志存高远 责任为先

第6章 传输层安全

地址：赣州市红旗大道86号 信息工程学院
网址：www.jxust.edu.cn
邮编：341000

目录/Contents

学习目标

- 总结Web安全威胁和Web流量安全手段
- 了解安全套接层
- 理解安全套接层和传输层安全的区别
- 了解HTTPS协议
- 了解SSH协议

Part

Web的安全需求

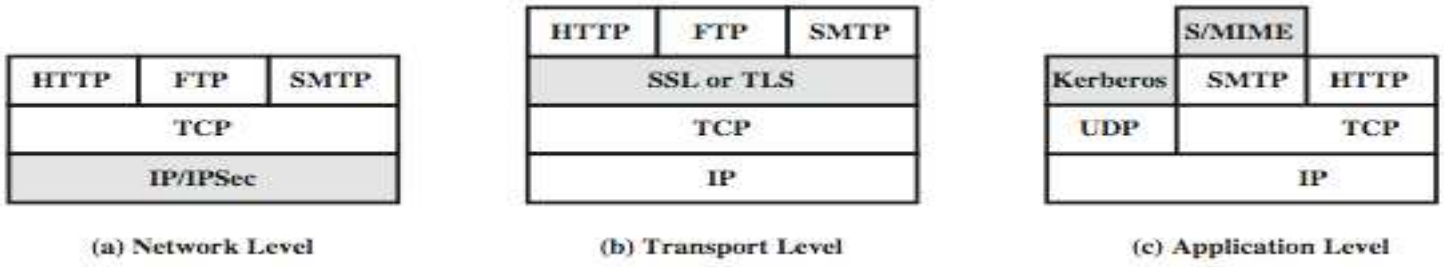
6.1 Web的安全需求

- World Wide Web就是互联网上最重要、最广泛的应用之一
- IPSec是提供Web安全性的一种方法，优点在于，它对终端用户和应用均是透明的，并且提供通用的解决方案。
- 另一种解决方案是在TCP之上实现安全性

6.1.1 Web安全威胁

	威胁	后果	对策
完整性	•修改用户数据 •特洛伊木马浏览器 •修改内存 •修改传输中的消息流量	•信息丢失 •设备受损 •对所有其他威胁的脆弱性	密码校验和
机密性	•在网上窃听 •从服务器窃取信息 •从客户端窃取数据 •窃取网络配置的信息 •窃取客户端与服务器通信的信息	•信息丢失 •秘密失去	加密, Web 委托代理
拒绝服务	•破坏用户线程 •用假消息使机器溢出 •填满磁盘或内存 •通过DNS攻击独立机器	•中断 •干扰 •防止用户完成工作	难于防止
认证	•冒充合法用户 •伪造数据	•用户错误 •相信虚假信息有效	密码技术

6.1.2 Web流量安全方法



TCP/IP协议栈中的安全设施的相对位置

✓Ipsec的好处是对终端用户和应用是透明的，能提供一种一般性的解决方案。如图（a）

✓SSL或TLS都是可执行协议软件包的一部分，从而对应用是透明的，Netscape进而IE都配置了SSL。大部分Web服务器都实现了该协议。如图（b）。

✓特定安全服务在特定的应用中体现，如图（c）。

02
Part

安全套接字层（SSL）

目录/Contents

02.1
Part

安全套接字SSL简介

安全套接字SSL简介

- 最被广泛使用的安全网络协议
  - 几乎被所有浏览器和网站服务器支持
  - https
  - 每年通过SSL协议支持的网络交易以亿万美元计
- 协议设计：[Woo 1994]
  - 协议实现:Netscape
- 衍生协议-传输层安全协议TLS: RFC 2246

安全套接字SSL简介

- **提供的服务**
  - **Confidentiality -保密性**
  - **Integrity -信息完整性**
  - **Authentication -认证**
- **最初目的：**
  - **电商网上交易的信息安全需求**
  - **需要加密解密协议（尤其对信用卡信息)**
  - **网站的真实性认证**
  - **（可选择的）用户真实性认证**
  - **跟新的商品提供商做交易手续最简化**

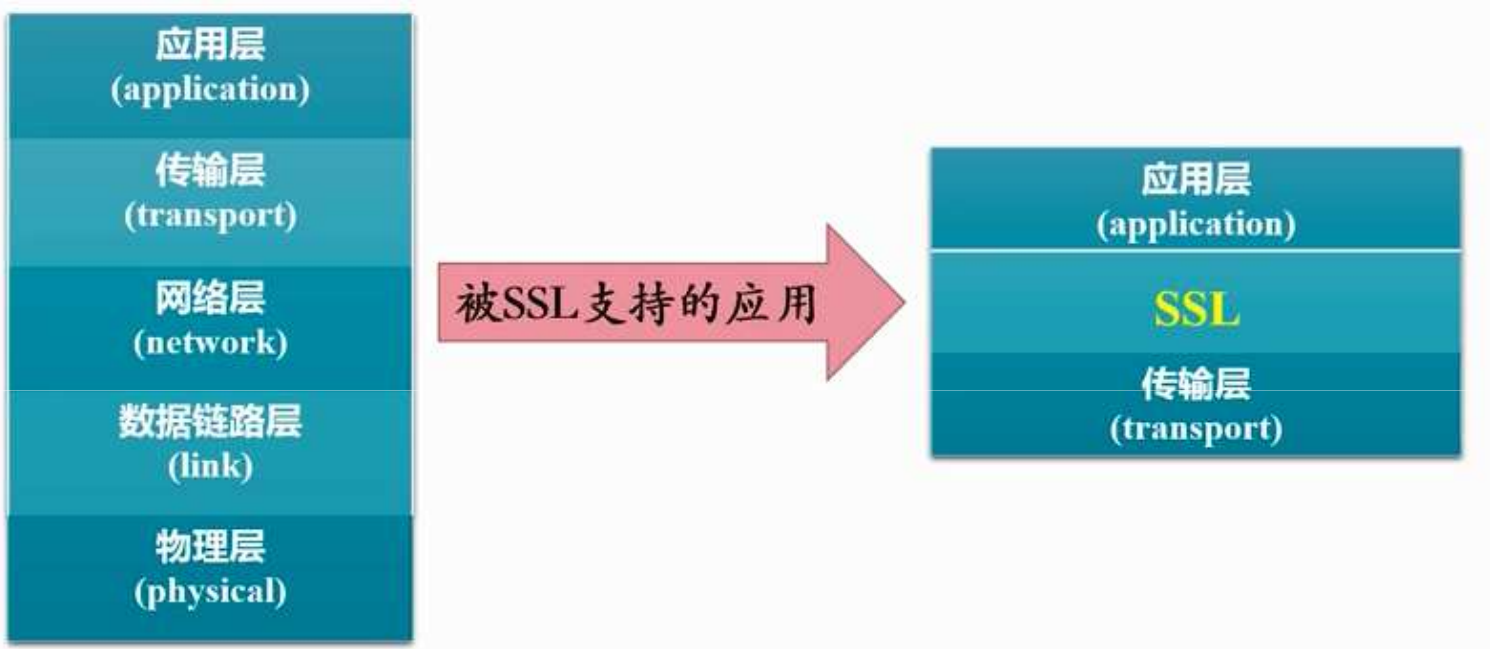
6.2 安全套接字层（SSL）

- **SSL（Secure Socket Layer，安全套接层）协议**
  - Netscape公司于1994年最先提出来的
  - 被设计成使用TCP来提供一种可靠的端到端的安全服务，是一种基于会话的加密和认证的协议
  - 在客户和服务器之间提供了一个安全的管道
    - 为了防止客户/服务器应用中的监听、篡改、消息伪造等，SSL提供了服务器认证和可选的客户端认证
    - 通过在两个实体间建立一个共享的秘钥，SSL提供保密性
- **SSL工作在传输层（TCP）和应用层之间**
  - SSL独立于应用层协议，高层协议可基于SSL进行透明的传输

SSL协议提供的主要服务

- **加密处理**
  - 加密数据以防止数据中途被窃取；
- **维护数据的完整性**
  - 确保数据在传输过程中不被改变。
- **认证服务**
  - 认证用户（可选）和服务器，确保数据发送到正确的客户机（可选）和服务器；

SSL和TCP/IP



- SSL 给应用层提供服务接口（application programming interface _ API)
- 很多编程语言有写好的程序模块，比如Python、C和Java SSL libraries/classes

