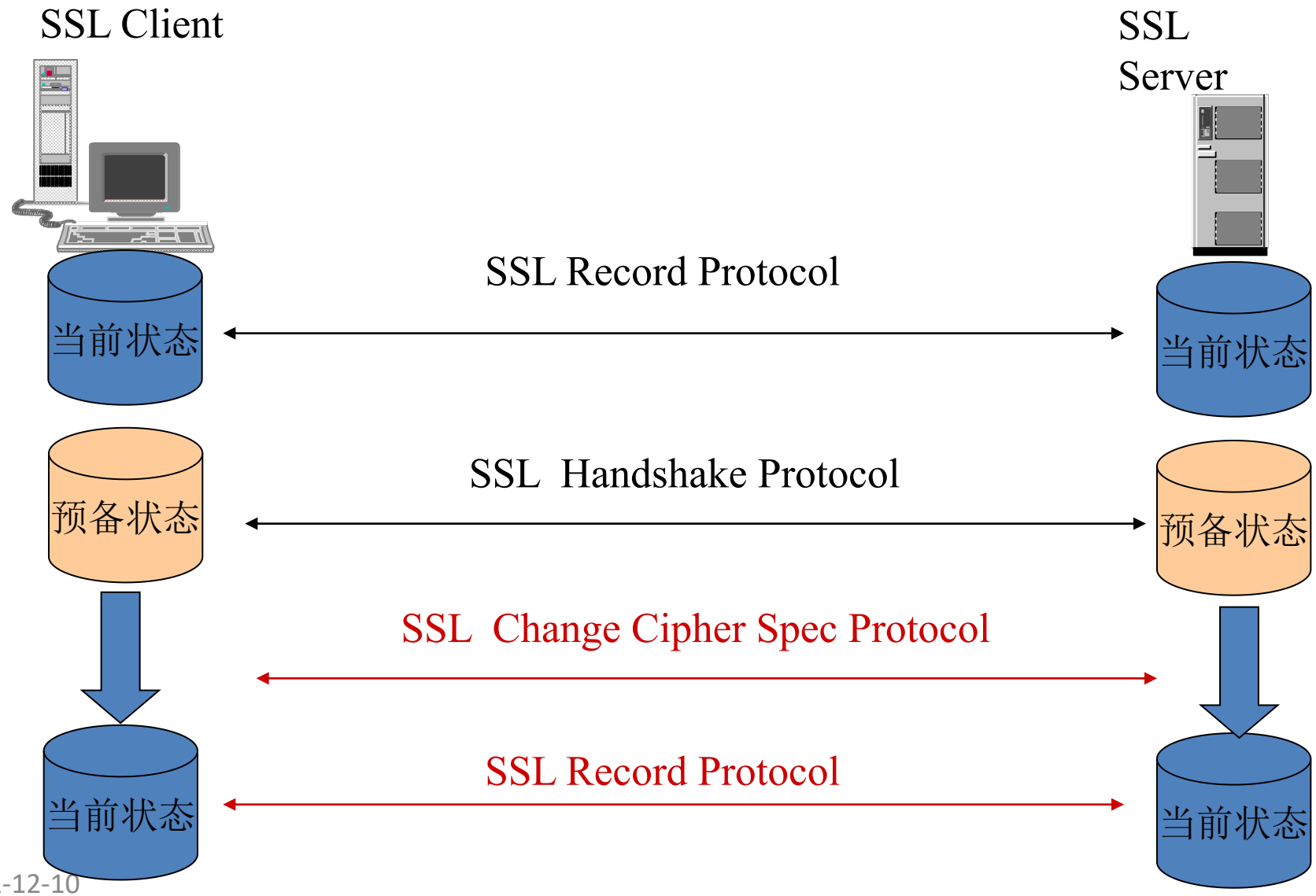


Change Cipher Spec (Cont.)

- Pending State（预备状态）
  - 用来保存握手协议协商的密码信息（如压缩、加密、计算MAC的算法以及密钥等）
- Current State（当前状态）
  - 用来保存记录层正在使用的密码信息（如压缩、加密、计算MAC的算法以及密钥等）

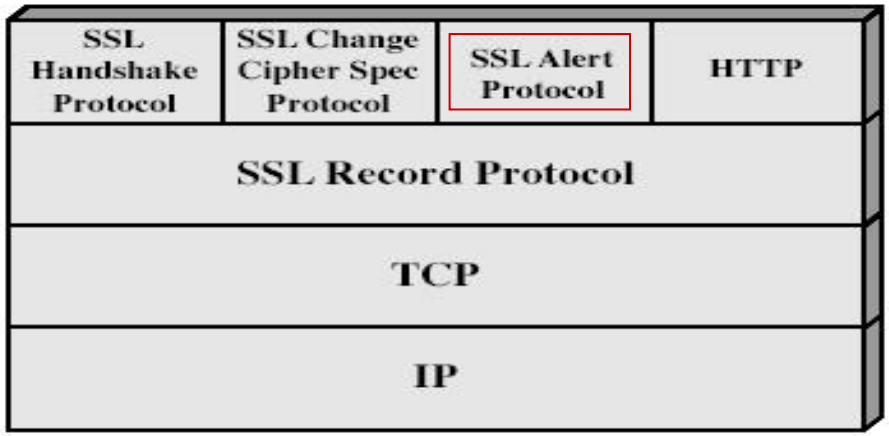


Change Cipher Spec (Cont.)



Alert protocol

- Alert Protocol：SSL警报协议
  - 用于将SSL握手或者数据加密等过程有关的告警传输给对方，向对方发出警告或者中止当前连接
  - 根据错误程度，Alert消息分为两类：警告消息（warning Msg）和致命消息（Fatal Msg）
    - 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（会话标识符）作废，以免这个会话被继续用来建立新的连接

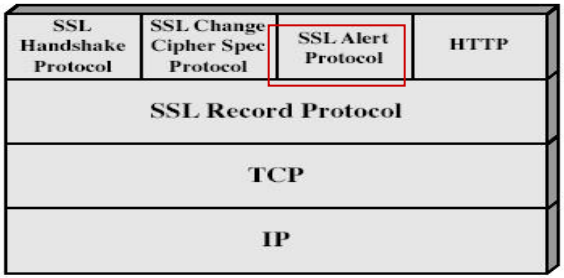


Alert protocol (Cont.)

- 根据功能，Alert消息被分为两类：
  Close_Notify 和Error Alerts
  - Close_Notify:通知对方关闭该连接
  - Alert Error：通知对方关闭该连接
  - 区别：
    - **前者关闭的连接可以被恢复**
    - **后者不能**
- Alert消息被加密传输

Alert protocol (Cont.)

- 警告消息（Warning Msg）
  - 结束通知
  - 无证书
  - 证书出错
  - 不支持的证书
  - 证书撤销
  - 证书过期
  - 未知证书



Alert protocol (Cont.)

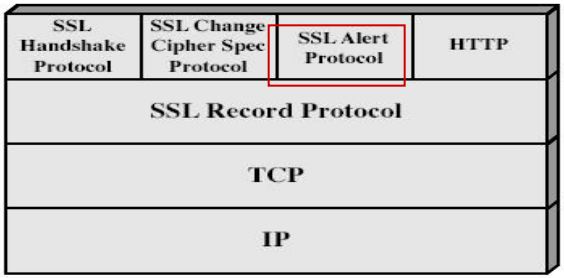
- 致命消息（Fatal Msg）
  - 意外消息
  - MAC记录出错
  - 解压失败
  - 握手失败
  - 非法参数

Alert protocol (Cont.)

- 警报协议格式

Level	Alert

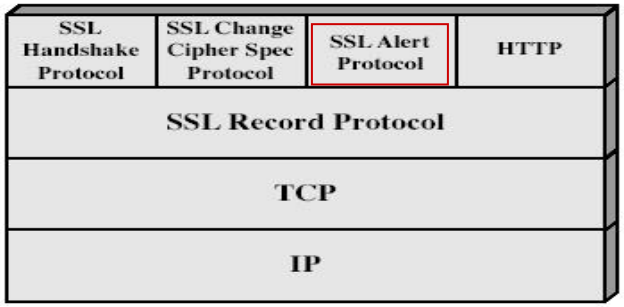
字段取值		含义
Level	1	警告
	2	SSL即将关闭
Alert	xx	均为严重警告消息
	yy	



Application protocol

- 应用层协议
  - 加密传输

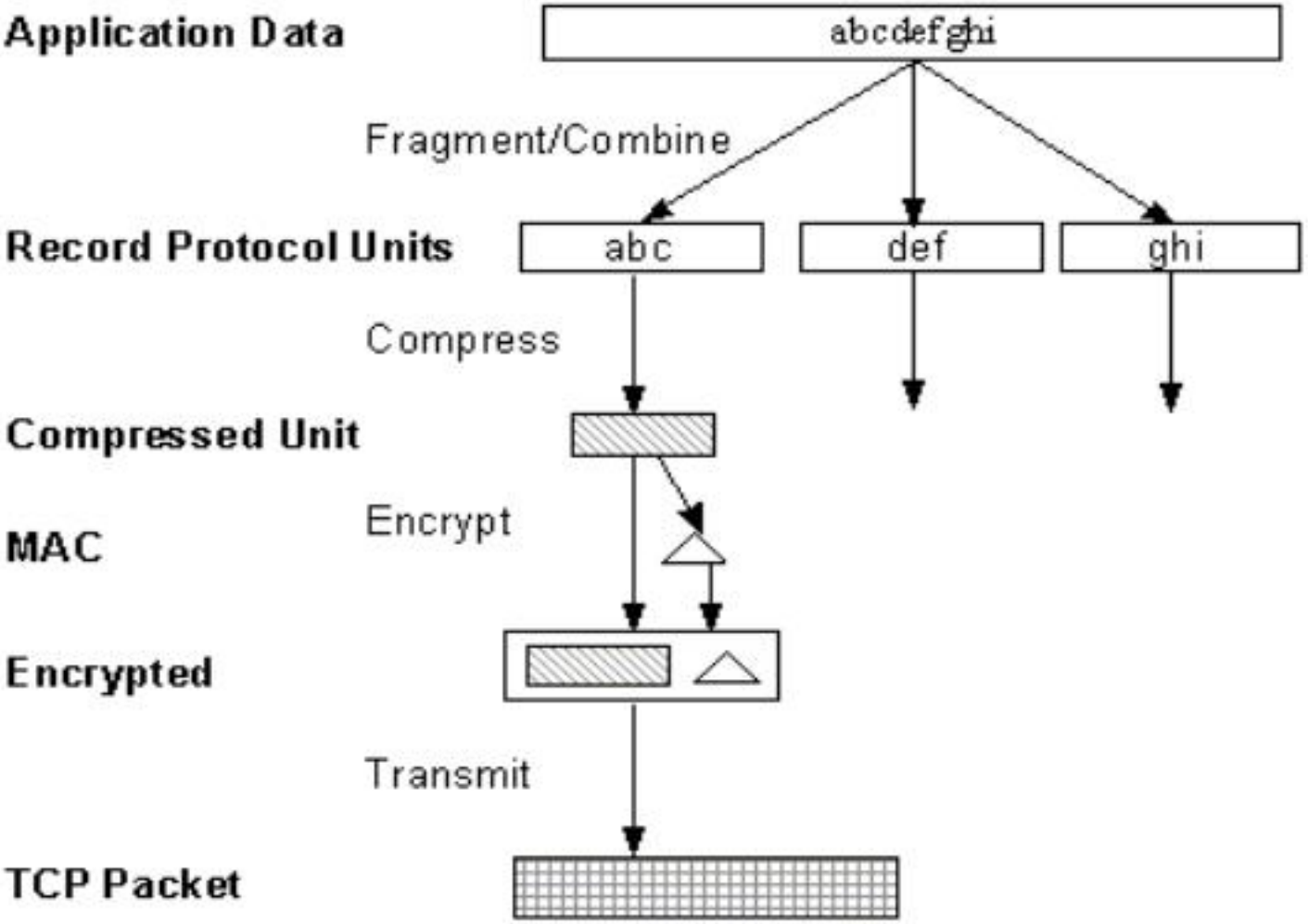
不透明的内容（大于1字节）



Working Process of Record layer

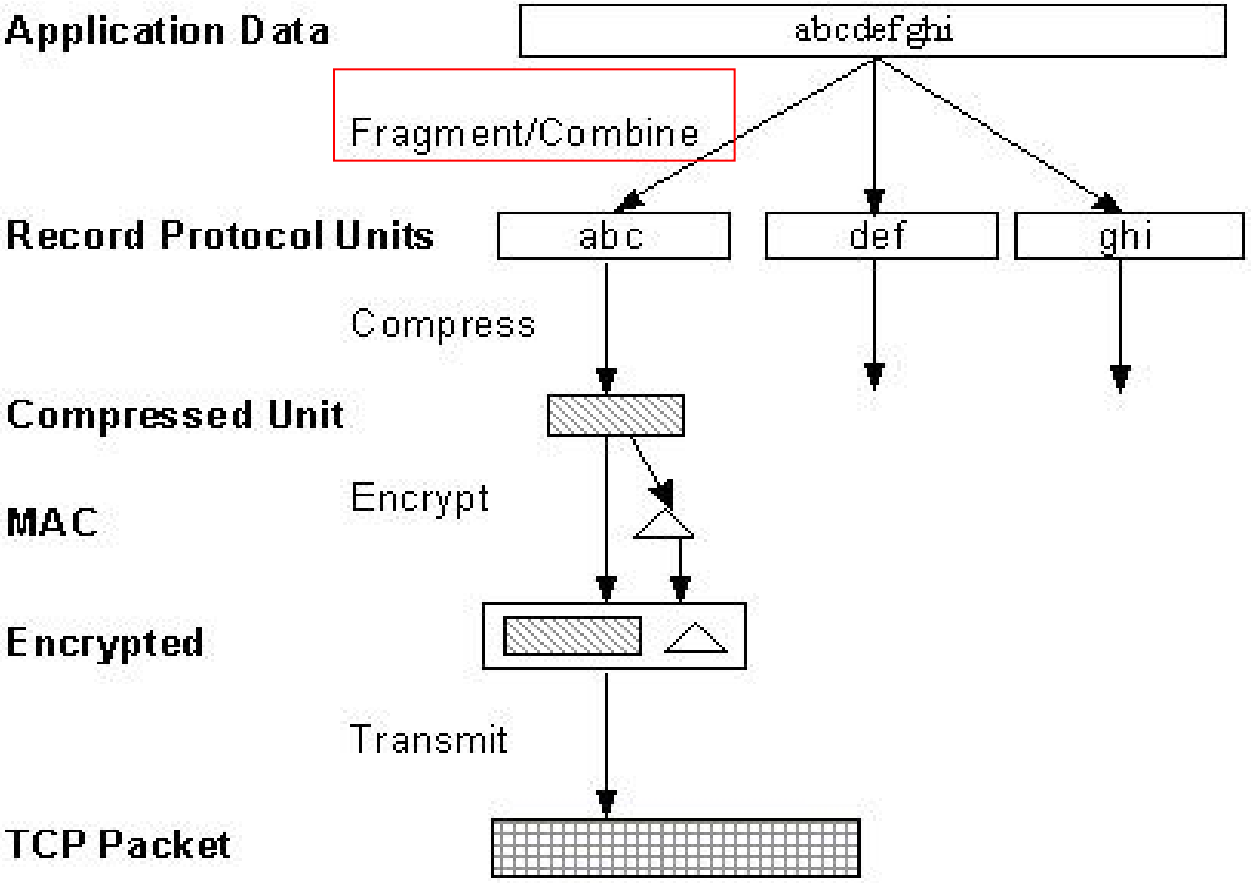
- 发方
  - Step 1 分片：从上层接收任意大小的数据块（ Records）
  - Step 2 压缩：用当前会话状态中给出的压缩算法明文结构SSLPlaintext压缩为压缩记录SSLCompressed
  - Step 3 MAC计算：用当前会话状态中指定的MAC算法对SSLCompressed计算消息摘要
  - Step 4机密：用加密算法加密压缩数据和消息摘要，形成密文结构SSLCiphertext
  - Step 5封装发送：将数据封装为可靠传输层协议的数据包，并发送到可靠传输层协议

Working Process of Record layer (Cont.)