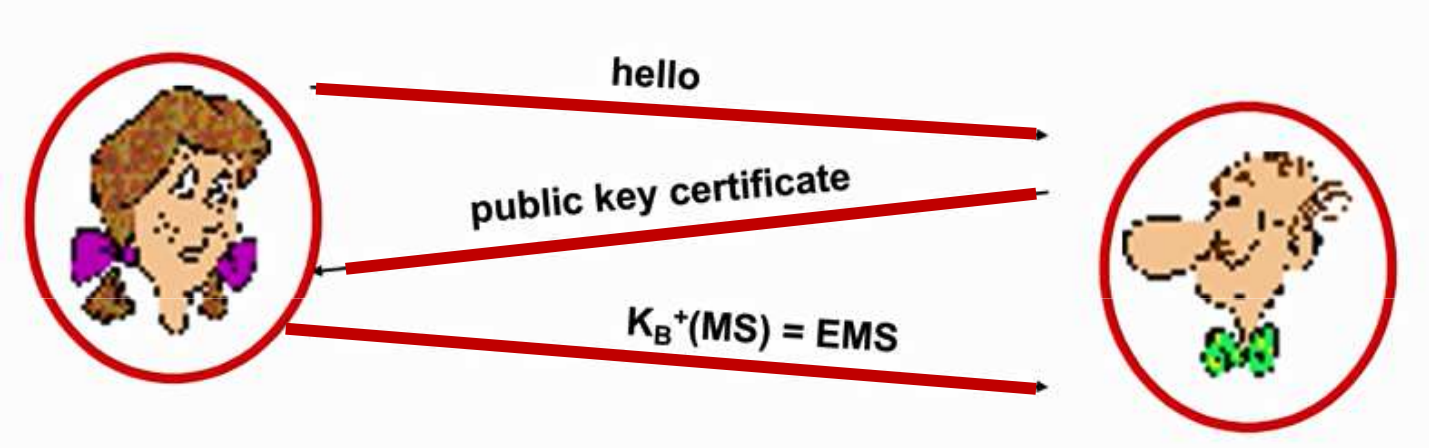
过程类似与PGP（第8章）

- **但更高一层的需求是**
  - **收发双方需要能随时交换新的信息，并且信息是双向的**
  - **需要对一个链接里交换的全部信息进行加密**
  - **要对收发双方进行认证：“握手”阶段（handshake phase)**

最简SSL协议设计

- **(收发双方）握手：**
  - **Alice和Bob用他们各自的电子证书、私钥去加密共享密钥，以及向对方实现身份认证**
- **密钥衍生：**
  - **Alice和Bob用他们共享的秘密信息或密码衍生出一组密钥，以备协议后续使用**
- **数据传输：**
  - **需要传输的数据被分割成一组数据记录段**
- **终断链接：**
  - **当数据传输结束，链接需要被终断时，用特定终断信息终断链接**

最简SSL协议：握手



- **MS**：**Master Key**
- **EMS**: **Encrypted Master Key**

最简SSL协议：密钥衍生

- **在编码解码协议里“反复使用同一个密钥”很不安全**
  - **解决方法：对信息认证和加密过程使用不同的密钥、密钥更新**
- **四种不同的密钥：**
  - **>K_C=对从用户（client)到服务器（server)的信息的加密密钥**
  - **>M_c=从用户端到服务器端的MAC密钥**
  - **>K_S=对从服务器（server)到用户（client)的信息的加密密钥**
  - **>M_S=从服务器端到用户端的MAC密钥**
- **上述四种密钥都是通过将Master Secret和其它随机数字输入密钥衍生函数 (Key Derivation Function — KDF)衍生而来**

最简SSL：数据记录段

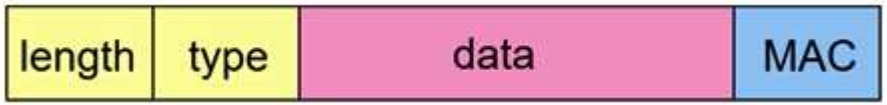
- **为什么不对数据一边加密一边写入TCP的端口？**
  - **如何计算和放置MAC位？**
  - **如果把MAC放在数据最末端，无法在数据全部传输完成之前做信息完整性（integrity)的检查**
  - **比如，对短信（instant messaging)协议，如何在接收方收到并显示信息之前对数据进行完整性 (integrity)检查？**
- **解决方案，把数据分割成一系列数据记录段**
  - **对每一个记录段（record)计算自己的MAC**
  - **接收方可以对每一个数据记录段进行信息完整性检查**
- **需要注意:对每一个数据记录段，接收方都需要区别MAC位和数据**
  - **根据实际应用，数据记录段的长度有可能不等**

最简SSL：sequence number

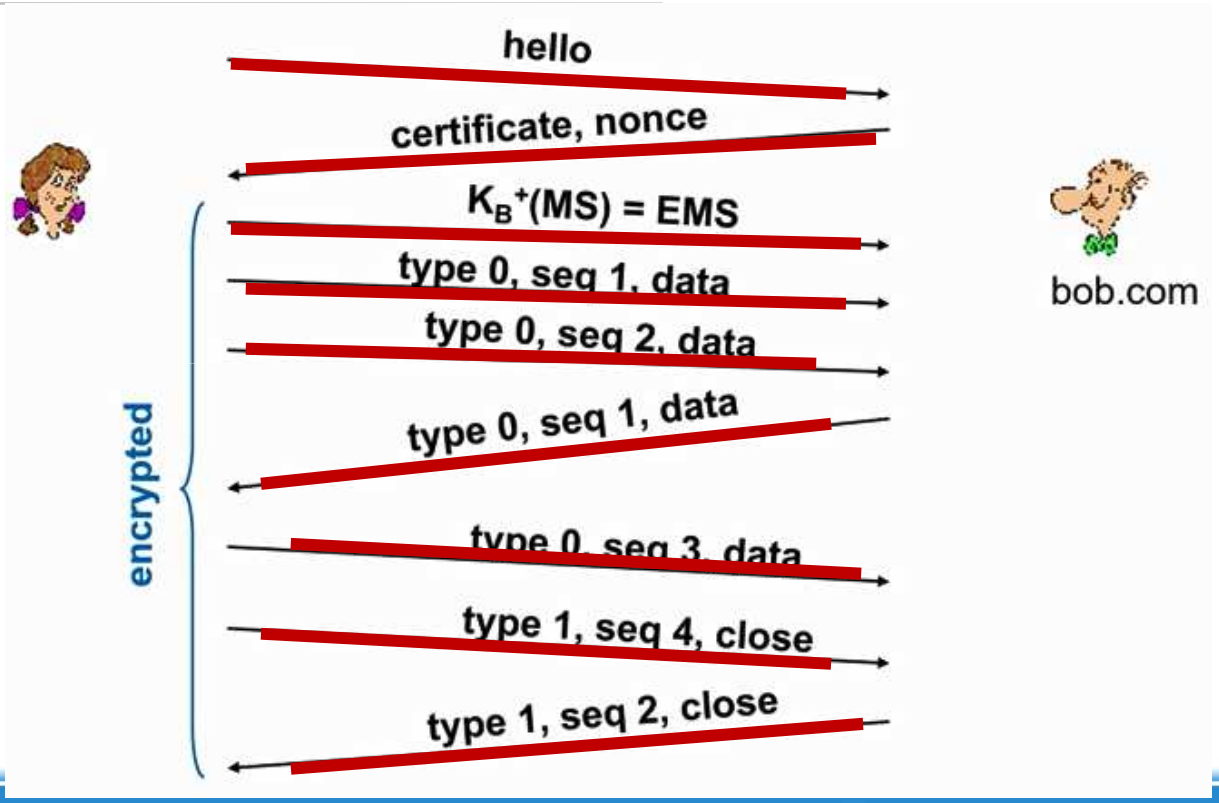
- **安全问题:攻击者可以进行录播攻击**
- **解决方法:在MAC里设置序列号**
  - **MAC = MAC(M_X, sequence ||data)**
  - **注意：这里并没有在协议信息里保留数据为存储序列号，而是把序列号直接加到MAC里**
- **进一步，如果:攻击者录播所有数据段**
- **解决方法：使用nonce**

最简SSL：控制信息

- **问题:截断攻击**
  - 攻击者伪造TCP终断连接信息（connection close segment)
  - 收发双方都以为不再有数据再传过来了，而事实是还有更多的数据要被传输
- **解决方案:标记数据记录段类别**
  - 标记位为0就指这一个记录段是数据段
  - 标记位为1就意味着信息就此结束，连接将被终断
- MAC = MAC(M_X，sequence || type || data)



最简SSL：总结


hello
certificate, nonce
$K_B^+(MS) = EMS$
type 0, seq 1, data
type 0, seq 2, data
type 0, seq 1, data
type 0, seq 3, data
type 1, seq 4, close
type 1, seq 2, close
encrypted
bob.com

最简SSL：问题

- **信息包中的每一数据段（field)都有多长?**
- **选择什么样的加密协议？**
- **收发方是否需要协商？**
  - **让用户方和服务器有选择不同加密算法的可能**
  - **让用户方和服务器一起，在收发数据之前，协商决定用什么算法**

SSL加密算法库

- **加密算法库**
  - **公共密钥算法**
  - **共享密钥算法**
  - **哈希函数算法（MAC)**
- **SSL支持加密算法库中的多种算法**
- **收发双方协商：什么步骤使用何种加密算法**
  - **用户端提供可能的算法**
  - **服务器端进行选择**

SSL加密算法库

- **普通SSL共享密钥算法**
  - **DES - Data Encryption Standard: block**
  - **3DES - Triple strength: block**
  - **RC2 — Rivest Cipher 2: block**
  - **RC4 - Rivest Cipher 4: stream**
- **SSL公共密钥算法**
  - **RSA**

SSL握手阶段

- **握手阶段的功能：**
  - **1对服务器认证**
  - **2建立密钥(establish keys)**
  - **3对用户进行认证(optional)**

SSL握手阶段（续）

- 1用户端给服务器发出一系列算法以及用户端nonce
- 2服务器在用户端发出的算法库中选择适当加密算法；并发回所选的算法&电子证书&服务器端的nonce
- 3用户端对服务器电子证书（certificate)做认证后，解密出服务器的公钥，生 成pre-master-secret，并用服务器的公钥对pre-master-secret进行加密，之后发给服务器

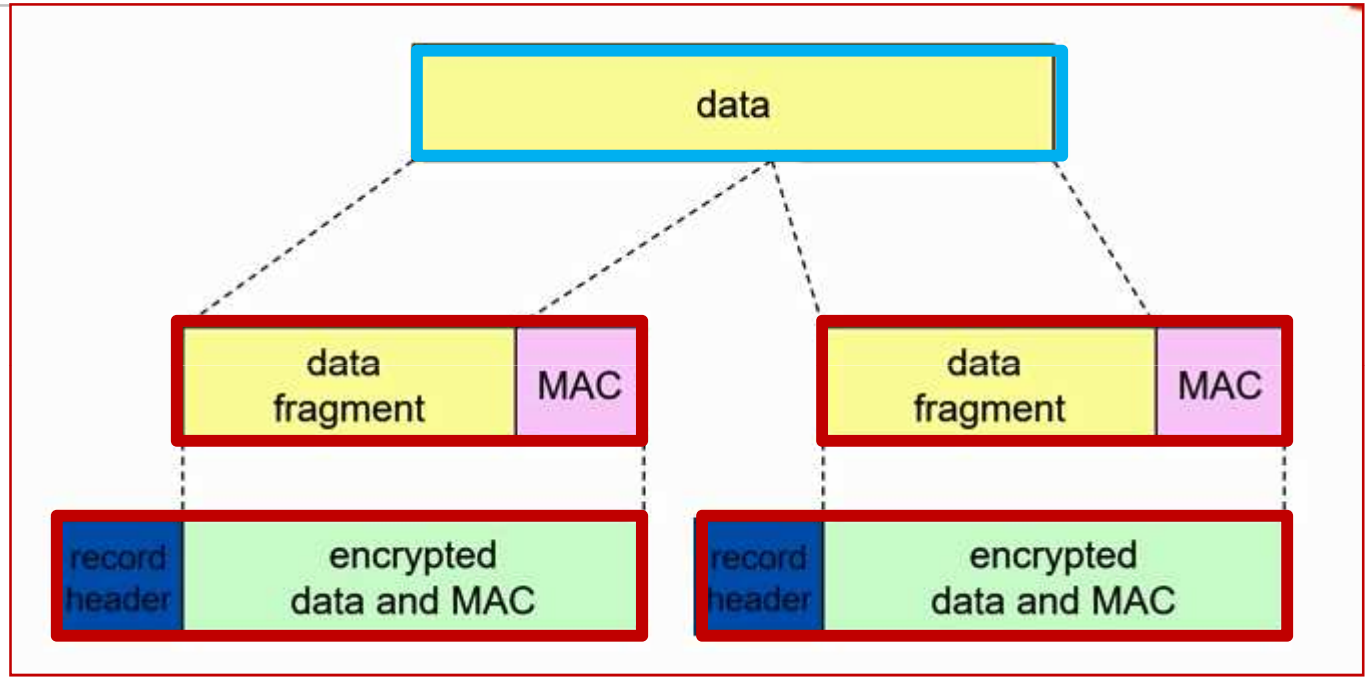
SSL握手阶段（续）

- **4用户端和服务端各自独立地，用pre-master-secret、nonce和事先设定的算法计算加密密钥和MAC密钥**
- **5用户端在每一条“握手”信息里都附上信息认证码（MAC）**
- **6服务器对所有“握手”信息发一次信息认证码**

SSL握手阶段（续）

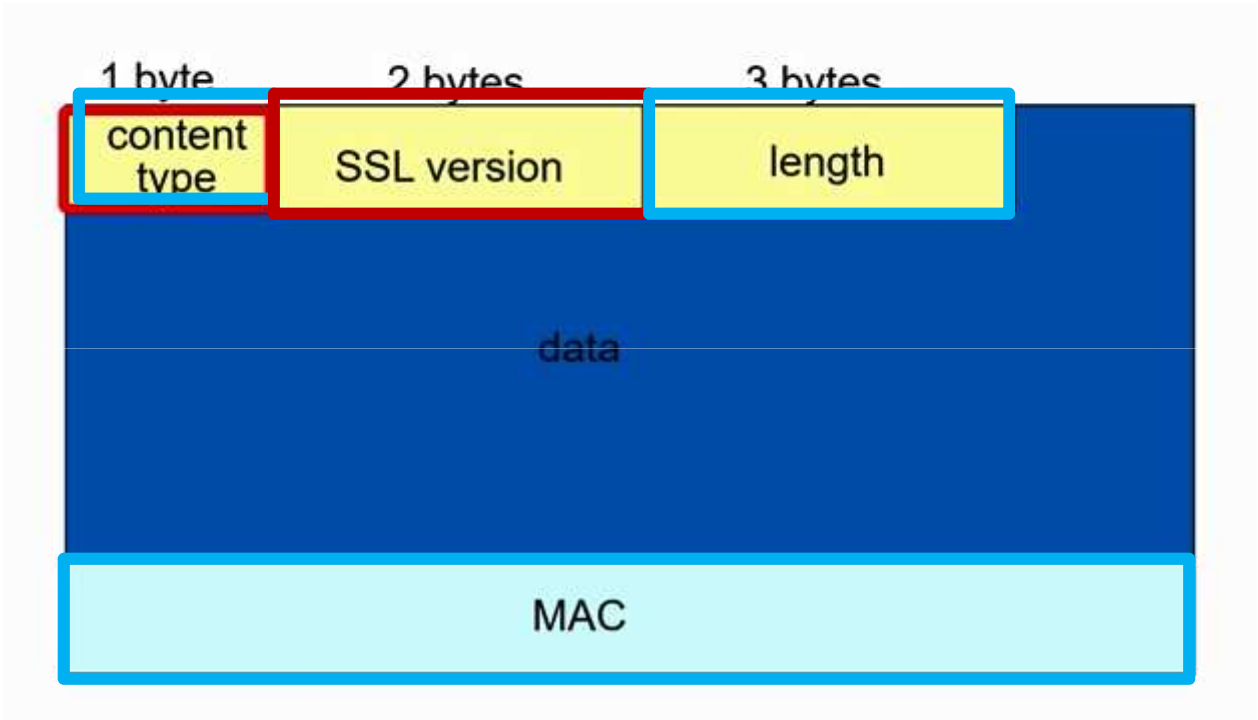
- 一般由用户端提供一系列的加密算法，其破解难度有高有低
- 中间人攻击有可能把难破解的算法从算法库中删除
  - 协议中最后两步确保上述情况不会发生
- 为什么有两个随机nonces?
  - 假设Trudy偷听到了 Alice和Bob之间的所有信息
  - 第二天，Trudy跟Bob建立一个TCP链接，并把前一天偷听来的数据包原封不动发给Bob—遍 ，这时Bob (Amazon)有可能会认为Alice对一种商品买了两遍
  - 解决方案：Bob对不同的链接附上不同的nonce,这样加密密钥在不同的两个链接(connection) 中是不一样的
  - 因此，第二天发给Bob的包会通不过信息正确性（integrity)检查

SSL协议：数据记录格式

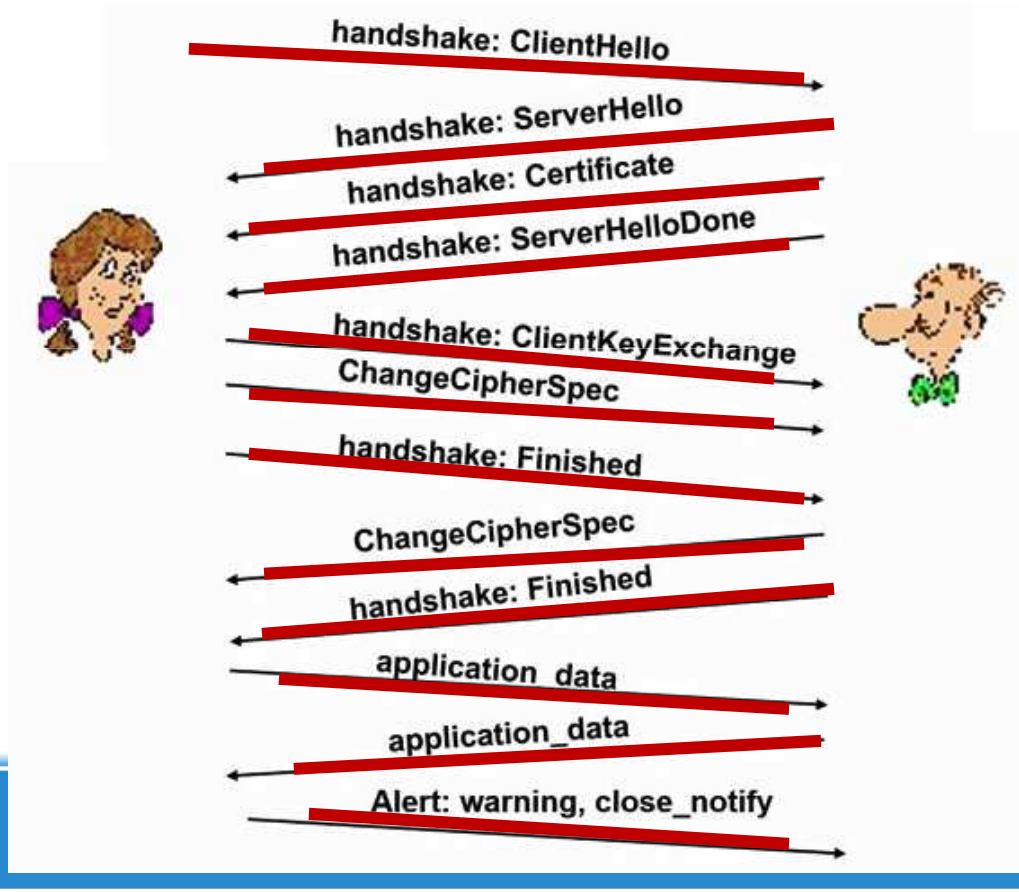


- **fragment:每一个 SSL fragment 含有2^{14} 字节（~16 Kbytes)**

SSL协议：数据记录格式



SSL实际连接过程


handshake: ClientHello
handshake: ServerHello
handshake: Certificate
handshake: ServerHelloDone
handshake: ClientKeyExchange
ChangeCipherSpec
handshake: Finished
ChangeCipherSpec
handshake: Finished
application_data
application_data
Alert: warning, close_notify