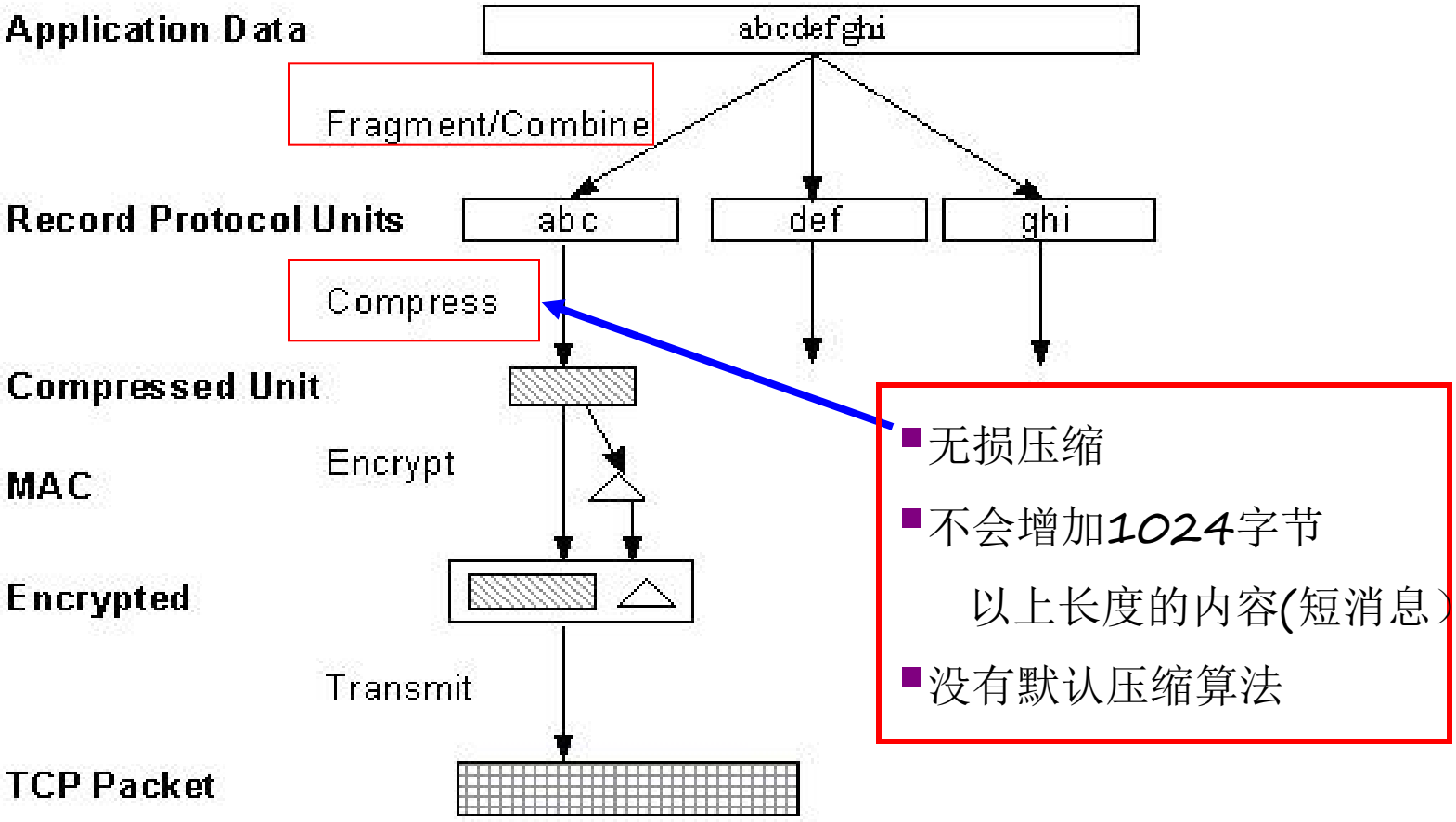
Working Process of Record layer (Cont.)

- 第一步：分片



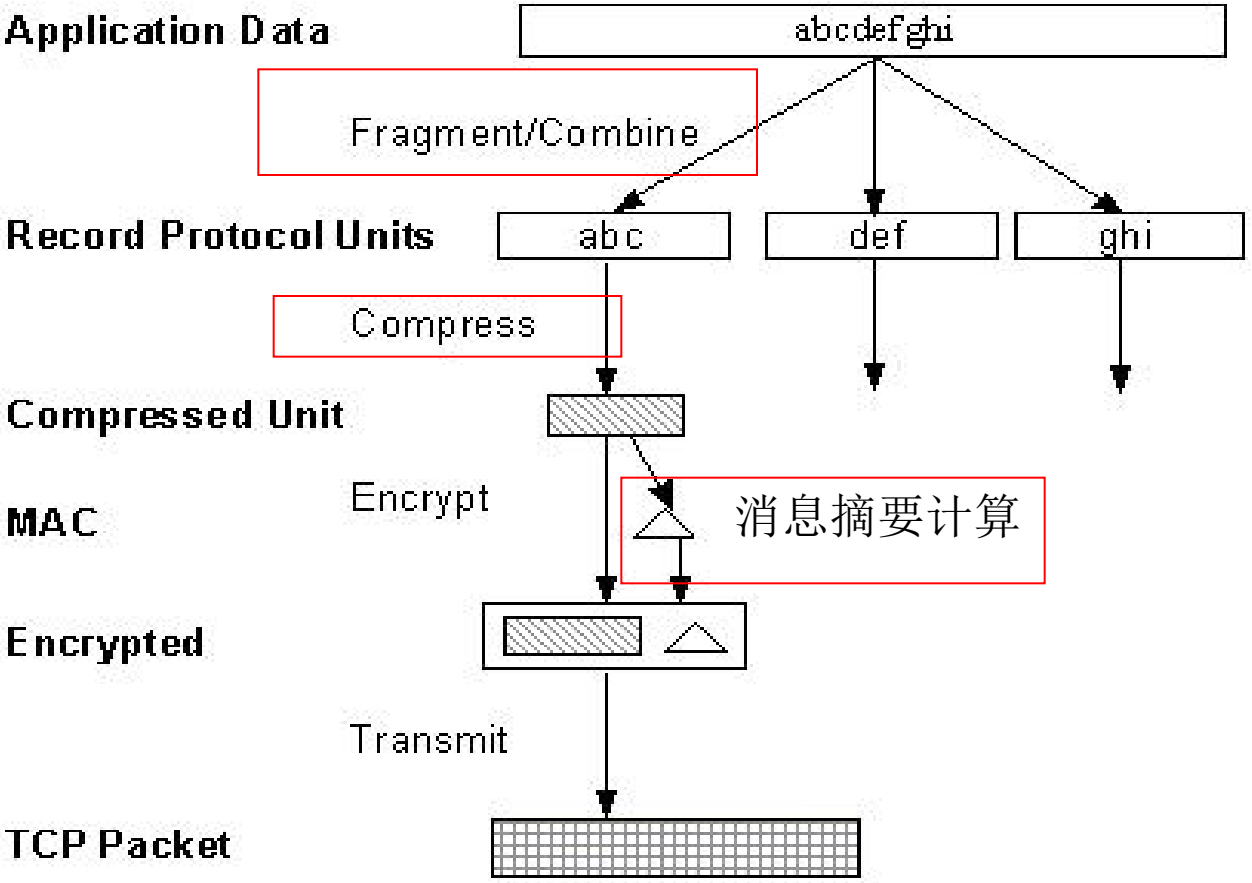
Working Process of Record layer (Cont.)

- 第二步：压缩



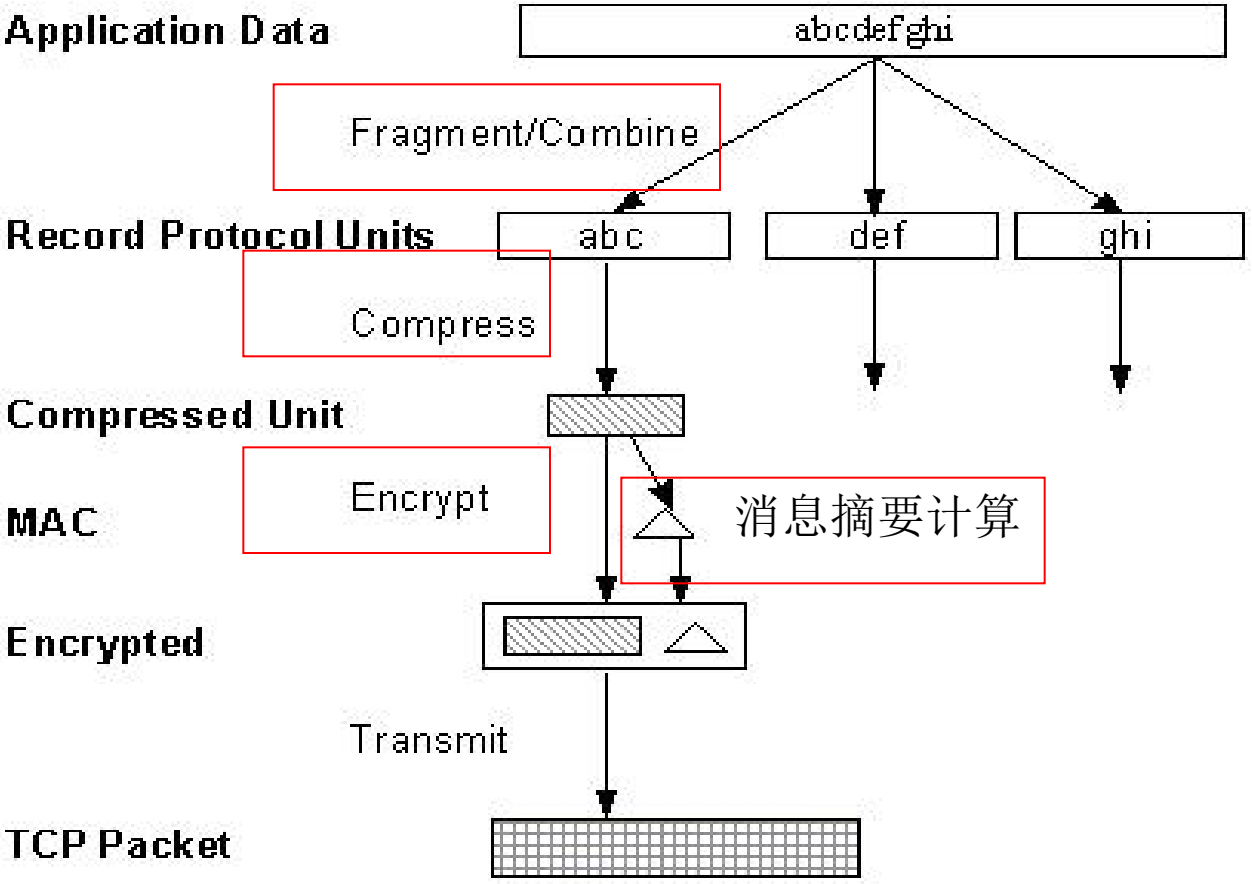
Working Process of Record layer (Cont.)

- 第三步：MAC计算



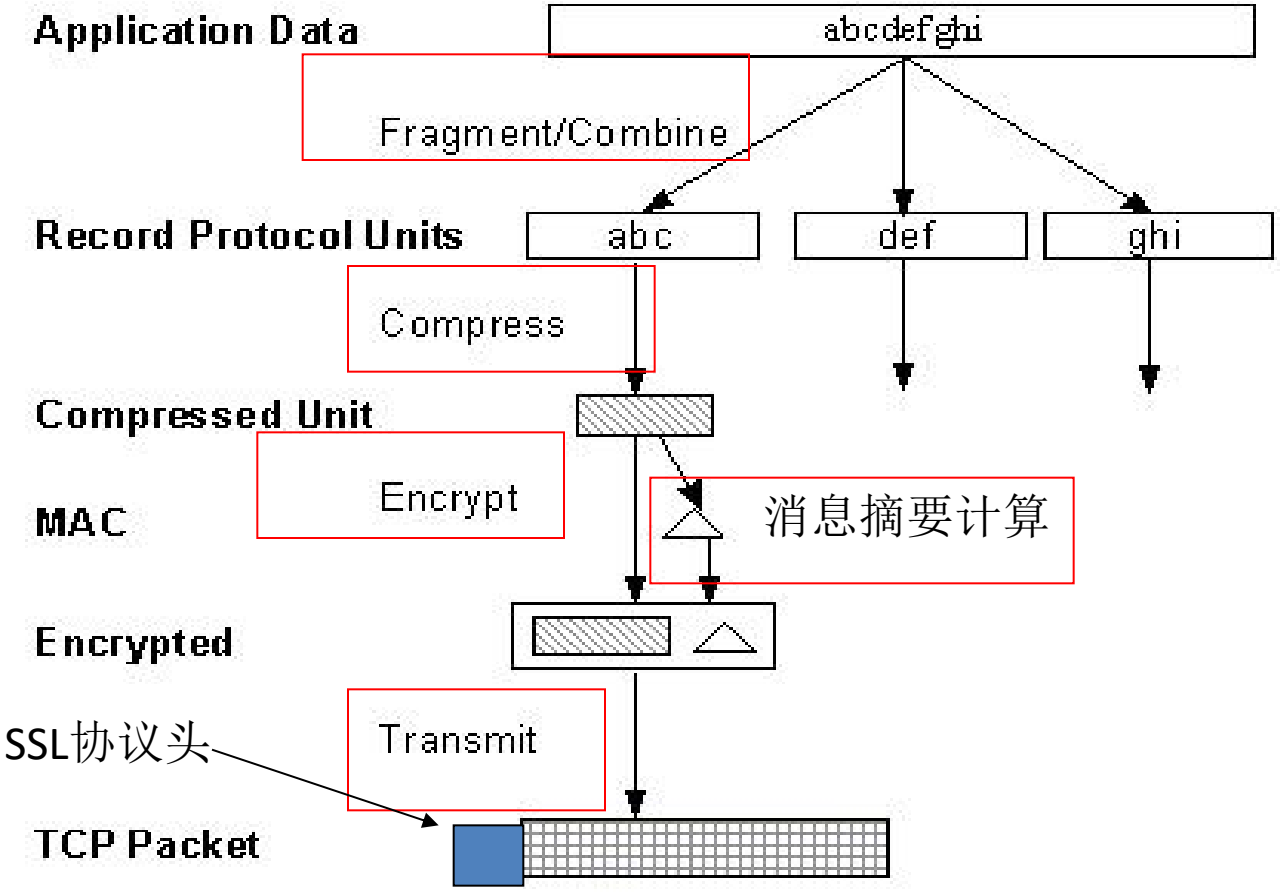
Working Process of Record layer (Cont.)

- 第四步：加密



Working Process of Record layer (Cont.)

- 第五步：封装发送



Working Process of Record layer (Cont.)

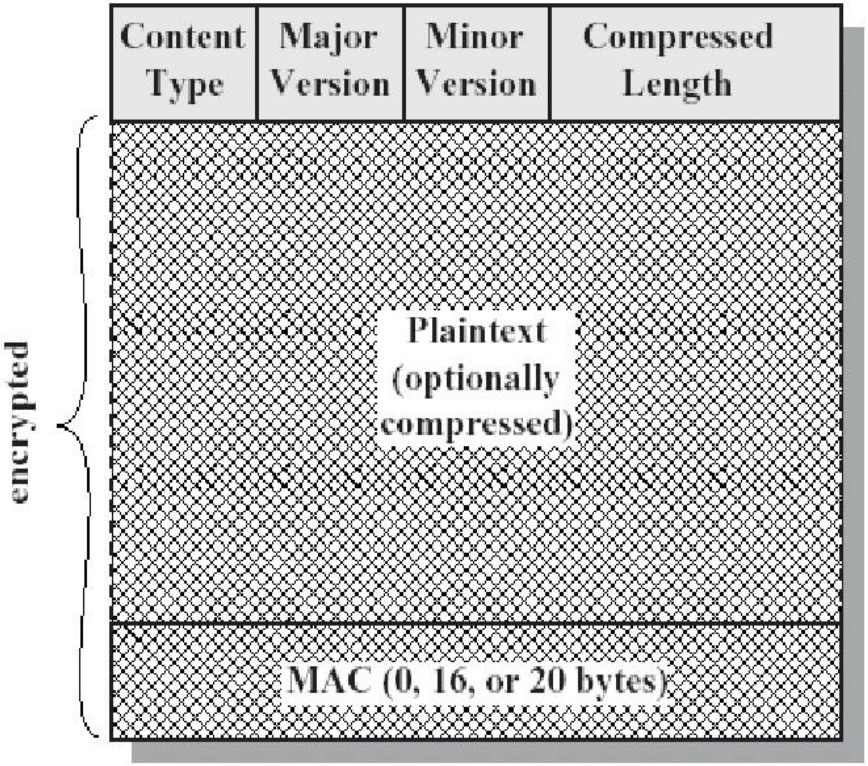
其中：

MAC_write_secret	: 共享的保密密钥
hash	: 密码散列函数（MD5或SHA-1）
pad_1	: 0x36重复48次(MD5)；或40次(SHA-1)
pad_2	: 0x5C重复48次(MD5)；或40次(SHA-1)
seq_num	: 该消息的序列号
SSLCompressed.type	: 更高层协议用于处理本分段
SSLCompressed.length	: 压缩分段的长度
SSLCompressed.fragment	: 压缩的分段(无压缩时为明文段)

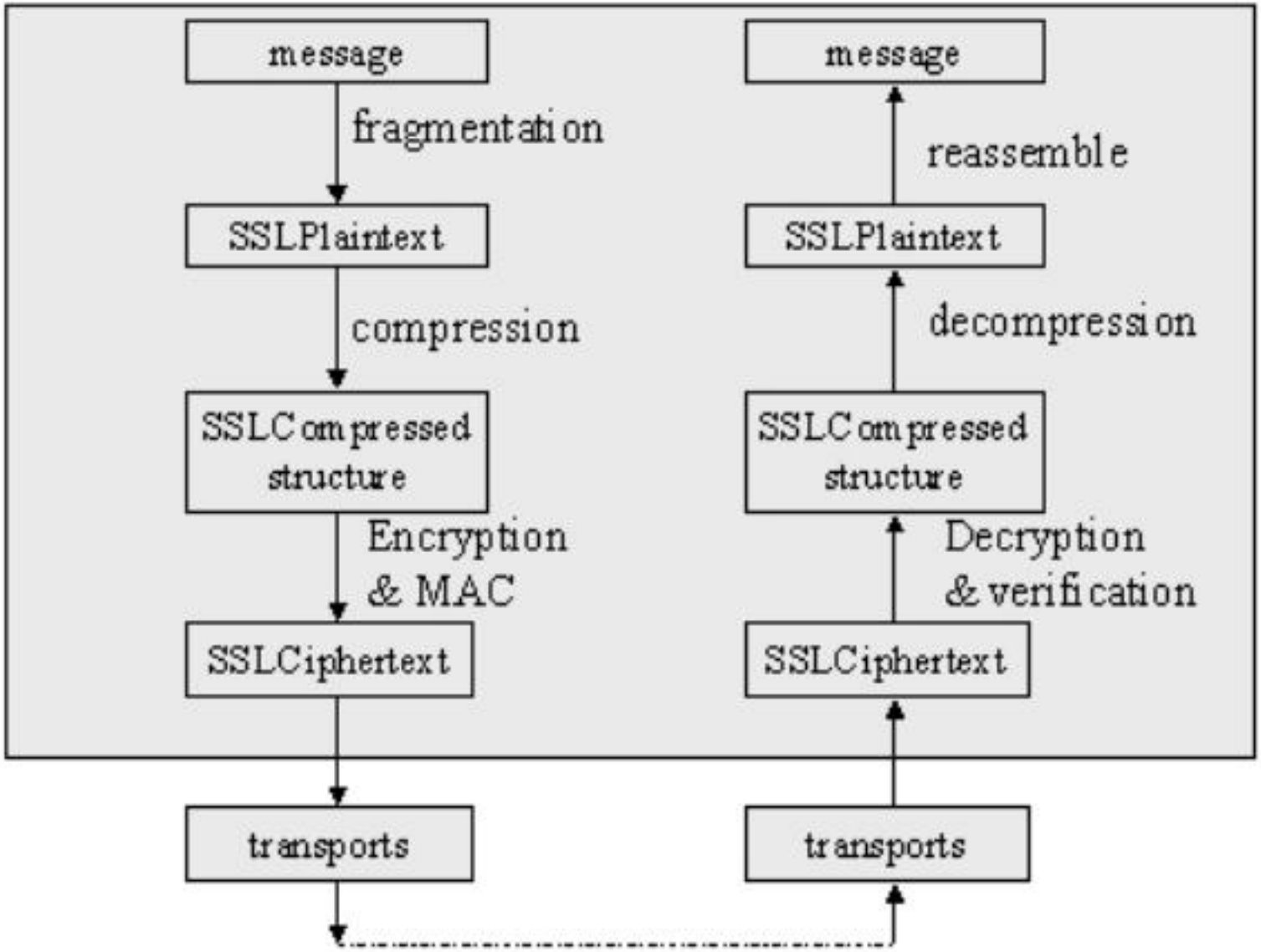
hash(MAC_write_seret||pad_2||
hash(MAC_write_secret||pad_1||seq_num||SSLCompressed.type||
SSLCompressed.length||SSLCompressed.fragment)

记录协议的工作流程

- ContentType; —— 8位，上层协议类型
- Major version; Minnor version—— 16位，主次版本
- 压缩长度：16位— 加密后数据的长度，不超过$2^{14}+2048$字节
- EncryptedData fragment; —— 密文数据



记录协议的工作流程


message
fragmentation
SSLPlaintext
compression
SSLCompressed structure
Encryption & MAC
SSLCiphertext
transports
message
reassemble
SSLPlaintext
decompression
SSLCompressed structure
Decryption & verification
SSLCiphertext
transports

问题

- Question：

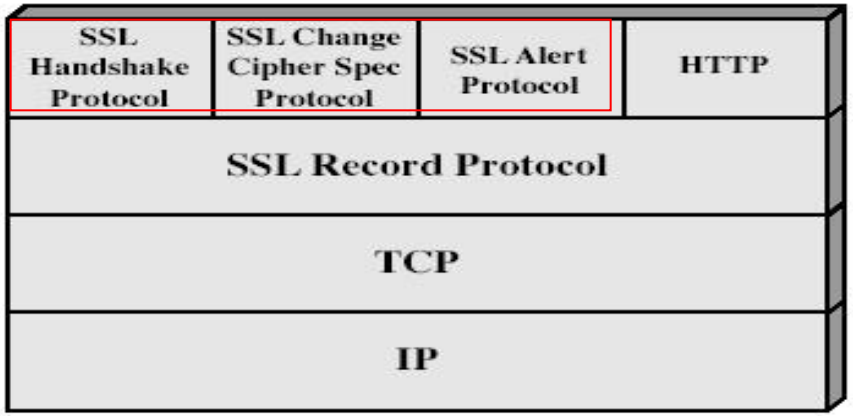
  Record protocol layer中使用的密码及相关信息如何得到？

- Answer：

  由handshake protocol协商并进一步计算得到。

Handshake protocol

- 握手协议
  - 握手协议时SSL中最复杂的部分
  - 此协议允许客户端和服务器相互认证、协商加密和MAC算法，保护数据使用的密钥通过SSL记录传递
  - 握手协议在传递应用程序数据之前使用



Handshake protocol 功能回顾

- 握手协议层的功能
  - 客户和服务器之间相互鉴别
  - 协商密钥交换算法
  - 协商加密算法和密钥
  - 协商压缩算法
  - 生成密钥
  - 完成密钥交换

Handshake protocol使用的协议

- 使用三个协议
  - SSL Handshake protocol
    - 核心协议
  - SSL Change Cipher Spec
    - 改变参数时使用
  - SSL Alert protocol
    - 出现错误时使用

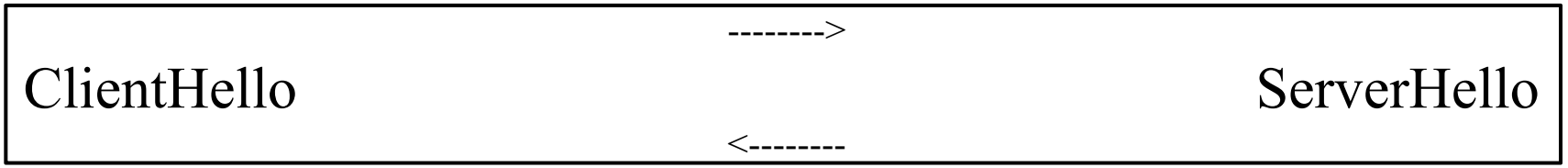
Handshake protocol的作用

- 握手协议建立一个会话或者恢复一个会话
- 握手协议中每次握手都会生成新的密钥，MAC秘密值和初始化向量等。
- Client和Server之间要建立一个连接，必须进行握手过程，每次握手都会存在一个会话和一个连接，连接一定是新的，但会话可能是已经存在的。

握手协议的本质

- 握手协议（Handshake protocol）本质上是一个密钥交换协议，但它也包含认证功能，因此可以视为认证和密钥交换协议
- 握手协议主要由四个过程组成
  - 建立安全能力
  - 服务器认证和密钥交换
  - 客户端认证和密钥交换
  - 完成
- Question：为什么需要服务器密钥交换和客户端密钥交换两个过程？
- Answer：因为SSL在同一个连接的两个方向采用不同的密钥