密钥衍生

- 用户端nonce,服务器nonce,和pre-master-secret —起被输入一个虚拟随机函数
  - 函数随机生成master secret
- master secret和新的nonces被一起输入到另外一个随机数生成函数，输出密钥块字段（key block）
- key block被分割处理，产生下列密钥：
  - 1.用户MAC密钥
  - 2.服务器MAC密钥
  - 3.用户数据加密密钥
  - 4.服务器数据加密密钥
  - 5.用户初始向量（client initialization vector)
  - 6.服务器初始向量（server initialization vector)

02.2
Part

SSL体系结构

6.2.1 SSL体系结构

- **两个重要的SSL概念是：**

SSL 连接

- 提供合适类型服务的传输
- 对于SSL，这种连接是点对点关系
- 连接是暂时的
- 每个连接都与一个会话相关联

SSL 会话

- 客户端和服务器之间的一种关联
- 由握手协议创建
- 定义一组可在多个连接之间共享的加密安全性参数
- 用于避免每个连接的新安全参数的昂贵协商

SSL的两个重要概念 - 1

- **SSL会话（session）**
  - 一个SSL会话指客户与服务器之间的联系
  - 会话由SSL握手协议创建，定义了一套安全加密参数，为多个连接所共享
  - 握手协议的职责是协调客户和服务器的状态，使得双方在不能精确地并行的情况下，也能工作一致
- **会话状态**
  - 会话标识
  - Peer 证书：peer的x509证书
  - 压缩方法
  - 密码规范：加密算法；消息摘要算法；有关密码的一些属性
  - 主密钥：客户和服务器共享的主密钥
  - 恢复：用一个标志位表示该会话是否可以初始化一个新的连接。

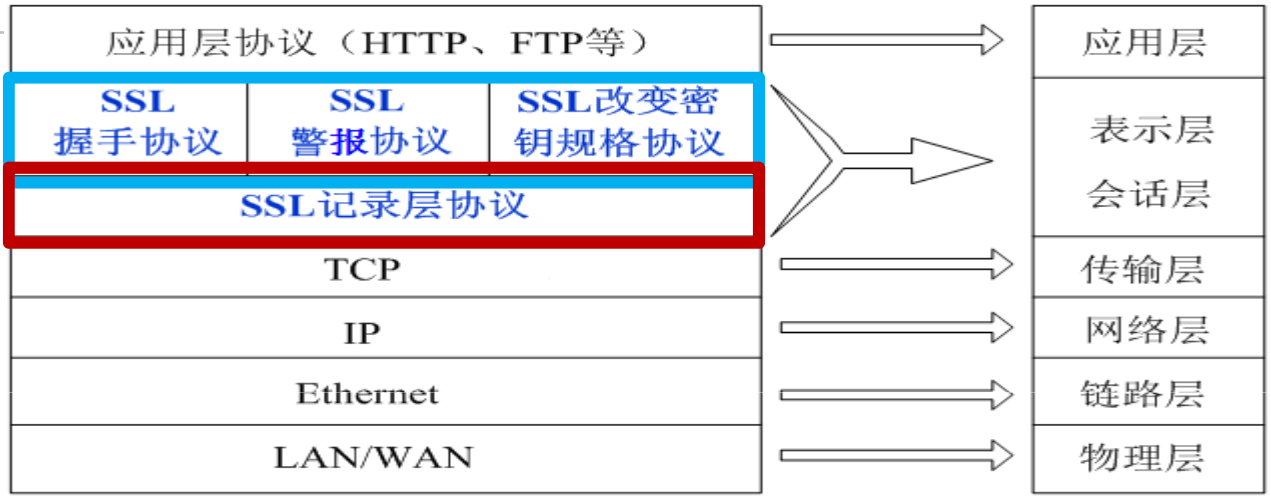
SSL的两个重要概念 - 2

- SSL连接（connection)
  - 用于实现特定类型的服务数据的安全传输
  - SSL的连接是点对点的关系
  - 连接是暂时的，每一个连接和一个会话关联
  - 一个SSL会话可包括多种安全连接
- 连接状态
  - 服务器和客户端随机序列
  - 服务器端写摘要协议(写MAC)
  - 客户端写摘要协议(写MAC)
  - 服务器端写密钥
  - 客户端写密钥
  - 初始化向量
  - 序列号

SSL安全连接的特性

- **连接具有私有性**
  - 在初始化连接后，协商密钥，基于对称密码体制对数据进行加密（如DES、RC4等）
- **对端实体鉴别**
  - 可采用非对称密码体制或公开密钥密码体制（如RSA、DSS）
- **连接是可靠的**
  - 消息传输使用加密MAC算法进行消息完整性检查

SSL安全协议的组成



●SSL不是单个的协议，而是两层协议

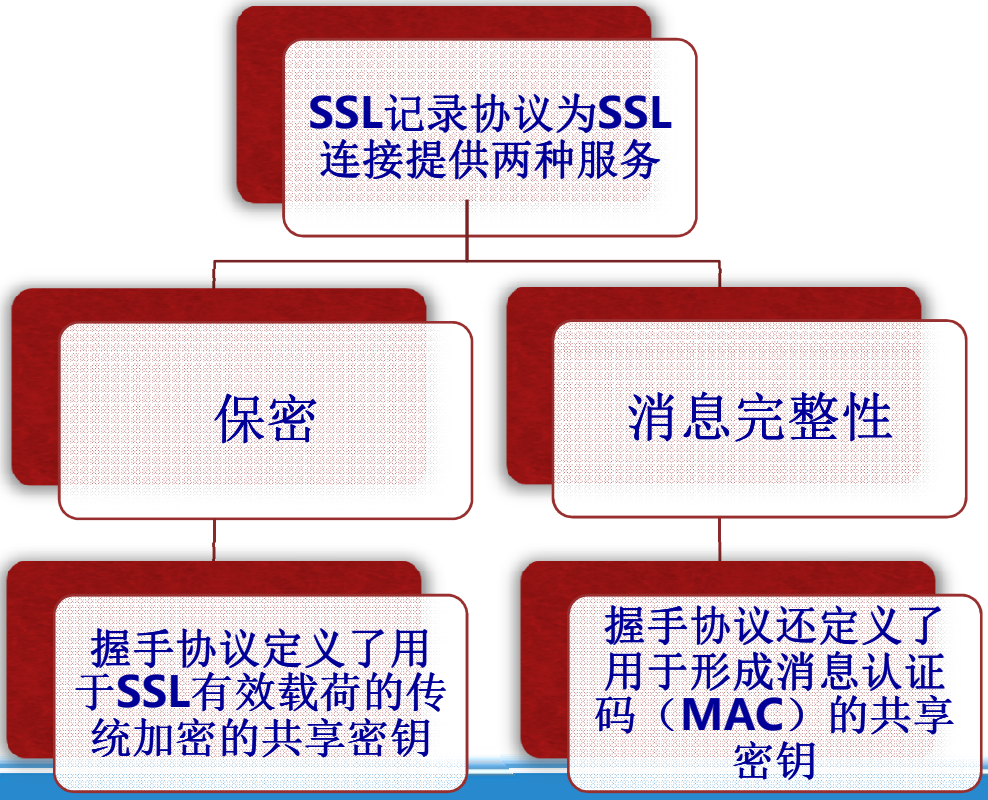
✓SSL记录协议基于可靠的传输层协议，用来封装高层协议

✓高层协议主要包括**SSL**握手协议、修改密码参数协议、警报协议、应用数据协议（如**HTTP**）等。

02.3
Part

SSL记录协议

6.2.2 SSL记录协议


SSL记录协议为SSL连接提供两种服务
保密
消息完整性
握手协议定义了用于SSL有效载荷的传统加密的共享密钥
握手协议还定义了用于形成消息认证码（MAC）的共享密钥

6.2.2 SSL记录协议（续）

- **记录层封装各种高层协议**
  - **以任意大小的非空块从高层接收尚未解释的数据**
  - **分块**
    - **将高层数据分割成SSL明文记录**
  - **压缩和解压**
    - **所有记录采用会话状态中定义的压缩算法进行压缩**
  - **记录的保护**
    - **所有记录通过会话状态中定义的加密算法和MAC算法进行保护**

SSL记录协议的操作过程

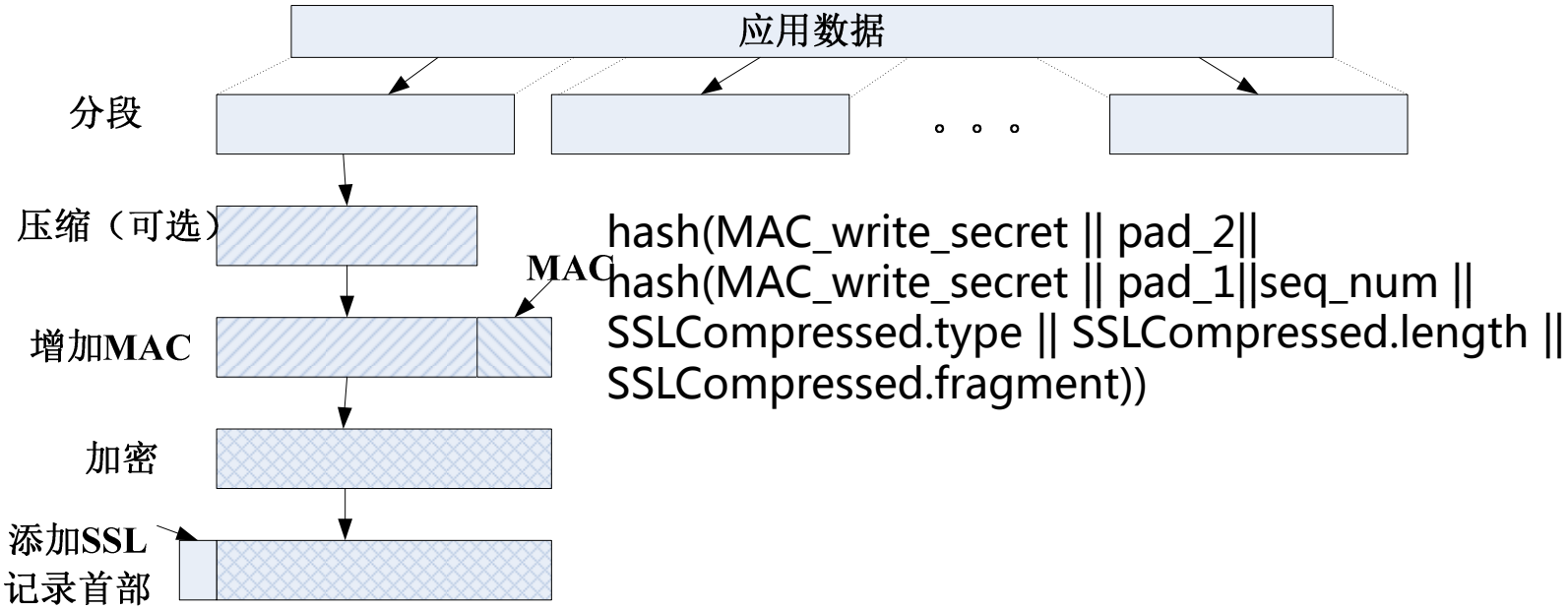


图6.3 SSL记录协议的运行流程

SSL记录首部的组成

- 内容类型（8比特）：用来说明封装的数据段的更高层的协议。已定义的内容类型有：修改密码规程协议、告警协议、握手协议和应用数据。
- 主要版本（8比特）：指示使用SSL的主要版本号；对于SSLv3.0，该值为3。
- 次要版本（8比特）：指示使用的次要版本号；对于SSLv3.0，该值为0。
- 压缩长度（8比特）：明文数据段以字节为单位的长度（如果使用压缩就是压缩数据段）。最大值为$2^{14}+2048$

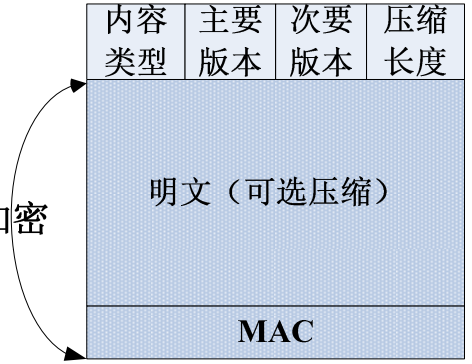


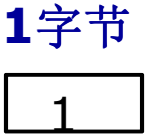
图6.4 SSL记录协议的格式

02.4
Part

SSL修改密码规格协议

6.2.3 SSL修改密码规格协议

- 位于SSL记录协议之上
- 仅定义了一个由单个字节“1”构成的消息报文
- 该消息将改变连接所使用的加密规约
- 用途：切换状态
  - 把密码参数设置为当前状态。在握手协议后，当安全参数协商一致后，发送此消息，以通知接收方下面的记录将受到刚达成的密码规范和密钥的保护。



密码变更规格协议

02.5
Part

警报协议

6.2.4 警报协议（或告警协议）

- **一种通过SSL记录协议进行传输的特定类型的消息**
- **主要作用**
  - **规定了告警的级别和告警的类型，在SSL协议执行过程中通过警报协议来显示信息交换过程中所发生的错误**

1字节	1字节
级别	类型

警报级别（或告警级别）

级别	告警名称	含义
1	警告	表示一个一般警告信息.
2	致命错误	表示出现了致命的错误,立即终止当前连接,同一会话的其它连接还可继续,不会再产生其它新的连接.

告警类型

类型	告警名称	含义
0	Close_notify	通知接受方，发送方在本连接中不再发送消息.
10	Unexpected_message	收到不适当的消息.
20	Bad_recode_mac	接收到的记录的MAC有错误（致命错误）.
30	Decompression_failure	解压缩失败（致命错误）.
40	Handshake_failure	发送方无法进行的安全参数设置（致命错误）.
41	No_certificate	认证中心没有证书.
42	Bad_certificate	证书已破坏.
43	Unsupported_certificate	不支持接收的证书类型.
44	Certificate_revoked	证书已经撤销.
45	Certificate_expired	证书过期.
46	Certificate_unknown	在产生证书时有不明问题.
47	Illegal_parameter	握手过程某个字段超出范围.

02.6
Part

SSL握手协议

6.2.5 SSL握手协议

- **握手协议**在SSL记录层之上，负责建立当前会话状态的参数
  - 客户端和服务器相互鉴别对方身份（利用证书）
  - 协商安全参数（加密和MAC算法，加密密钥等）
- **由一系列在客户和服务器间交换的消息组成**
- **每个消息由3个字段组成：消息类型、以字节为单位的消息长度以及这个消息有关的参数内容**

1字节	3字节	≥0字节
类型	长度	内容

握手协议

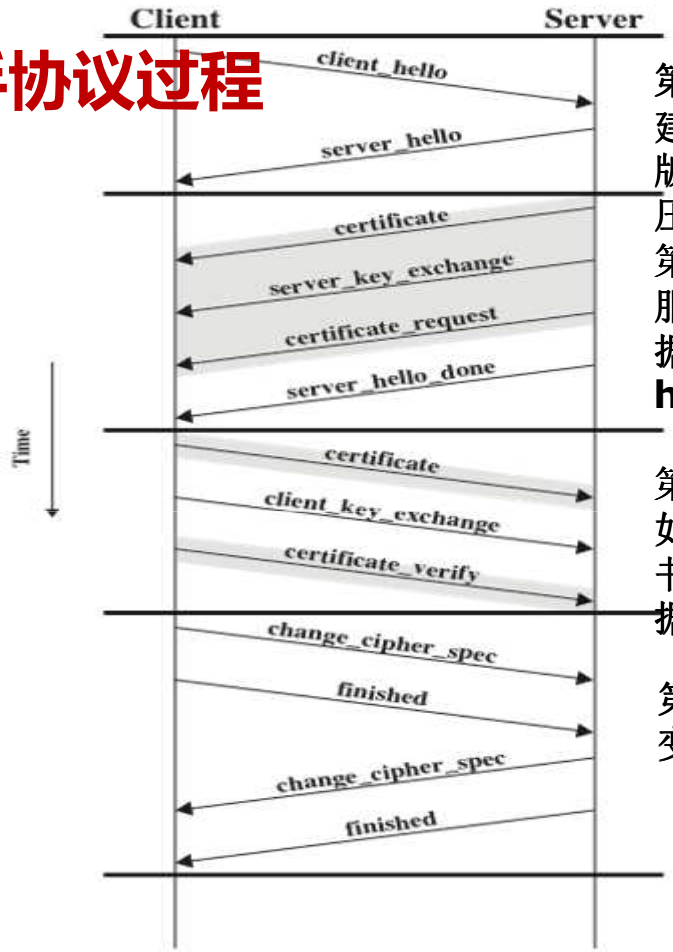
握手协议定义的消息类型 - 1

消息类型	说明	参数
hello_request	握手请求，服务器可在任何时候向客户端发送该消息。若客户端正在进行握手过程就可忽略该消息。否则客户端发送cleint_hello消息，启动握手过程。	无
client_hello	客户启动握手请求，当客户第一次连接服务器时向服务器发送的第一条消息。该消息中包括了客户端支持的各种算法。若服务器端不能支持，则本次会话可能失败。	版本、随机数、会话ID、密文族、压缩方法
server_hello	其结构与client_hello消息相同，该消息是服务器对客户端client_hello消息的回复。	版本、随机数、会话ID、密文族、压缩方法
server_certificate	服务器提供的证书。如果客户要求对服务器进行认证，则服务器在发送server_hello消息后，向客户端发送该消息。证书的类型一般是X.509v3。	X．509v3证书链
server_key_exchange	服务器密钥交换。当服务器不使用证书，或其证书中仅提供签名而不提供密钥时，需要使用本消息来交换密钥。	参数、签名