Handshake protocol的第一步

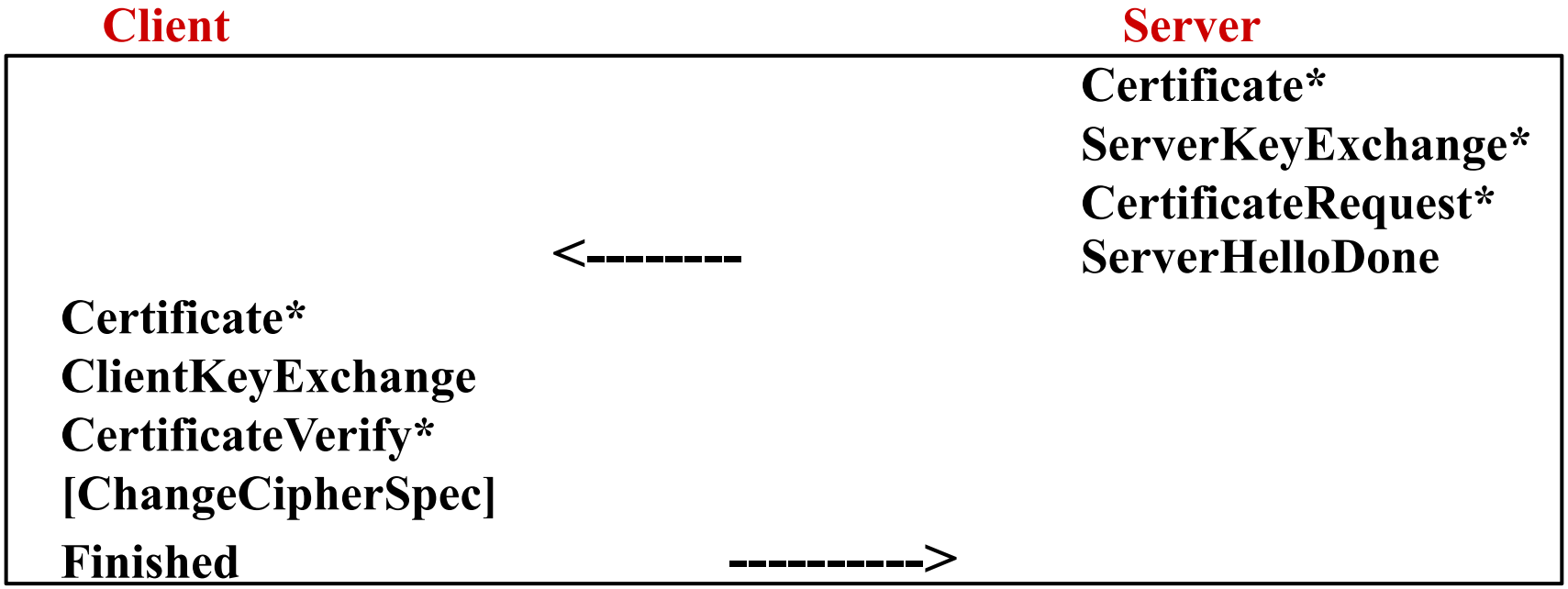


第一步完成：

（1）客户告诉服务器自己的要求或能力

（2）服务器按照自己的要求或能力选择并回应客户

Handshake protocol的第二步



第二步完成（服务器）：

（1）向客户证明自己的身份

（2）完成密钥交换

（3）向客户提出自己的认证要求（可选）

（4）告诉客户自己本阶段结束

第二步完成（客户）：

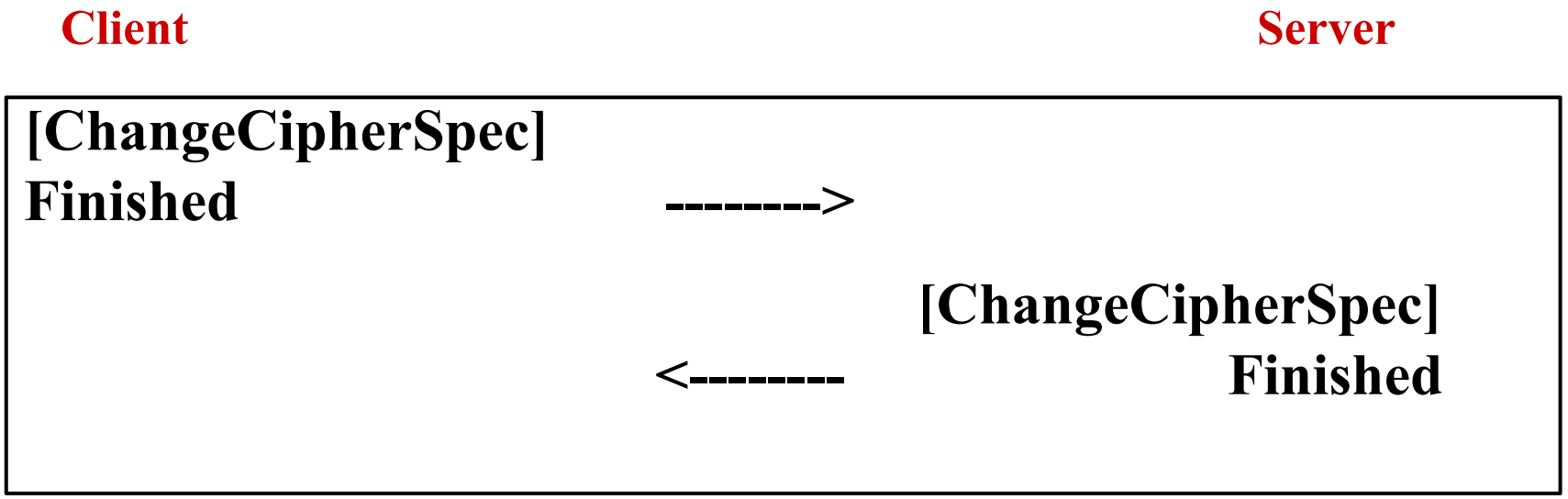
（1）向服务器证明自己的身份（可选）

（2）完成密钥交换

（3）向服务器发送自己的认证校验信息（可选）

（4）告诉服务器自己本阶段结束

Handshake protocol的第三步



第三步完成（服务器）：

（1）向客户通告启用新的密码参数

（2）告诉客户自己本阶段结束

第三步完成（客户）：

（1）向服务器通告启用新的密码参数

（2）告诉服务器自己本阶段结束

Handshake protocol的第四步

Client　　Server

Application Data <-------> Application Data

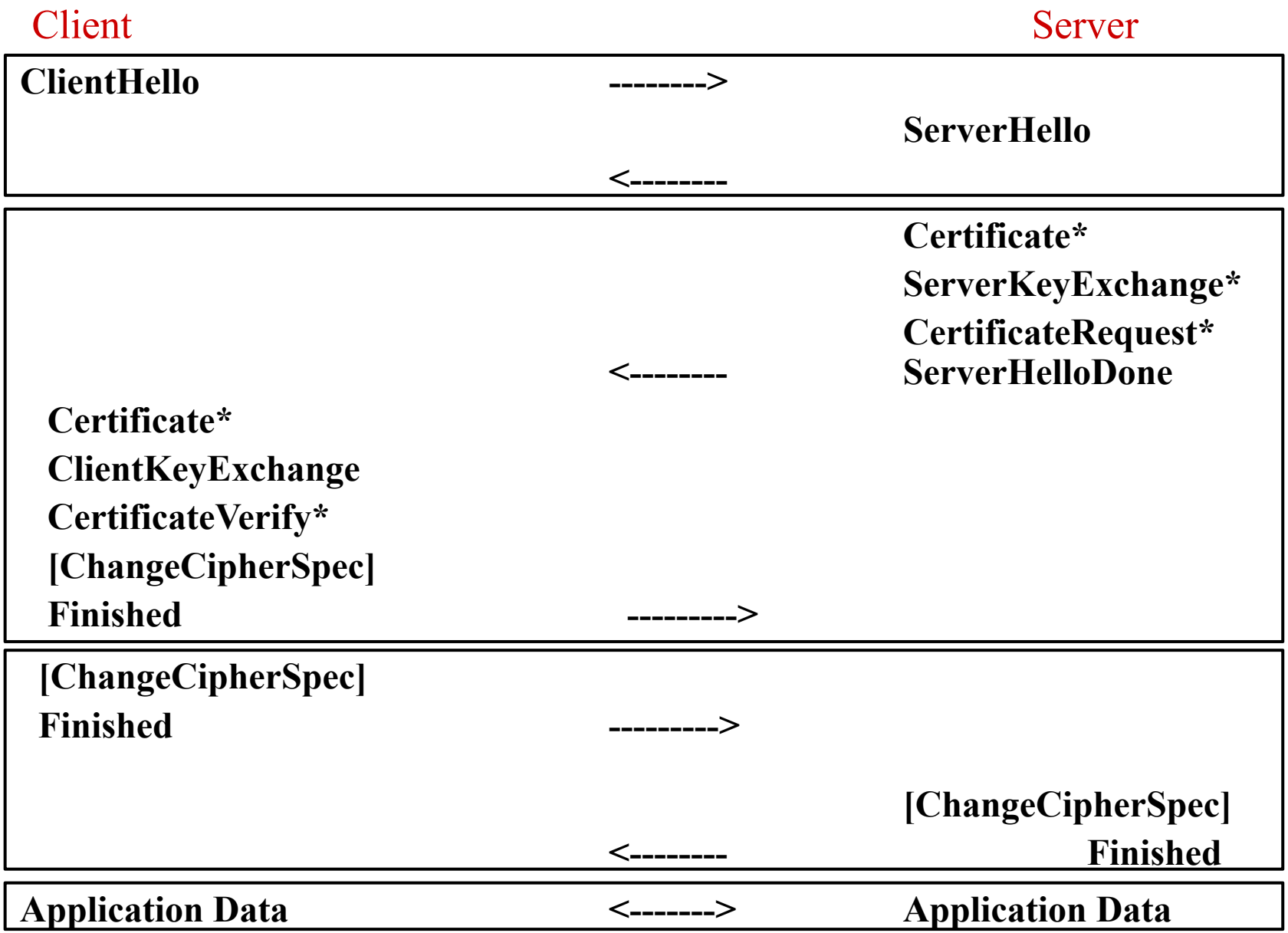
第四步完成（服务器）：

（1）利用启用的密码参数加密发送数据

第四步完成（客户）：

（1）利用启用的密码参数加密发送数据

SSL Handshake protocol总结



第一阶段： 建立安全能力（客户->服务器）

The Client Hello message

- SSL Version (highest) that is understood by the client. （SSL版本号）
- A Random number to compute the secret key（随机数，防重放攻击，记为ClientHello.random）
- Session Identifier（会话标识）
  - 0： 新建一个会话和连接
  - 非0：在已有会话上建立连接
- Data Encryption to identify the encryption methods available to the Client（密码组），包括两个部分
  - 密钥交换算法（Key Exchange）
  - 密码规范（cihper Spec）
- Data Compression method for message exchange（压缩方法）

第一阶段： 建立安全能力（服务器->客户）

The Server Hello message

- SSL Version (highest) that is understood by the client. （SSL版本号）
- A Random number to compute the secret key（随机数，防重放攻击，记为ServerHello.random ）
- Session Identifier（会话标识）
- Data Encryption to identify the encryption methods （密码组）
- Data Compression method for message compression（压缩方法）

问题1：两个Hello消息如何产生？

- 客户端根据自己的实际情况构建client Hello message
- Question: How does the server compute the server hello message and send to client?
- Answer:
  - SSL Version?

    The highest the client available
  - A Random number to compute the secret key?

    Server generate it and it is independent of client's number
  - Session Identifier（会话标识）?

    If the client's identifier is not 0 then the same as client's, else generate a new identifier
  - Data Encryption to identify the encryption methods?

    The one chose from client's encryption methods list.
  - Data Compression method for message compression?

    The one chose from client's compression methods list.

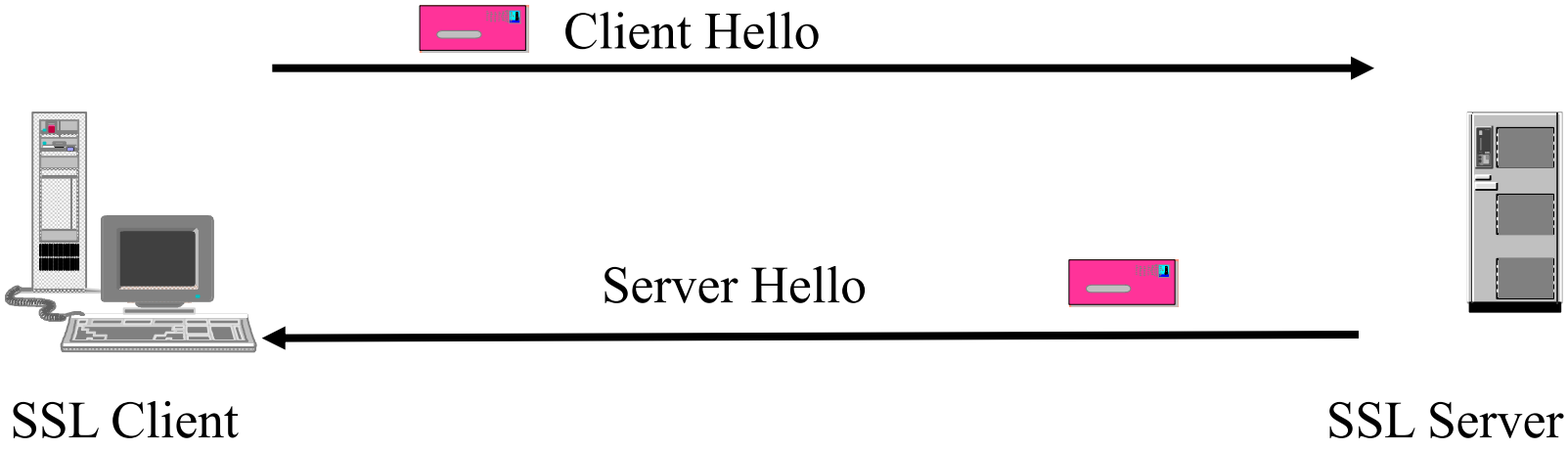
问题2：密码参数有哪些？

- **密钥交换算法:**
  - 1)RSA密钥交换协议客户端用服务器的RSA公钥加密的密钥（此时，客户端必须拥有服务器的公钥证书）
    2)DH密钥交换协议
    3)Fortezza密钥交换协议
    - 固定DH：包含认证中心签发的Diffie-Hellman公钥参数的服务器证书（公钥证书包含客户端Diffie-Hellman公钥参数，该参数由客户端在证书数中提供，或者在密钥交换消息中提供）
    - 临时Diffie-Hellman:此技术用来创建一次性密钥。在这种情况下，Diffie-Hellman公钥在交换时使用服务器的RSA或者DSS私钥签名，客户端使用服务器的相应公钥验证签名
    - 匿名DH：使用基本的Diffie-Hellman
- **密码规范(CipherSpec)**
  - 密码算法
    MAC算法
    密码类型：流密码或者分组密码是否可出口（只在美国以外使用）
    MAC长度
    密钥材料
    IV大小
- **压缩算法**

问题（续）

- Question： Why does SSL use three kind of DH algorithms?
- Answer: Provides different security abilities
  - 临时DH安全最高
  - 匿名DH安全性最差，容易受到中间人攻击

握手协议工作过程第一步



握手协议工作过程第二步

- 包含是否需要对客户端进行认证、服务器自己的证书消息等。该过程一般包含四条消息：
  - 证书消息（Certificate）
  - 服务器密钥交换消息（Server_Key_Exchange)
  - 客户端证书请求消息（Certificate_Request)
  - 服务器结束消息（Server_Hello_Done)

第二阶段：服务器认证和密钥交换（服务器->客户）

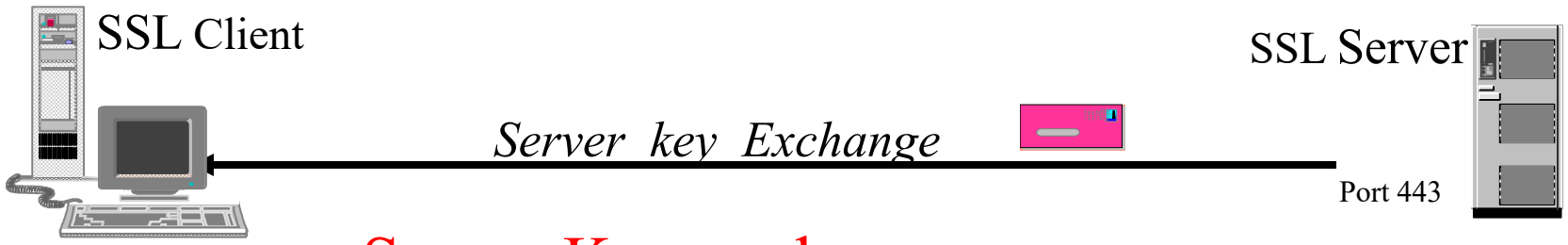
The Server Certificate message

- The server Identifier information
- A Digital Certificate of the sever information encrypted with the CAs Private Key（也可能是一组证书）
  - This contains the server‘s Public Key （包含服务器公钥）

说明：

服务器证书消息是服务器向客户端传送自己的证书，使得客户端知道服务器的公钥以及其他信息。

第二阶段：服务器认证和密钥交换（服务器->客户）



Server Key exchange message

- The Certificate type to indicate the type of public key
- The Certificate Authority is a list of distinguished names of Certificate Authorities acceptable to the Server

说明：

- 服务器密钥交换消息用来向客户端发送服务器自己的密钥信息
- The Server Key exchange message不是必须的
  如果使用了固定Diffie-Hellman或者RSA密钥交换，则不需要。
  反之，如果使用匿名Diffie-Hellman、瞬时Diffie-Hellman、Fortzza或者服务器在使用RSA时仅用了RSA签名密钥。
  - RSA：在第一阶段包含了服务器的公钥
  - 固定DH：由于在固定DH中，服务器在第一阶段发送的证书消息中包含了服务器自己的公钥

第二阶段：服务器认证和密钥交换（服务器->客户）

The Client Certificate Request message

- The Certificate type to indicate the type of public key (证书类型)
  - RSA
  - DSS
  - Diffie-Hellman

  .....
- The Certificate Authority is a list of distinguished names of Certificate Authorities acceptable to the Server(认证中心CA列表）
  服务器可以接受的认证中心列表

说明：

如果服务器不使用匿名Diffie-Hellman，则客户端证书请求消息是必须的。它的目的是要求客户端向自己发送证书等消息，进行客户认证。

第二阶段：服务器认证和密钥交换（服务器->客户）

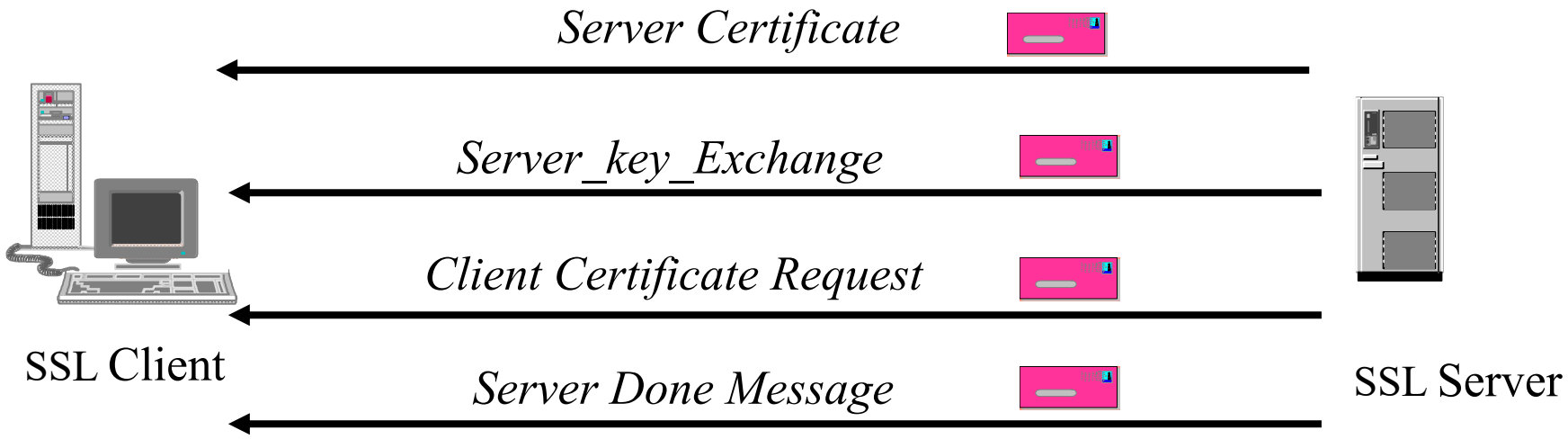
Server Done message

表明服务器的hello阶段结束；

在此消息之后，服务器将等待客户端的应答。

第二阶段：服务器认证和密钥交换（服务器->客户）

总结



第三阶段：客户机验证和密钥交换

- 在接收到服务器完成消息（Sever done)之后
  - 客户端需要验证服务器是否提供了合法的证书
  - 检查server_hello参数是否可以接受
  - 如果所有条件满足，则客户端向服务器发回一个或者多个消息：
    - 客户端证书消息（Certificate)：如果服务器请求了证书，必须有该消息
    - 客户端密钥交换消息(Client_Key_Exchange)：必须发送
    - 客户端证书校验消息（Ceritificate_Verify)：可以发送(便于服务器验证自己的证书）

第三阶段：客户机验证和密钥交换（客户->服务器）

Client Certificate message

- The Client Identifier information
- A Digital Certificate of the client information

说明：

如果服务器请求了证书，但是客户端没有合适的证书，则发送”无证书警报“消息

第三阶段：客户机验证和密钥交换（客户->服务器）

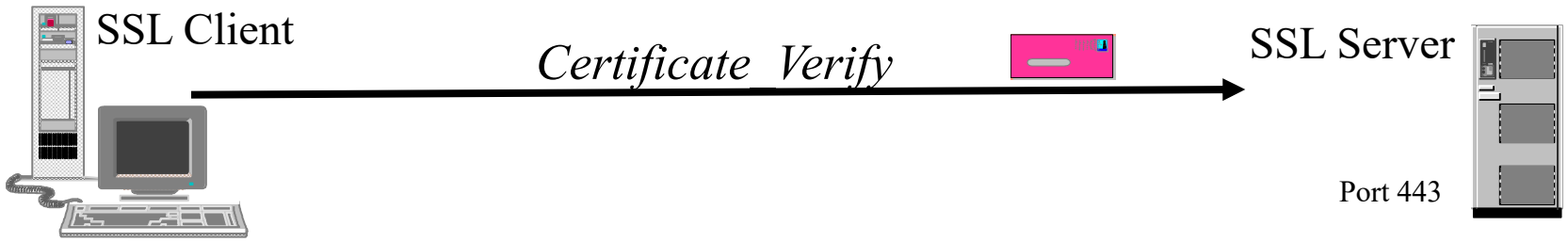
Client Key Exchange message

- The encrypted session key which will serve as a pre-master secret key encrypted with the server's public key.