握手协议定义的消息类型 - 2

消息类型	说明	参数
certificate_ request	用于服务器向客户端要求一个客户证书。	类型、授权
server_hello_ done	该消息表明服务器端的握手请求报文已经发送完毕，正在等待客户端的响应。客户端在收到该消息时，将检查服务器提供的证书及其他参数是否是有效、可以接受的。	无
client_ certificate	客户端对服务器certificate_request消息的响应，只有在服务器端要求客户证书的时候使用。一般该消息是客户端收到server_hello_done消息后所发送的第一条消息。若客户端没有合适的证书，则向服务器端发送no_certificate的告警消息（无证书可能导致握手失败）	X．509 v3证书链
client_key_ exchange	客户密钥交换。当客户不使用证书，或其证书中仅提供签名而不提供密钥时，需要使用本消息来交换密钥。	参数、签名
certificate_ verify	该消息用于向服务器提供对客户证书的验证。	签名
finished	该消息在“加密规约修改”（Change Cipher Spec）消息之后发送，以证实握手过程已经成功完成。本消息发送后，发送方开始使用协商的新参数来执行操作。该消息需要在两个方向上传送。	散列值

握手协议过程



第一阶段：
建立安全连接请求，包括协议版本、会话**ID**、密码构件、压缩方法和初始随机数

建立安全能力

第二阶段：
服务器发送证书、密钥交换数据和证书请求，最后发送**hello**消息阶段的结束信号

服务器认证和密钥交换

第三阶段：
如果有证书请求，客户端发送证书。之后客户端发送密钥交换数据，也可以发送证书验证信息

客户端认证和密钥交换

第四阶段：
变更密码构件和结束握手协议

完成

注：加阴影的传输是可选的，或者是与情况相关的消息，它们并非总会被发送

握手协议过程 - 1

- **第一阶段 安全能力的建立**
  - (1) 客户 → 服务器 ：client_hello
    - 版本
    - 随机数：防止重放攻击
    - 会话标志
    - 密码套件
      - 密钥交换方法：RSA、Diffie-Hellman、Fortezza
      - CipherSepc:密码算法、MAC算法、密码类型、散列长度、IV等
    - 压缩方法
  - (2) 服务器 → 客户 ：server_hello
    - 同client_hello

握手协议过程 - 2

- **第二阶段 服务器认证和密钥交换**
  - **(3) 服务器 → 客户 ：server_certificate**
    - **发送X.509证书**
  - **(4) 服务器 → 客户 ：server_key_exchange**
    - **瞬时Diffie-Hellman等需要**
  - **(5) 服务器 → 客户 ：certificate_request**
    - **可能需要**
  - **(6) 服务器 → 客户 ：server_hello_done**
    - **无参数**

握手协议过程 - 3

- **第三阶段 客户认证和密钥交换**
  - (7) 客户 → 服务器 ：client_certificate
  - (8) 客户 → 服务器 ：client_key_exchange
  - (9) 客户 → 服务器 ：certificate_verify
- **第四阶段 结束阶段**
  - (10) 客户 → 服务器 ：change_cipher_spec
  - (11) 客户 → 服务器 ：finished
  - (12) 服务器 → 客户 ：change_cipher_spec
  - (13) 服务器 → 客户 ：finished

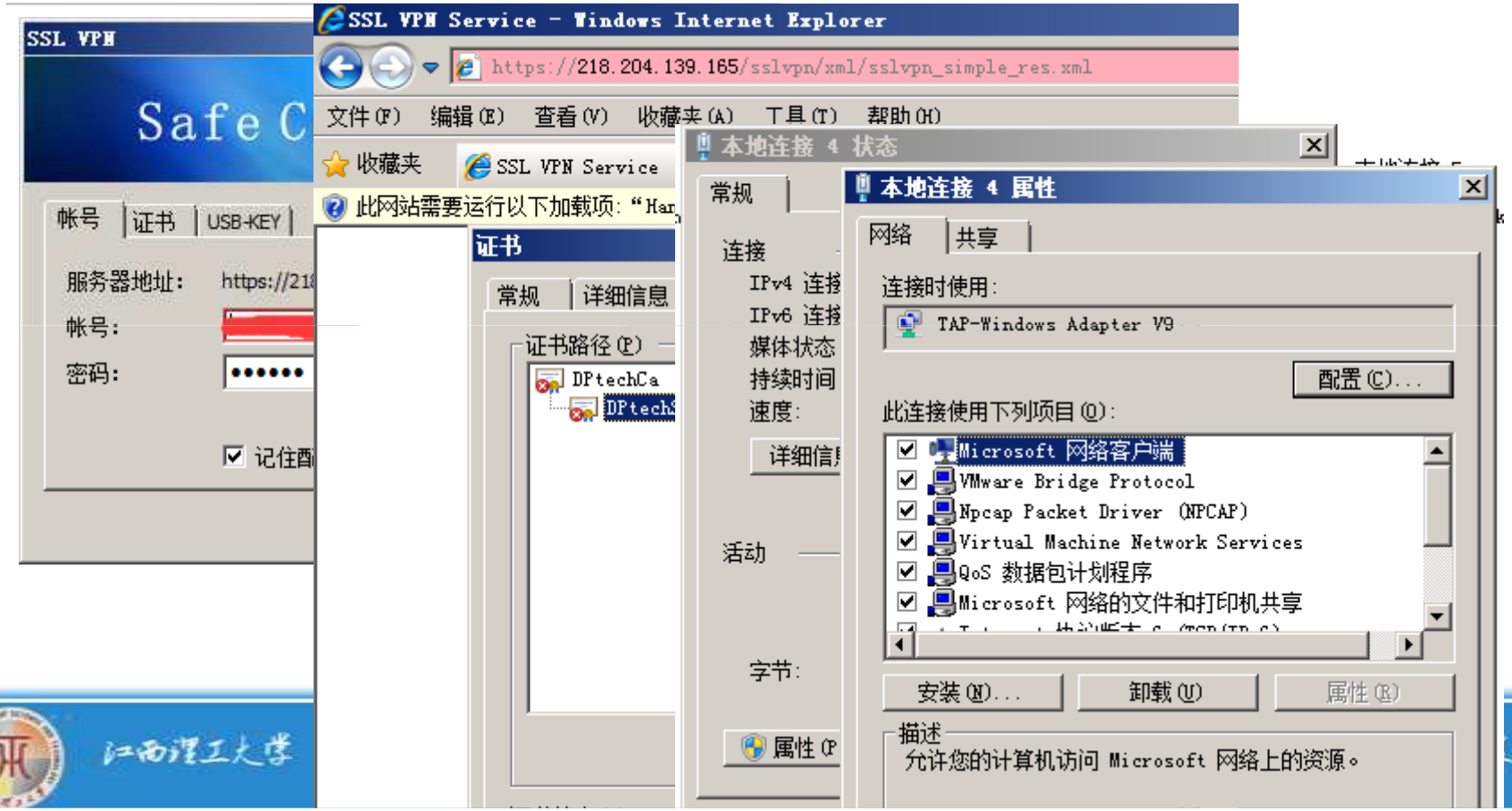
SSL工作过程

- **发送方的工作过程**
  - 从上层接收要发送的数据
  - 对信息进行分段，生成若干记录
  - 使用指定的压缩算法进行数据压缩数据（可选）；
  - 使用指定的MAC算法生成MAC；
  - 使用指定的加密算法进行数据加密；
  - 发送数据