说明：

- 如果是RSA，该消息包含客户端生成的48字节的预主密钥（ pre-master secret key ），并使用服务器证书中的公钥或者服务器密钥交换消息中的临时RSA密钥加密，它用于生成主密钥（ master secret key ）
- 如果是Diffie-Hellman，该消息包含客户端的Diffie-Hellman公钥参数
- 如果是Fortezza，则包含客户端的Fortezza参数

第三阶段：客户机验证和密钥交换（客户->服务器）



Certificate Verify Message

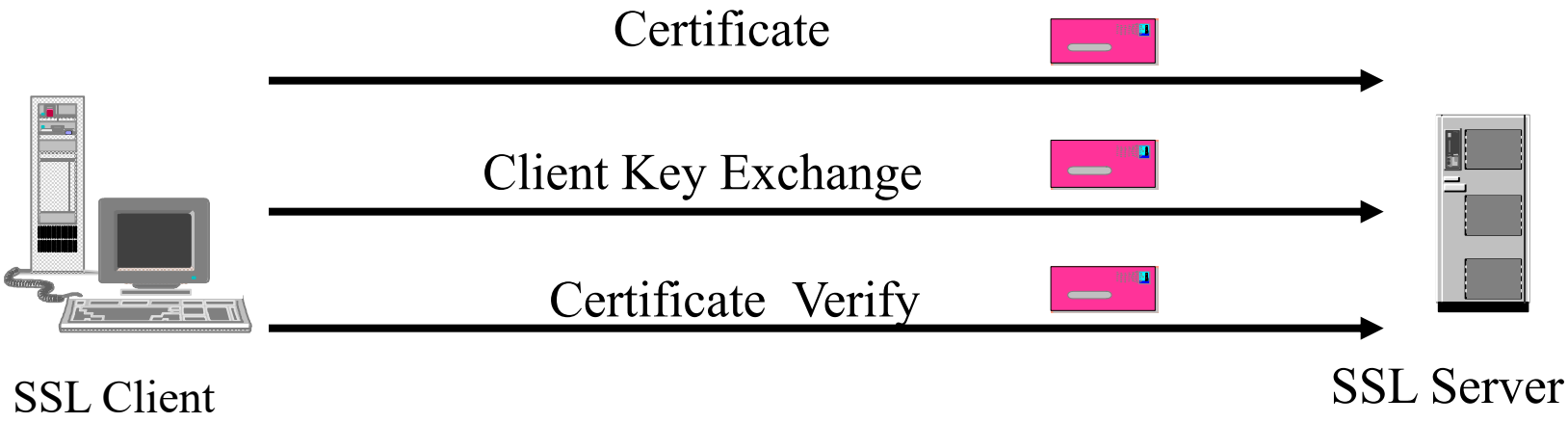
- The Certificate Verify Message is composed of the Hash value of the client's certificate.

说明：

证书验证消息很复杂，它对签名的所有消息即密钥信息等进行Hash运算，并用自己的私钥加密，从而即是有人盗用了客户端证书，也无法发送该证书验证消息。

第三阶段：客户机验证和密钥交换（客户->服务器）

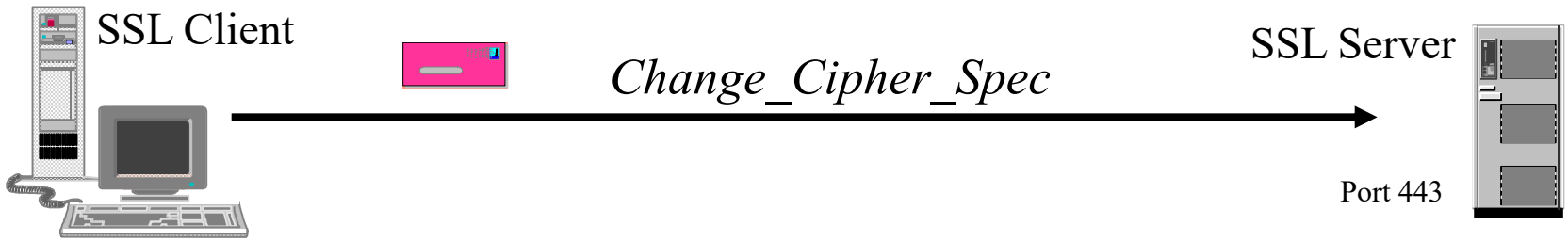
总结



第四阶段：完成

- 此阶段完成SSL安全连接的设置，主要包含以下消息：
  - 客户端：
    - 客户端发送的修改密码规范消息（Change_Cipher_Spec)
    - 客户端发送的完成消息（Finished)
  - 服务器
    - 服务器发送的修改密码规范消息（Change_Cipher_Spec)
    - 服务器发送的完成消息（Finished)

第四阶段：完成（客户->服务器）



The Client's Change cipher spec

说明：

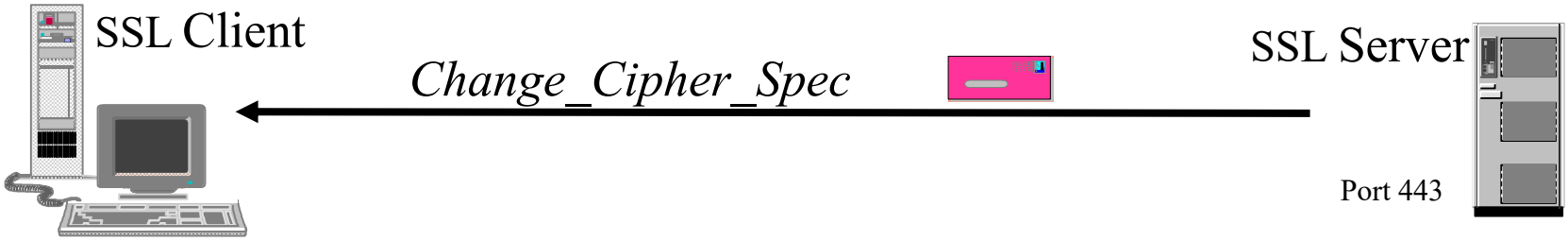
- 该消息不是握手协议的一部分，是使用修改密码规范协议来完成的 .它通知服务器启用新的密钥和算法。
- 在此之后，客户端使用新的算法、密钥和密码发送新的完成消息

第四阶段：完成（客户->服务器）

The Client Finish message

- The Client Finish message is composed of
  - The client authenticates the server with a message encrypted with the newly generated shared keys. （客户端用新建立的会话密钥来认证服务器）
  - This validates to the server that a secure connection has been created（确保连接的合法性）.

第四阶段：完成（客户->服务器）



The Server’s Change cipher spec

说明：

- 该消息不是握手协议的一部分，是使用修改密码规范协议来完成的 。它通知客户端启用新的密钥和算法。
- 在此之后，服务器端使用新的算法、密钥和密码发送新的完成消息

第四阶段：完成（客户->服务器）

Server Finish message

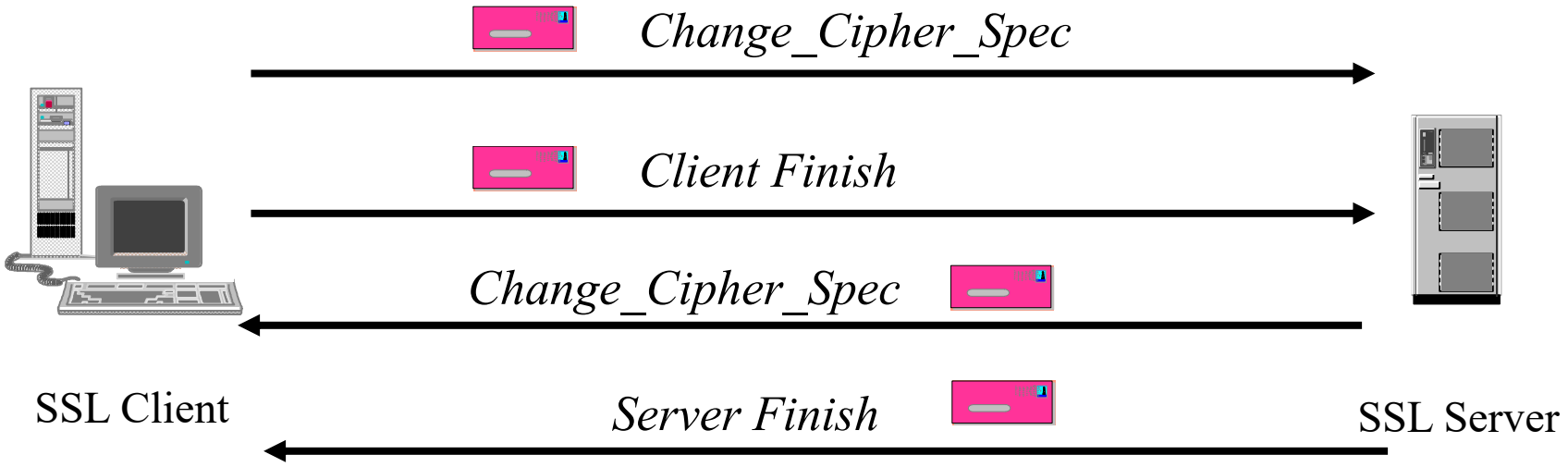
- The Server Finish message is composed of
  - The server authenticates the client with a message encrypted with the newly generated shared keys（服务器用新的会话密钥来认证客户）.
  - This validates to the client that a secure connection has been created(确保连接的合法性）

说明:

the Server and client can now begin to use their six shared keys for bulk data encryption utilizing the SSL Record Layer protocol（从此刻开始，客户端和服务器的记录层协议将启用新的密钥和算法进行数据加密）

第四阶段：完成（客户->服务器）

总结



问题

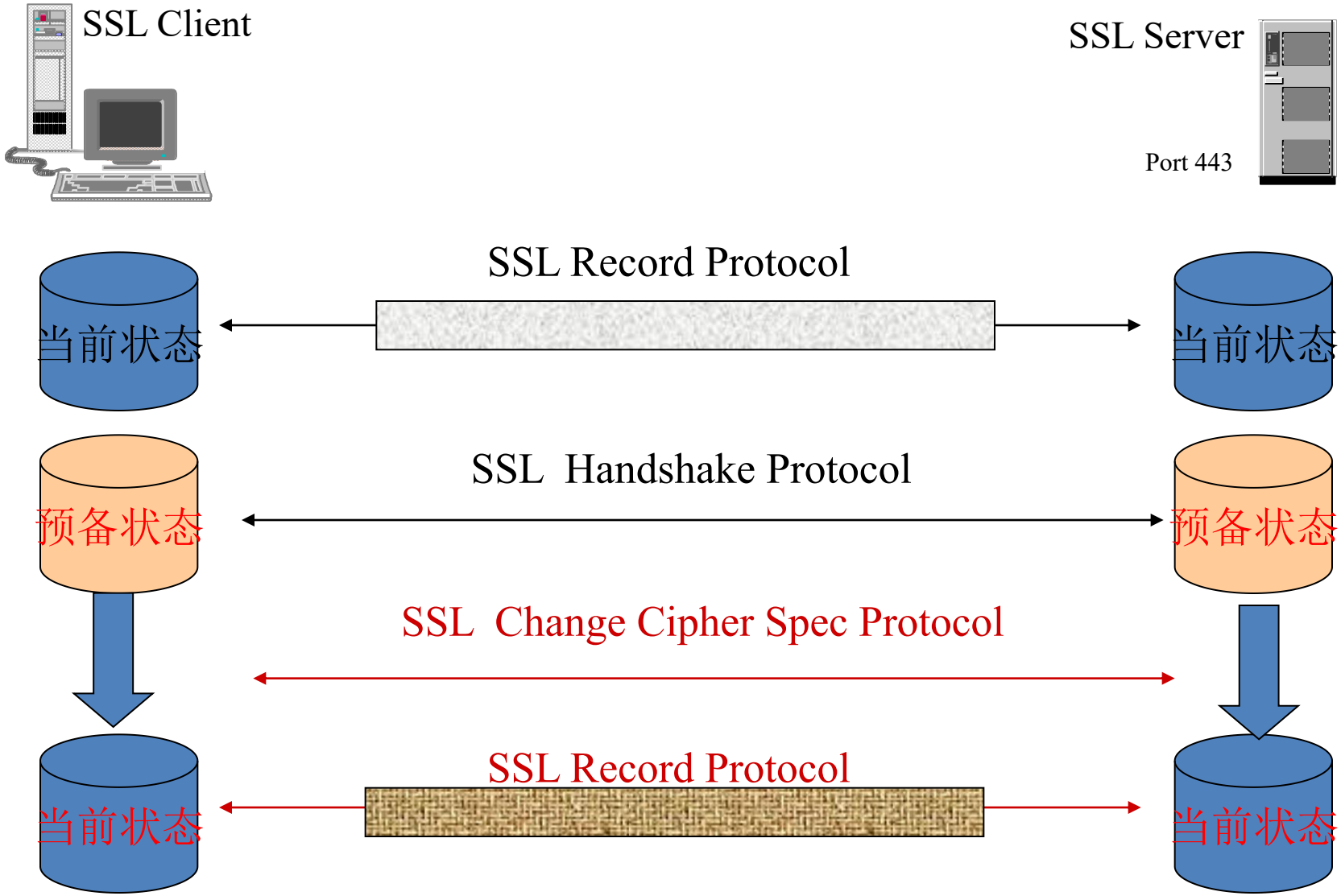
- Question：

  Why does SSL need the finish operation？

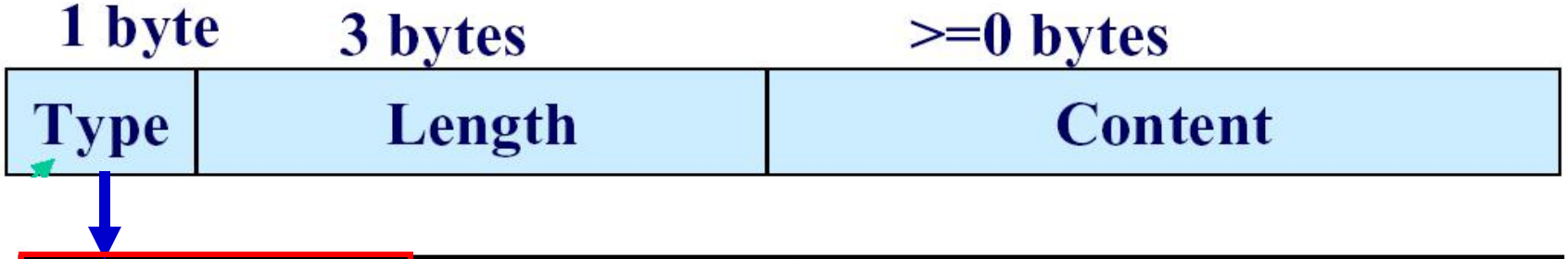
- Answer：

  从预备状态向当前状态改变

第四阶段：完成（客户->服务器）

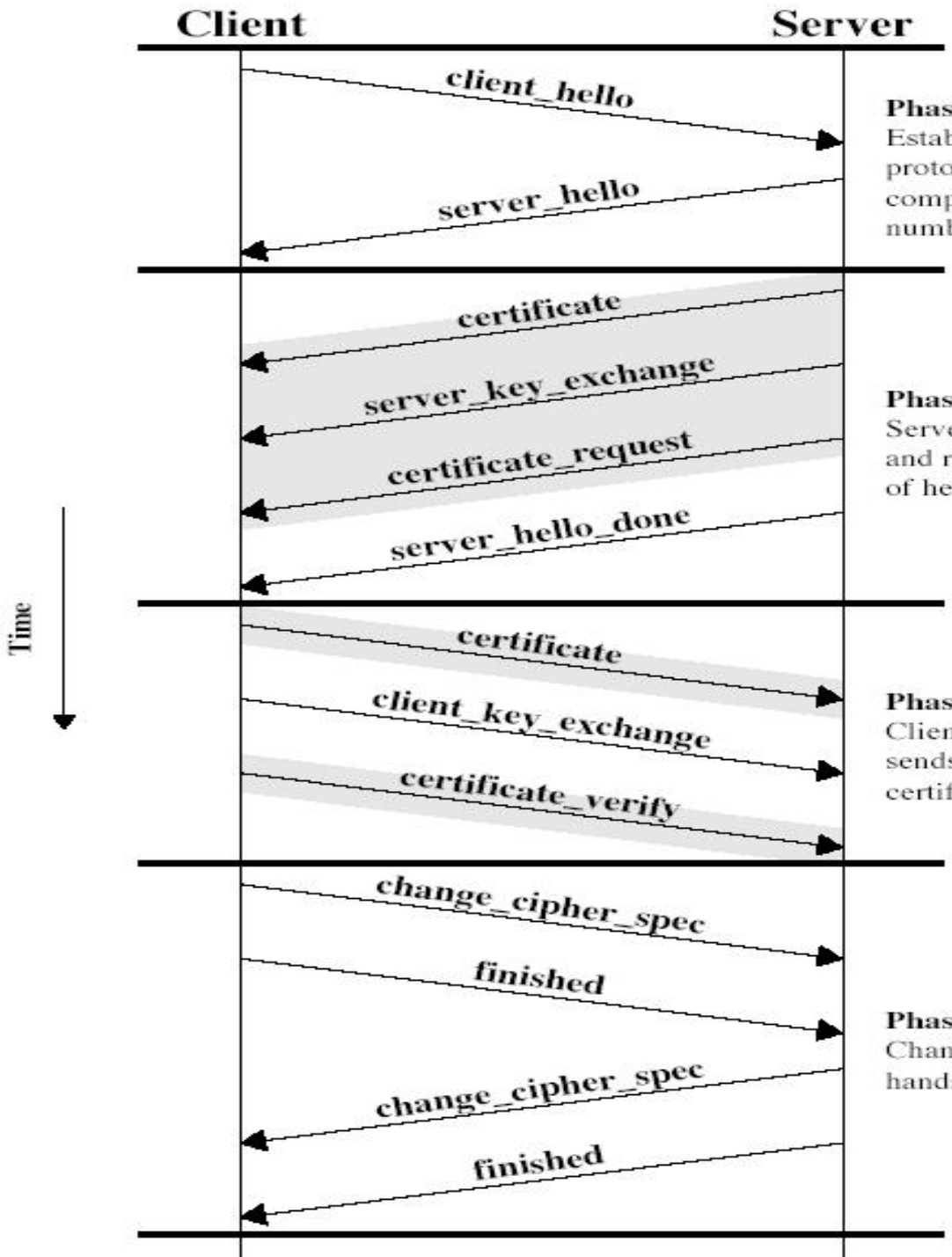


SSL握手协议报文—总结



消息	参数	描述
hello_request	Null	服务器发出此信息给客户端启动握手协议
client_hello	版本，随机数，会话id，密码参数，压缩方法	客户端发出client_hello启动SSL会话，该信息标识密码和压缩方法列表，服务器响应
server_hello		
certificate	X.509 v3证书链	服务器发出的向客户端验证自己的消息
server_key_exchange	参数，签名	密钥交换
certificate_request	类型，CAs	服务器要求客户端认证
server_done	Null	指示服务器的Hello消息发送完毕
certificate_verify	签名	对客户证书进行验证
client_key_exchange	参数，签名	密钥交换
finished	Hash值	验证密钥交换和鉴别过程是成功的

SSL握手协议报文一览总结



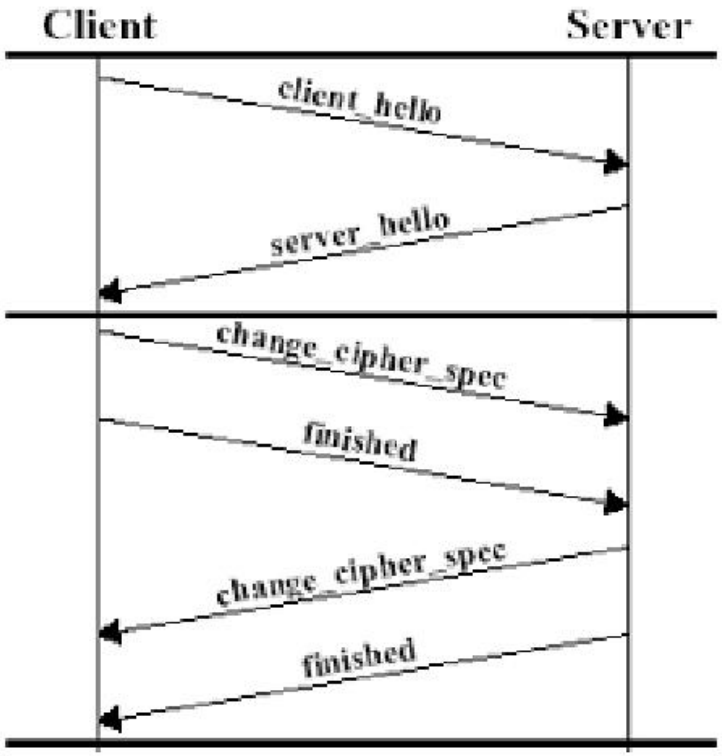
SSL握手协议报文—总结

- 在握手协议中，服务器证书请求和客户端证书请求都是可选的，因此SSL的握手过程实际蕴含三种验证方式：
  - 客户端和服务器均被认证
  - 只验证服务器
  - 客户端和服务器均未被认证（即匿名模式）

SSL握手协议报文—总结

重用一个SSL会话

- 客户和服务器在交换**hello**消息中，客户要求重用已有的**SSL**会话，服务器同意使用**cache**中的会话
  *** session id**
- 跳过第二第三阶段，直接把**SSL**会话中的参数传递给**SSL**记录层



问题

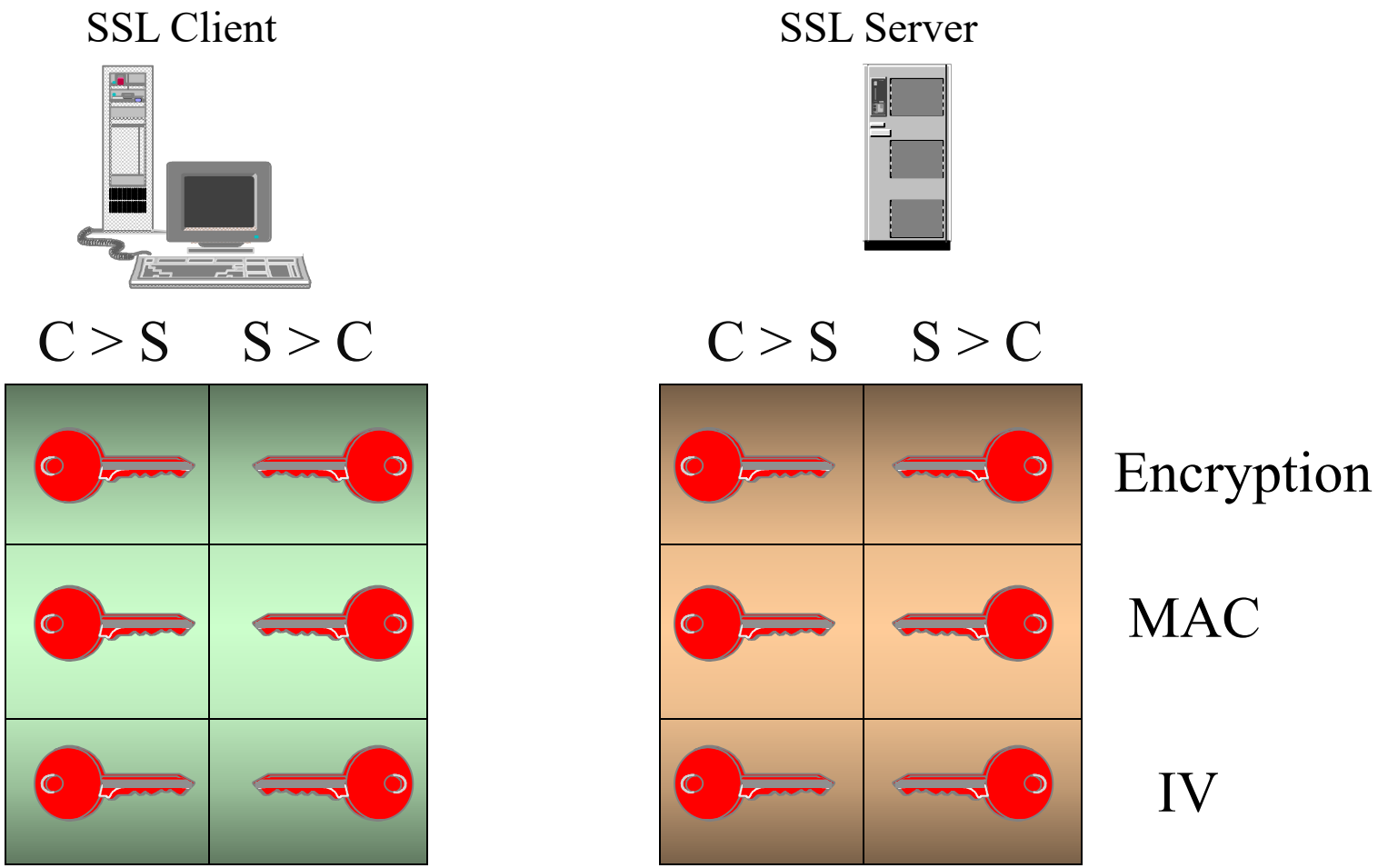
- Question：

在handshake协议中，只交换了一个秘密（称为预主秘密），那么SSL是如何得到数据通信所需要的加密密钥（即工作密钥）、MAC计算密钥和加密算法需要的IV值？

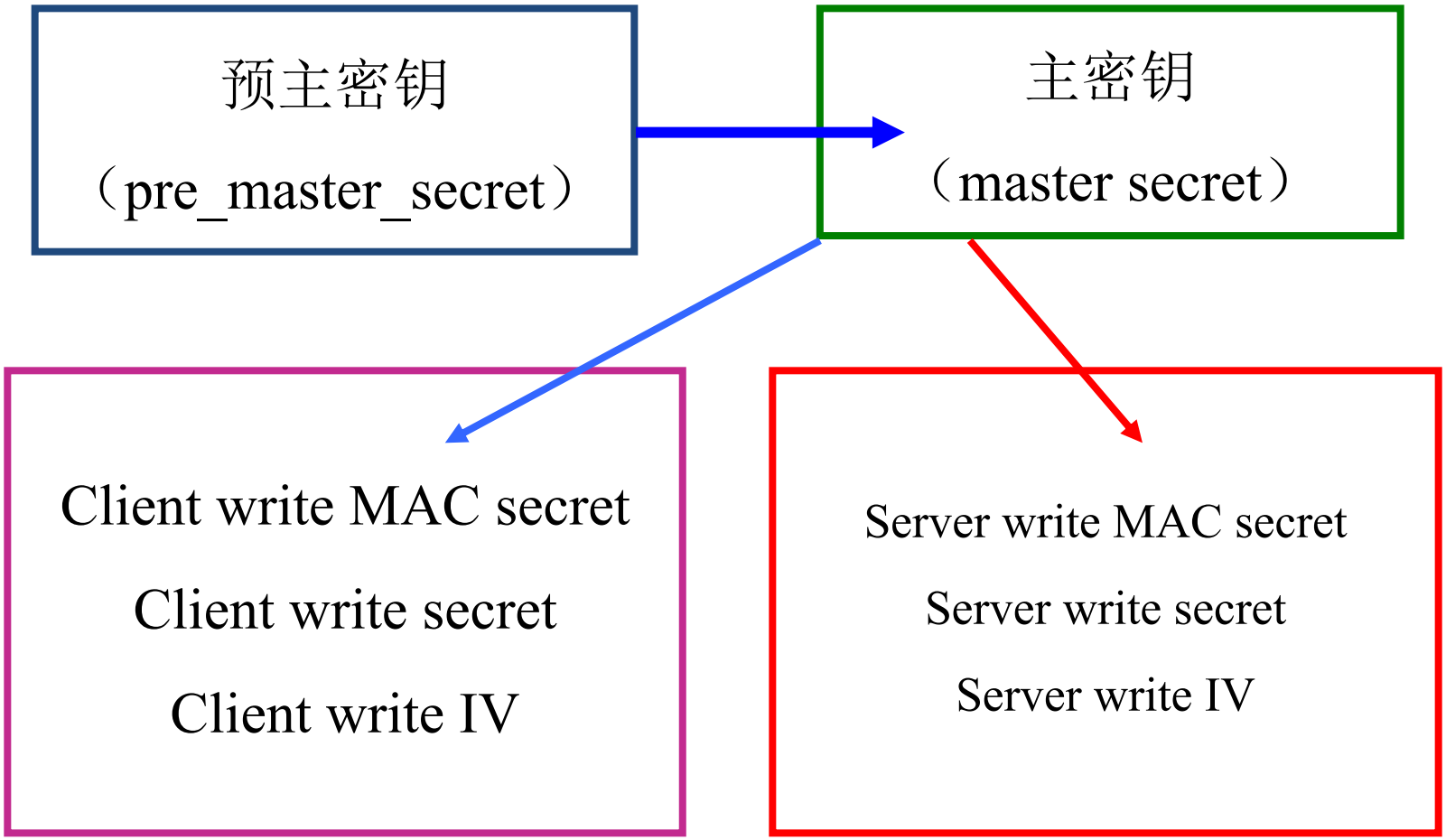
- Answer：

根据SSL的密钥计算规则。

SSL第2和3阶段交换结果：六对密钥



密码参数的生成


预主密钥
（pre_master_secret）
主密钥
（master secret）
Client write MAC secret
Client write secret
Client write IV
Server write MAC secret
Server write secret
Server write IV

主密钥的创建

- 经过Handshake Protocol之后，双方共享预主密钥（pre_master_secret），然后派生出一个48字节值（384 bit）的主秘密(Master Secret)，由客户机和服务器共享：
  - 预主密钥pre_master_secret 的交换方法主要有两种：
    - Diffie-Hellman：客户端和服务器同时生成Diffie-Hellman临时公钥，密钥交换后双方执行Diffie-Hellman计算，创建共享预主密钥
    - RSA：由客户端生成48字节的预主密钥，并用服务器的RSA公钥加密后发送到服务器，服务器用其密钥解密，得到密钥
- Master Secret 分两个步骤生成：
  - Exchange pre_master_secret（交换预主密钥）
  - Compute the master_secret（双方计算主密钥）