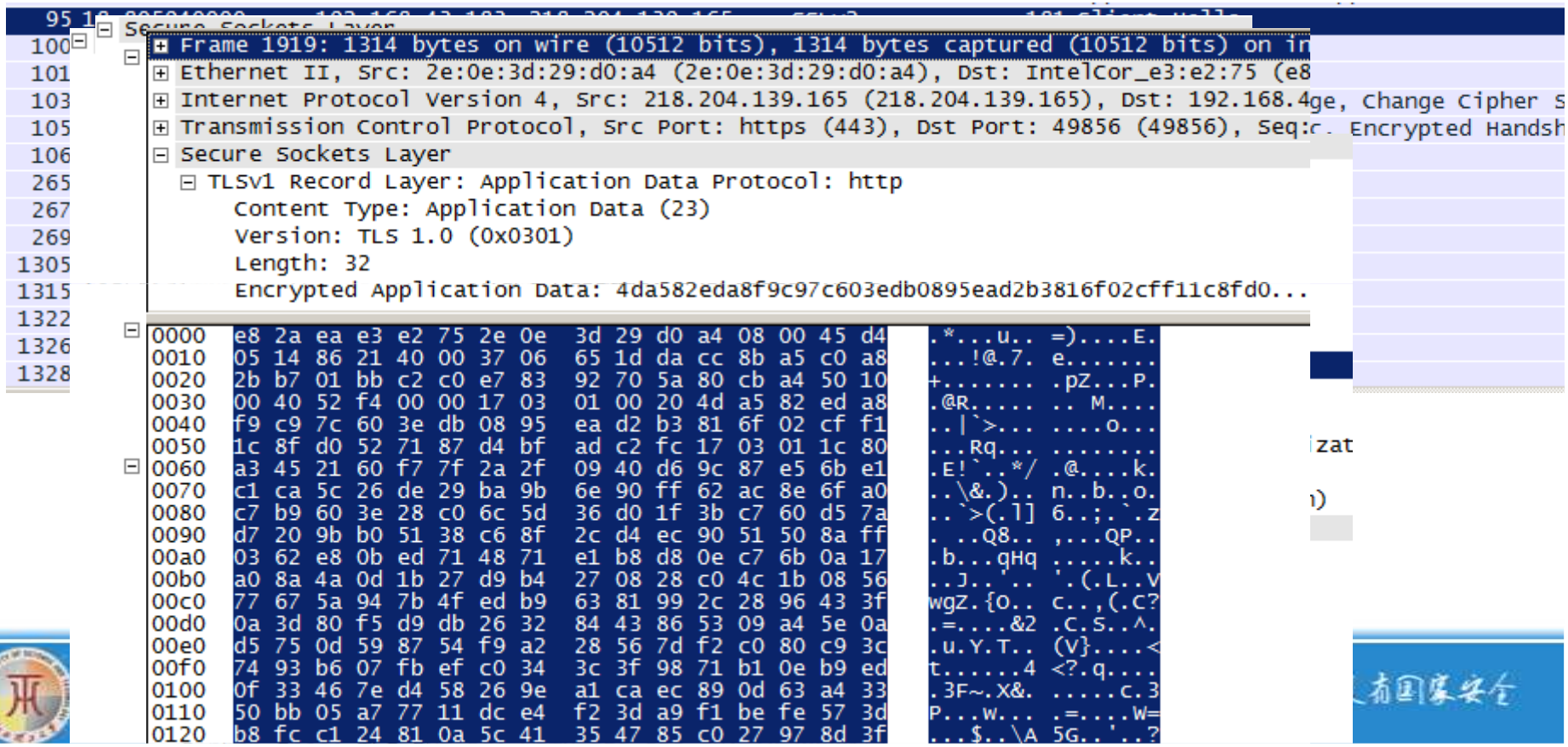
SSL工作过程

- **接收方的工作过程**
  - 接收数据；
  - 使用指定的解密算法解密数据；
  - 使用指定的MAC算法校验MAC；
  - 使用压缩算法对数据解压缩（在需要时进行）；
  - 将记录进行数据重组；
  - 将数据发送给高层。

我校SSL VPN


SSL VPN
Safe C
帐号
证书
USB-KEY
服务器地址:
帐号:
密码:
记住
SSL VPN Service - Windows Internet Explorer
https://218.204.139.165/sslvpn/xml/sslvpn_simple_res.xml
文件(F) 编辑(E) 查看(V) 收藏夹(A) 工具(T) 帮助(H)
收藏夹
SSL VPN Service
此网站需要运行以下加载项: "Har
证书
常规
详细信息
证书路径(P)
DPtechCa
本地连接 4 状态
常规
连接
IPv4 连接
IPv6 连接
媒体状态
持续时间
速度:
详细信息
活动
字节:
属性(P
本地连接 4 属性
网络
共享
连接时使用:
TAP-Windows Adapter V9
配置(C)...
此连接使用下列项目(O):
Microsoft 网络客户端
VMware Bridge Protocol
Npcap Packet Driver (NPCAP)
Virtual Machine Network Services
QoS 数据包计划程序
Microsoft 网络的文件和打印机共享
安装(N)...
卸载(U)
属性(R)
描述
允许您的计算机访问 Microsoft 网络上的资源。

我校SSL VPN


Frame 1919: 1314 bytes on wire (10512 bits), 1314 bytes captured (10512 bits) on in
Ethernet II, Src: 2e:0e:3d:29:d0:a4 (2e:0e:3d:29:d0:a4), Dst: IntelCor_e3:e2:75 (e8
Internet Protocol Version 4, Src: 218.204.139.165 (218.204.139.165), Dst: 192.168.4
Transmission Control Protocol, Src Port: https (443), Dst Port: 49856 (49856), Seq:
Secure Sockets Layer
TLSv1 Record Layer: Application Data Protocol: http
Content Type: Application Data (23)
Version: TLS 1.0 (0x0301)
Length: 32
Encrypted Application Data: 4da582eda8f9c97c603edb0895ead2b3816f02cff11c8fd0...
0000 e8 2a ea e3 e2 75 2e 0e 3d 29 d0 a4 08 00 45 d4
0010 05 14 86 21 40 00 37 06 65 1d da cc 8b a5 c0 a8
0020 2b b7 01 bb c2 c0 e7 83 92 70 5a 80 cb a4 50 10
0030 00 40 52 f4 00 00 17 03 01 00 20 4d a5 82 ed a8
0040 f9 c9 7c 60 3e db 08 95 ea d2 b3 81 6f 02 cf f1
0050 1c 8f d0 52 71 87 d4 bf ad c2 fc 17 03 01 1c 80
0060 a3 45 21 60 f7 7f 2a 2f 09 40 d6 9c 87 e5 6b e1
0070 c1 ca 5c 26 de 29 ba 9b 6e 90 ff 62 ac 8e 6f a0
0080 c7 b9 60 3e 28 c0 6c 5d 36 d0 1f 3b c7 60 d5 7a
0090 d7 20 9b b0 51 38 c6 8f 2c d4 ec 90 51 50 8a ff
00a0 03 62 e8 0b ed 71 48 71 e1 b8 d8 0e c7 6b 0a 17
00b0 a0 8a 4a 0d 1b 27 d9 b4 27 08 28 c0 4c 1b 08 56
00c0 77 67 5a 94 7b 4f ed b9 63 81 99 2c 28 96 43 3f
00d0 0a 3d 80 f5 d9 db 26 32 84 43 86 53 09 a4 5e 0a
00e0 d5 75 0d 59 87 54 f9 a2 28 56 7d f2 c0 80 c9 3c
00f0 74 93 b6 07 fb ef c0 34 3c 3f 98 71 b1 0e b9 ed
0100 0f 33 46 7e d4 58 26 9e a1 ca ec 89 0d 63 a4 33
0110 50 bb 05 a7 77 11 dc e4 f2 3d a9 f1 be fe 57 3d
0120 b8 fc c1 24 81 0a 5c 41 35 47 85 c0 27 97 8d 3f

02.7
Part

密码计算

6.2.6 密码计算

- 通过密钥交换创建共享主密钥
- 从主密钥生成加密参数

通过密钥交换创建共享主密钥

- 共享主密钥是通过安全密钥交换为该会话生成的一次性48字节值
  - 交换预备主密钥（pre_master_secret）
  - 双方计算主密钥（master_secret）
- 交换预备主密钥
  - RSA：客户端产生一个48字节的预备密钥，RSA公钥加密，发给服务器
  - Diffi-Hellman:各自产生公钥，交换后，计算共享预备主密钥

计算主密钥

- **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))**

从主密钥生成加密参数

- 密码规格要求客户端写入MAC值的密钥，服务器写入MAC值密钥，客户端写入密钥，服务器写入密钥，客户端写初始向量IV和服务器写IV，这些密钥是从主密钥按顺序生成的。
- 其方法是主密钥利用散列函数来产生安全字节序列，安全字节序列足够长以便生成所有需要的参数。

生成密码参数

- **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)) ||...**

03
Part

传输层安全（TLS）

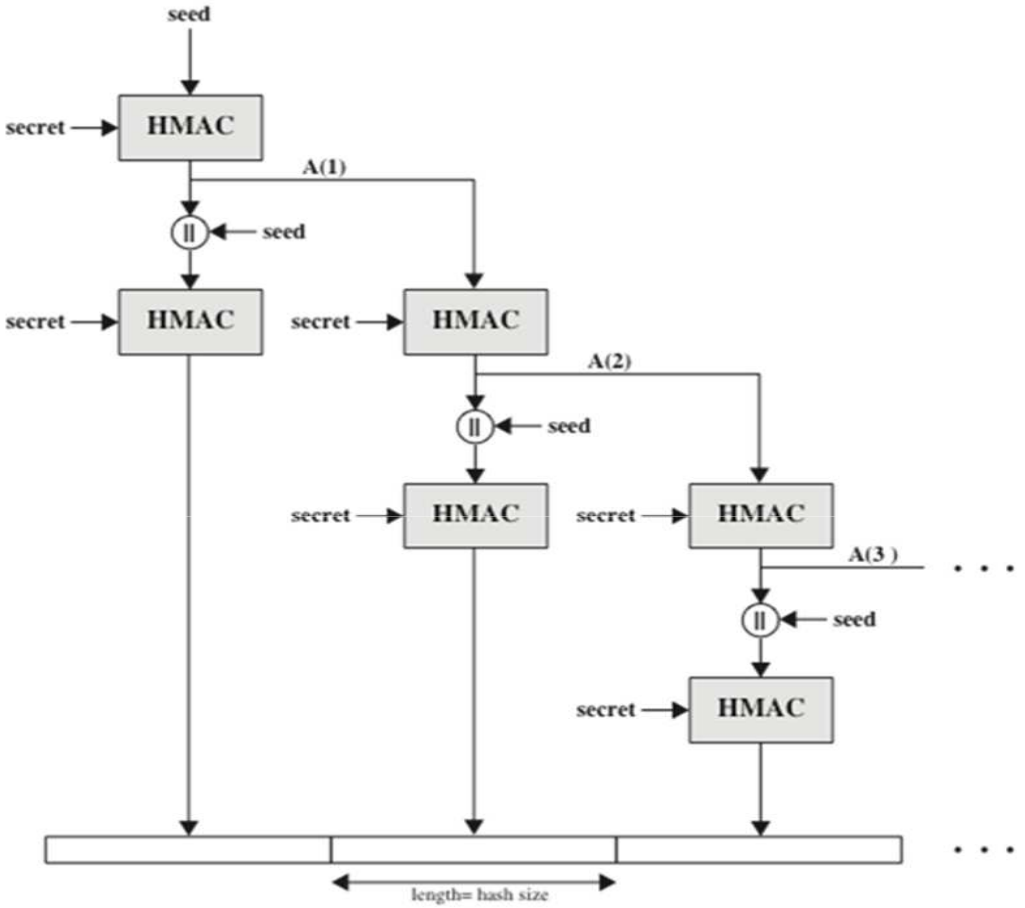
6.3 传输层安全（TLS）

- 传输层安全(TLS)是IETF标准的初衷，其目标是编写SSL的互联网标准版本。
- TLS v1协议本身基于SSL v3，很多与算法相关的数据结构和规则十分相似。