主密钥的创建

- For Asymmetric cryptographic computations（非对称密码算法计算）
  - The asymmetric algorithms are used in the handshake protocol to authenticate parties and to generate shared keys and secrets.
- The pre_master_secret should be deleted from memory once the master_secret has been computed（计算后需从内存中删除）

master_secret =

MD5(pre_master_secret + SHA('A' + pre_master_secret + ClientHello.random + ServerHello.random)) +
MD5(pre_master_secret + SHA('BB' + pre_master_secret + ClientHello.random + ServerHello.random)) +
MD5(pre_master_secret + SHA('CCC' + pre_master_secret + ClientHello.random + ServerHello.random));

主密钥的创建：DH交换

- 对于Diffie-Hellman
  - A conventional Diffie-Hellman computation is performed to compute the empheral K;
  - The negotiated key (K) is used as the pre_master_secret
  - and is converted into the master_secret, as specified above.

```
master_secret =
    MD5(pre_master_secret + SHA('A' + pre_master_secret +
        ClientHello.random + ServerHello.random)) +
    MD5(pre_master_secret + SHA('BB' + pre_master_secret +
        ClientHello.random + ServerHello.random)) +
    MD5(pre_master_secret + SHA('CCC' + pre_master_secret +
        ClientHello.random + ServerHello.random));
```

主密钥的创建（续）

- 对于 RSA
  - When RSA is used for server authentication and key exchange
    - A 48-byte pre_master_secret is generated by the client
    - Encrypted under the server's public key, and sent to the server
    - The server uses its private key to decrypt the pre_master_secret
    - Both parties then convert the pre_master_secret into the master_secret, as specified above.
- The pre_master_secret should be deleted from memory once the master_secret has been computed（计算后需从内存中删除）

```
master_secret =
    MD5(pre_master_secret + SHA('A' + pre_master_secret +
        ClientHello.random + ServerHello.random)) +
    MD5(pre_master_secret + SHA('BB' + pre_master_secret +
        ClientHello.random + ServerHello.random)) +
    MD5(pre_master_secret + SHA('CCC' + pre_master_secret +
        ClientHello.random + ServerHello.random));
```

其他密码参数的创建

- Symmetric cryptographic calculations and the CipherSpec（SSL中的密码规约参数）
  - The technique used to encrypt and verify the integrity of SSL records is specified by the currently active CipherSpec(密码规约）.
  - A typical example would be to encrypt data using DES and generate authentication codes using MD5.
  - The encryption and MAC algorithms are set to SSL_NULL_WITH_NULL_NULL at the beginning of the SSL Handshake Protocol, indicating that no message authentication or encryption is performed（SSL协议开始是并不进行加密或完整性验证）.
  - The handshake protocol is used to negotiate a more secure CipherSpec and to generate cryptographic keys（握手协议用来协商密码参数，并生成密钥）

其他密码参数的创建：加密密钥和MAC密钥

- Converting the master secret into keys and MAC secrets
  - The master secret is hashed into a sequence of secure bytes, which are assigned to the MAC secrets, keys, and non-export IVs required by the current CipherSpec（当前的密码参数）.
  - CipherSpecs require a client write MAC secret, a server write MAC secret, a client write key, a server write key, a client write IV, and a server write IV, which are generated from the master secret in that order.
  - Unused values, such as FORTEZZA keys communicated in the KeyExchange message, are empty.

其他密码参数的创建：加密密钥和MAC密钥（续）

- The following inputs are available to the key definition process:
  - MasterSecret
  - ClientHello.random
  - ServerHello.random
- When generating keys and MAC secrets
  - the master secret is used as an entropy source（熵源）
  - and the random values provide unencrypted salt material and IVs for exportable ciphers.

其他密码参数的创建：加密密钥和MAC密钥（续）

- To generate the key material（计算方法）：

```
key_block =
  MD5(master_secret + SHA(`A' + master_secret +
                          ServerHello.random +
                          ClientHello.random)) +
  MD5(master_secret + SHA(`BB' + master_secret +
                          ServerHello.random +
                          ClientHello.random)) +
  MD5(master_secret + SHA(`CCC' + master_secret +
                          ServerHello.random +
                          ClientHello.random)) + [...];
```

until enough output has been generated.

其他密码参数的创建：加密密钥和MAC密钥（续）

- Then the key_block is partitioned as follows.

client_write_MAC_secret[CipherSpec.hash_size]

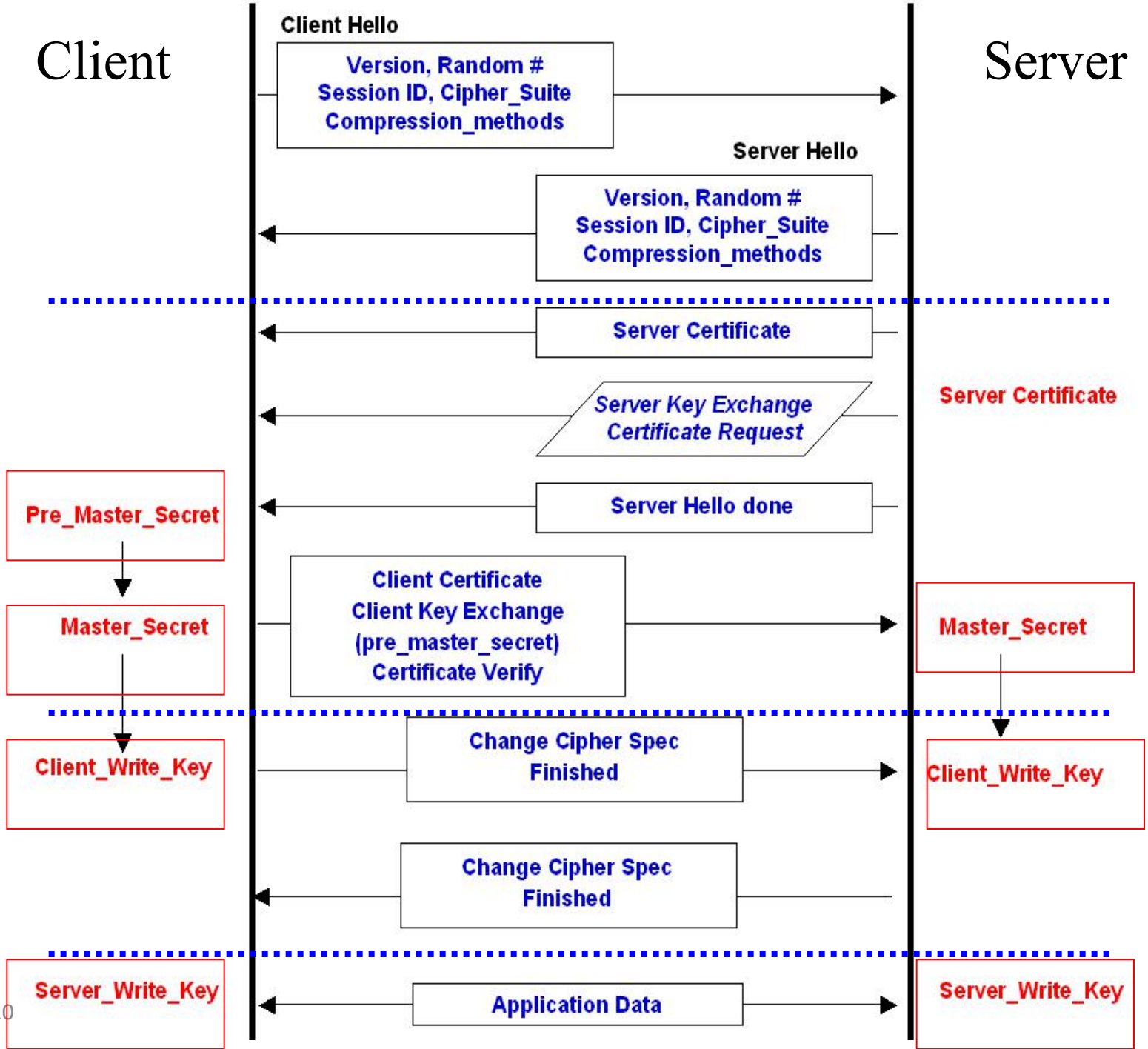
server_write_MAC_secret[CipherSpec.hash_size]

client_write_key[CipherSpec.key_material]

server_write_key[CipherSpec.key_material]

client_write_IV[CipherSpec.IV_size] /* non-export ciphers */

server_write_IV[CipherSpec.IV_size] /* non-export ciphers */


Client
Server
Client Hello
Version, Random #
Session ID, Cipher_Suite
Compression_methods
Server Hello
Version, Random #
Session ID, Cipher_Suite
Compression_methods
Server Certificate
Server Key Exchange
Certificate Request
Server Hello done
Server Certificate
Pre_Master_Secret
Master_Secret
Client Certificate
Client Key Exchange
(pre_master_secret)
Certificate Verify
Master_Secret
Client_Write_Key
Change Cipher Spec
Finished
Client_Write_Key
Change Cipher Spec
Finished
Server_Write_Key
Application Data
Server_Write_Key

SSL的具体实现

SSL实现

- OpenSSL, 最新0.9.7a, 实现了SSLv2,SSLv3, TLSv1.0
  - Openssl —— a command line tool.
  - ssl(3) —— the OpenSSL SSL/TLS library.
  - crypto(3)—— the OpenSSL Crypto library.
  - URL: http://www.openssl.org
- SSLeay
  - http://www2.psy.uq.edu.au/~ftp/Crypto/
- Internet号码分配当局已经为具备SSL功能的应用分配了固定的端口号,
  - 例如带SSL的HTTP(https)被分配以端口号443
  - 带SSL的SMTP(ssmtp)被分配以端口号465
  - 带SSL的NNTP(snntp)被分配以端口号563

SSL与Web安全

- HTTP

  Clear text communication.

- HTTPS

  RFC 2817 and RFC 2818 describes HTTP over TLS

SSL与web安全

- First implementation of HTTP over SSL was issued in 1995 by Netscape
- Netscape was prevented from choosing shttp
  - S-HTTP（secure HTTP） another protocol for securing messages
  - Treats each request – response pair as a single unit
  - Allows protect different messages between server-client differently

HTTP connection behavior

- HTTP 1.0
  - Close connection after each response
  - Images: require response/request pair
- HTTP 1.1
  - Connection : Keep-Alive (persistent connection)
  - Some browsers still open number of connections to load page with images in parallel
  - Large number of parallel connections is optimized by SSL session resumption.

Basic Technologies

- HTTP (Hypertext Transfer Protocol)
  - First protocol to use SSL
  - Request/Response structure
  - Most browsers speak http protocol
- HTML (Hypertext Markup Language)
  - Offers the ability to structure the document
  - Provide links to move to another documents
- URL (Uniform Resource Locator)

Security of web—Summary

	威胁	后果	对策
完整性	修改用户的数据 特洛伊木马浏览器 修改内存 修改传输的数据流	信息丢失 机器暴露 其它所有威胁的弱点	加密校验和
保密性	网上偷听 从服务器处偷信息 从客户端处偷信息 关于网络配置的信息 关于客户连接的信息	丢失信息 丢失隐私	加密、Web 代理
业务否决	-中断用户连接 -用伪造的威胁淹没服务器 塞满硬盘或内存 攻击 DNS 隔离服务器	中断 骚扰 阻止用户完成正常工作	难以防范
鉴别	冒充合法用户 数据伪造	以假乱真 误信错误信息	加密技术

Web安全的特点

- 提供双向的服务，攻击防范能力脆弱
- 作为可视化窗口和商业交互平台，提供多种服务，事关声誉
- 底层软件庞大，如apache约10M，历来是漏洞之最，攻击手段最多
- 如果被攻破可能导致成为进入企业的跳板
- 配置比较复杂

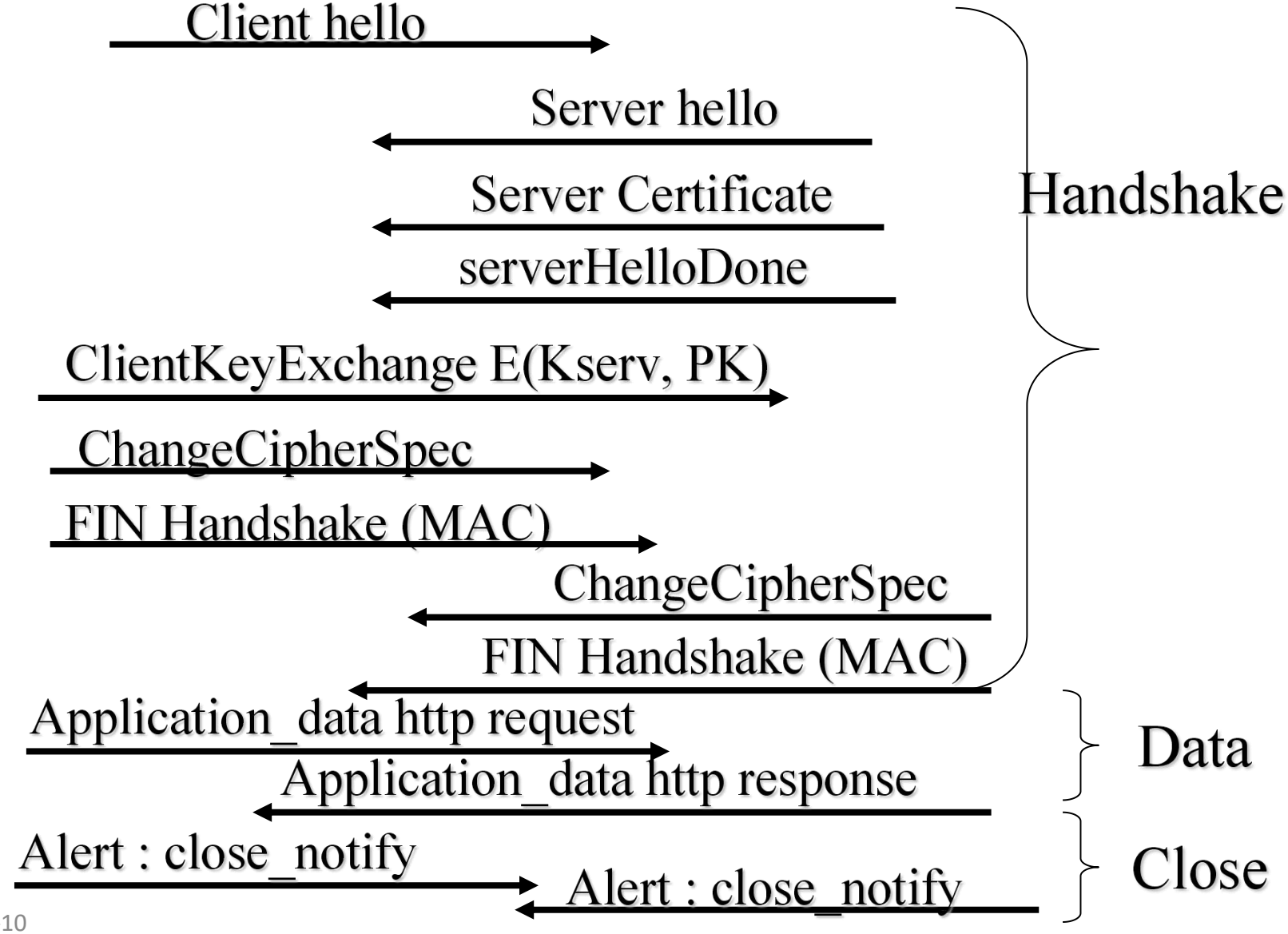
Web安全的组成部分

- Browser 安全
- Web Server安全
- Browser 与Web Server之间网络通信安全

Web安全方案

- 网络层：IPSec
- 传输层：SSL/TLS
- 应用层：SET/SHTTP

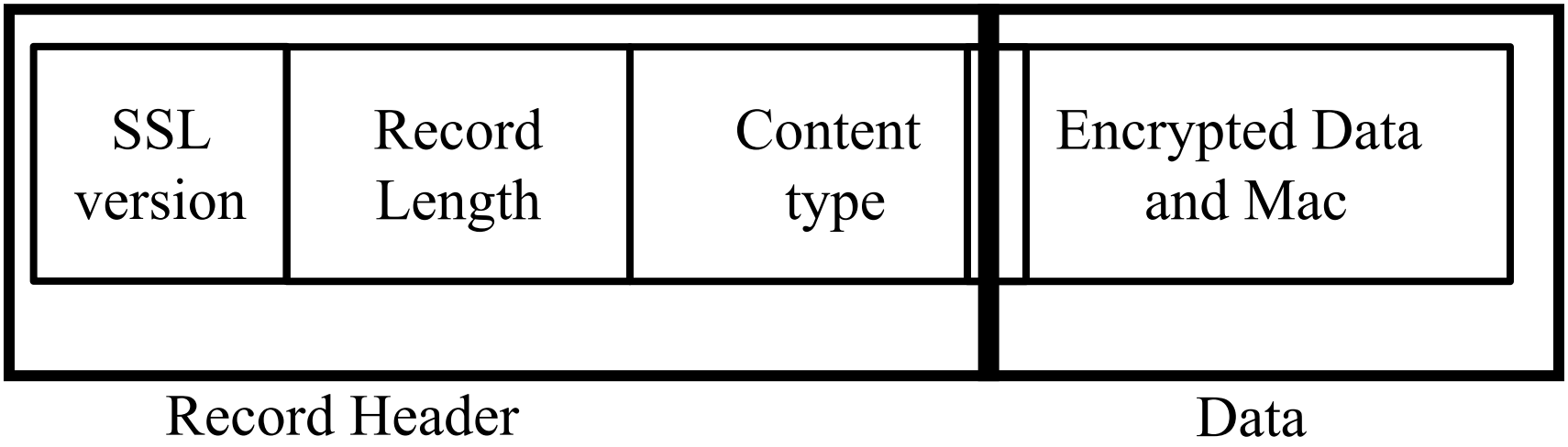
HTTP Over SSL(TLS): Protocol



HTTP Over SSL(TLS): Overview

- Client makes connection to the server
- Handshakes SSL
- Transmits data over SSL channel
  - Assigned port 443
  - Special configuration of the server needed (mod_ssl), possible to configure another port
- No client data is sent until SSL connection established
- Long web pages require spanning multiple SSL records

HTTP Over SSL(TLS): Protocol Format



- Content types:
  - ChangeCipherSpec
  - Handshake
  - Application Data
  - Alert (signaling error and closure)

HTTP Over SSL(TLS): Connection closure

- When one side initiates closure
  - Must send close_notify
  - Optionally withholds its TCP FIN until close_notify received from other side.
  - If TCP FIN received without close_notify this may indicate possible attack

HTTP Over SSL(TLS): End-point Authentication

- When user's browser receives certificate
  - CA is a known CA
  - Browser compares:
    - Name of the site that it intended to connect with
    - Hostname in certificate

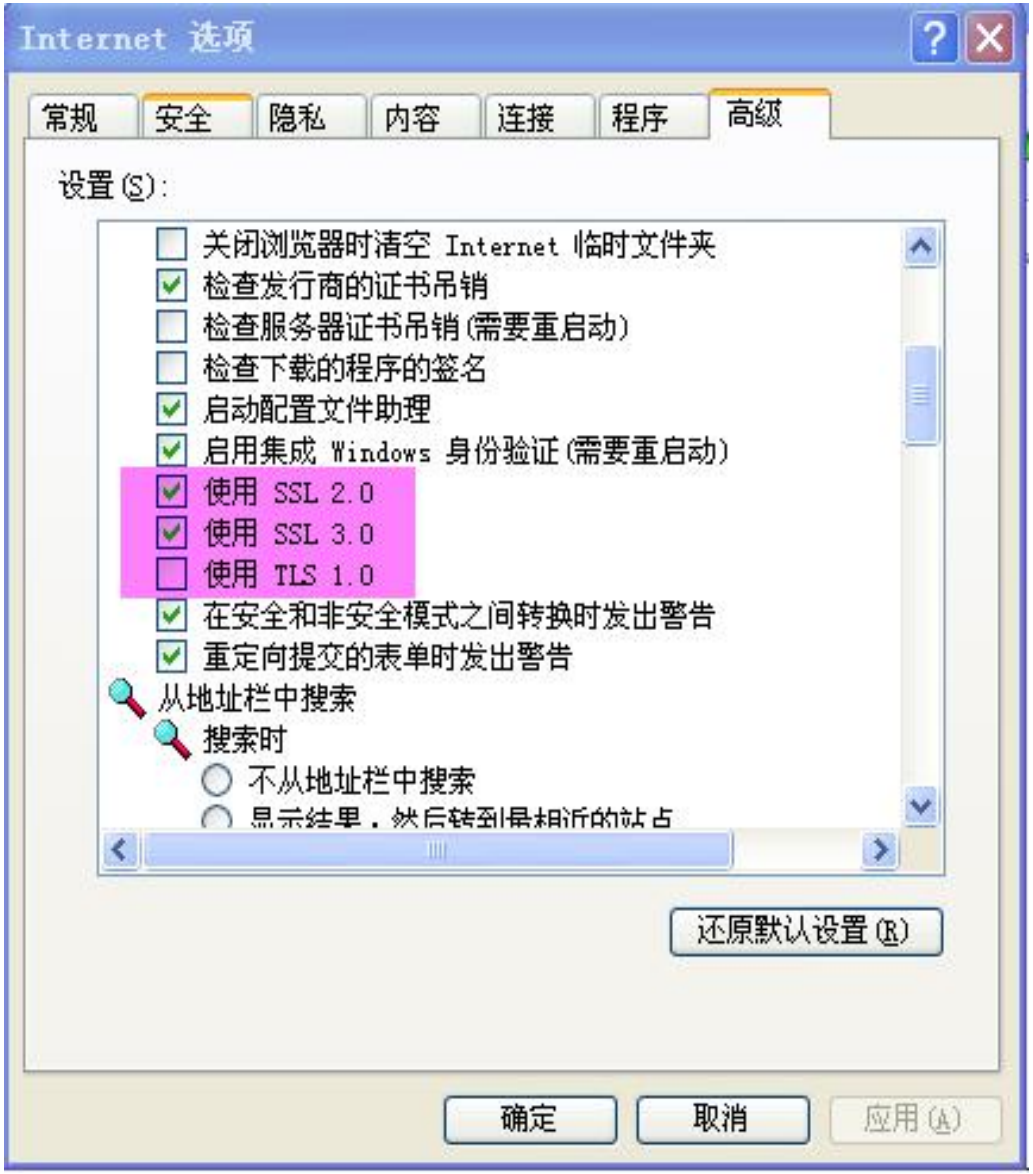
HTTP Over SSL(TLS): Session Resumption

- Session resumption（会话恢复）
  - Minimizes the number of SSL handshakes, within the same session
  - sides speak with already established keys
  - If the close was premature（过早） in no case SSL implementation should resume session

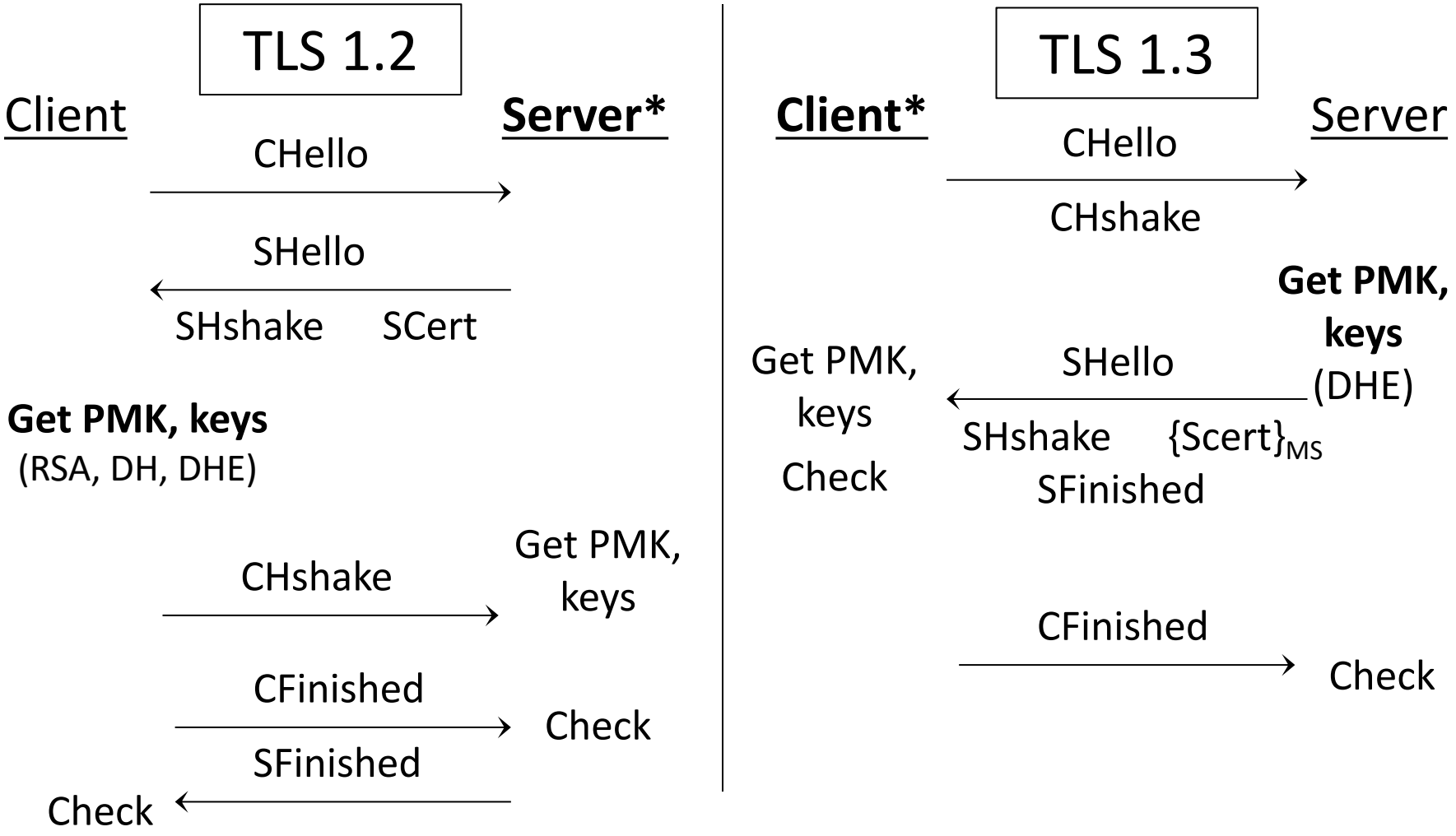
HTTP Over SSL(TLS): Error Handling

- Servers logs errors. Later maybe examined by administrator
- Clients receive via dialog boxes
  - SSL implementation should:
    - Report legitimate errors（报告合法错误）
    - Not overwhelm the user (hide unimportant)（不致过渡影响用户）

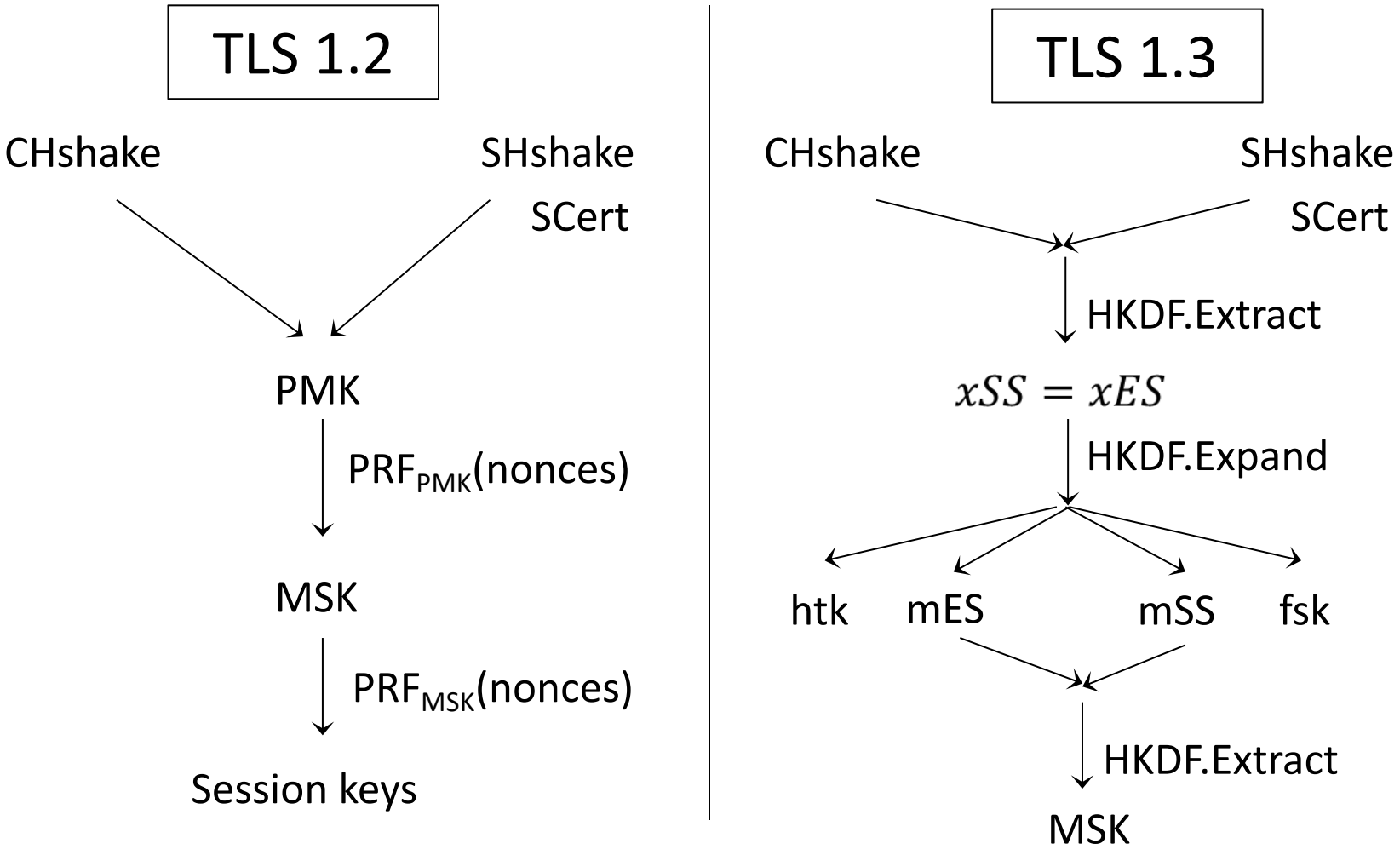
HTTP Over SSL(TLS): SSL in IE



TLS 1.3 (basic)


TLS 1.2
Client
Server*
CHello
SHello
SHshake
SCert
Get PMK, keys
(RSA, DH, DHE)
CHshake
Get PMK, keys
CFinished
Check
SFinished
Check
TLS 1.3
Client*
Server
CHello
CHshake
Get PMK, keys
(DHE)
Get PMK, keys
SHello
SHshake
{Scert}MS
Check
SFinished
CFinished
Check

Key Derivation



More Features of TLS 1.3

- Multiple operating modes
  - If the client wants to resume sessions: 0 RTT
  - If the client knows the server's PK: different resumption
  - Full operation mode present too
- Restricted cipher suites: no RC4, no CBC, no RSA
- Independence of htk, fsk, msk:
  - Better provable security
- Session hash used in key derivation:
  - Better freshness
  - Better security

What we Have and What we Want

Desired Changes	Actual Changes
• Kill backward compatibility	• Advertised for TLS 1.3
• Kill CBC mode	• Done; de-facto is AEAD
• Kill RC4	• Done
• Kill renegotiation	• Modified it; unclear consequences
• Kill finite fields (keep EC)	• Done
• Kill RSA mode	• Done
• Kill data compression	• Partial

请各位同学交流指正！