6.3 传输层安全（TLS）（续）

- RFC5246 非常接近于SSLv3
- 记录格式与SSL完全相同
- TLS不同之处
  - ✓ 消息认证码的计算使用了RFC2104定义的HMAC算法；而SSL也使用相同算法，只是填充部分采用了与密钥串接的方式而不是异或方式，安全强度基本相同
  - ✓ 定义了更多的报警码
  - ✓ 密钥构建细小差别 TLS不支持Fortezza
  - ✓ 密钥计算

伪随机函数


seed
secret
HMAC
A(1)
seed
secret
HMAC
secret
HMAC
A(2)
seed
secret
HMAC
secret
HMAC
A(3)
seed
secret
HMAC
length= hash size

04
Part

HTTPs协议

6.4 HTTPs协议

- 6.4.1 连接的初始化
- 6.4.2 连接关闭

6.4 HTTPs协议

- **HTTPs是指用HTTP 和SSL结合来实现网络浏览器和服务器之间的安全通信 ，RFC2818。**

HTTPS和HTTP的区别

- ✓ **https协议需要到CA申请证书，一般免费证书很少，需要交费。**
- ✓ **http信息是明文传输，https 则是具有安全性的SSL加密传输协议。**
- ✓ **http和https使用的是完全不同的连接方式，用的端口也不一样，前者是80，后者是443。**
- ✓ **http的连接很简单，是无状态的；HTTPS协议是由SSL+HTTP协议构建的可进行加密传输、身份认证的网络协议，比http协议安全。**

HTTPs解决的问题

- **信任主机的问题**
  - 采用https的服务器必须从CA 申请一个用于证明服务器用途类型的证书。目前所有的银行系统网站，关键部分应用都是https 的。客户通过信任该证书，从而信任了该主机。
- **通讯过程中的数据的泄密和被篡改**
  - 一般意义上的https，就是服务器有一个证书。保证服务器是他声称的服务器。
  - 服务端和客户端之间的所有通讯，都是加密的。

    客户端产生一对称的密钥，通过服务器的证书来交换密钥
  - 所有的信息往来都加密

HTTPs解决的问题

少许对客户端有要求的情况下，会要求客户端也必须有一个证书。

- **除了用户名/密码，还有一个CA 认证过的身份。**
- **个人银行的专业版是这种做法，具体证书可能是拿U盘（即U盾）作为一个备份的载体。**

6.4.1 连接的初始化

对于HTTPS，充当HTTP客户端的代理也充当TLS客户端

- 客户端在适当的端口上启动与服务器的连接，然后发送TLS ClientHello以开始TLS握手
- 当TLS握手完成时，然后客户端可以发起第一个HTTP请求
- 所有HTTP数据都要以TLS应用数据的形式发送

HTTPS连接中有三层不同的意思：

- 在HTTP层面，HTTP客户端通过向下一层发送连接请求来请求与HTTP服务器的连接。 通常，下一层是TCP，但它也可以是TLS / SSL。
- 在TLS层，在TLS客户端和TLS服务器之间建立会话。 此会话可以随时支持一个或多个连接。
- 建立连接的TLS请求首先在客户端的TCP实体和服务器端的TCP实体之间建立TCP连接。

6.4.2 连接关闭

- HTTP客户端或服务器可以通过在HTTP记录中包含Connection：close行来指示关闭连接。
- 关闭HTTPS连接要求TLS关闭与远程端的对等TLS实体的连接，这将涉及关闭底层TCP连接。
- TLS实例必须在关闭连接之前启动关闭警报的交换。
  - 在发送关闭警报后，TLS实例可以关闭连接而无需等待对等方发送其关闭警报，从而生成“不完整关闭”。
- 未公布的TCP关闭可能是某种攻击的证据，因此HTTPs客户端应该在发生这种情况时发出某种安全警告。

HTTPs协议运行实例 - 1

1. 客户机使用IE浏览器向服务器发送客户端的SSL版本号、密码设置、随机数和需要服务器使用SSL协议与客户机进行通信的其它信息；IE浏览器使用https协议向服务器申请建立SSL会话；
2. 服务器向客户端发送服务器端的SSL版本号、密码设置、随机数和客户端使用SSL协议与服务器通信需要的其它信息。同时服务器端发送它自己的数字证书供客户认证，如果认为客户端需要身份认证则要求客户发送证书，该操作是可选的；

HTTPs协议运行实例 - 2

3. 客户端利用服务器发送信息认证服务器的真实身份并取得公开密钥等，如果服务器不被认证，用户被告警发生了问题，通知不能建立带有加密和认证的连接。若服务器能被成功地认证，客户机将继续下一步。

4. 客户机利用数字信封技术为将要进行的会话创建会话预密钥pre_master_secret，并用服务器的公钥加密它，然后向服务器发送加密的会话预密钥；

HTTPs协议运行实例 - 3

5. 当客户能被成功认证后，服务器会使用它的私人密钥解密从客户端得到的会话预密钥pre_master_secret， 并生成真正的会话密钥master_secret。同时客户端也从相同的pre_master_secret开始得到相同的master_secret 。会话密钥master_secret是对称密钥用于加密和解密在SSL会话期间交换的信息，并验证信息的完整性 ；
6. 会话密钥生成后，客户端向服务器发送消息，通知服务器以后从客户端来的消息将用会话密钥加密，这表明握手的客户端部分已经完成；

HTTPs协议运行实例 - 4

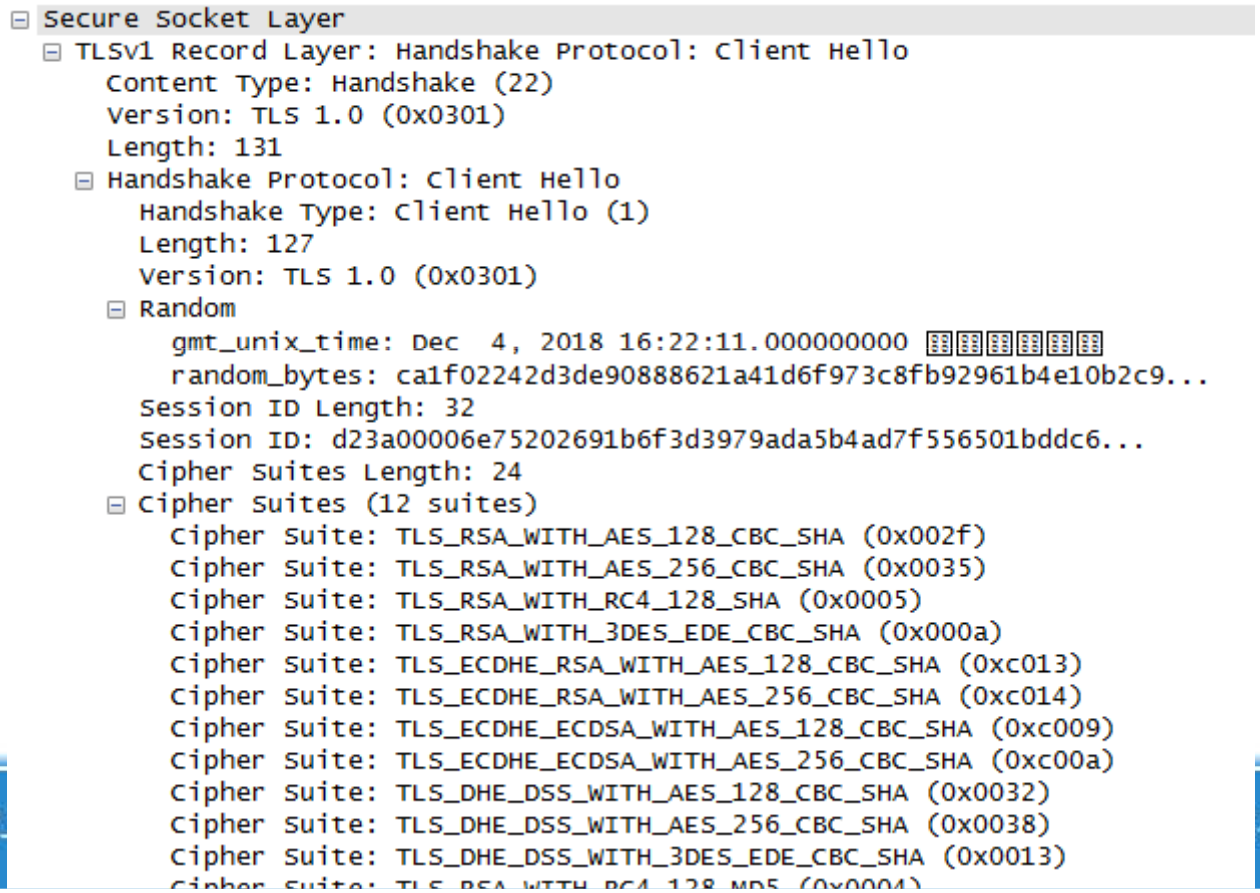
7. 服务器向客户端发送相同的消息，通知从服务器来的消息将用会话密钥加密，这表明握手的服务器部分已经完成；
8. SSL会话开始，使用安全通道发送消息，客户端和服务器使用会话密钥加密和解密它们彼此发送的数据和验证数据完整性；
9. 当通信完成后，一般情况会话密钥会被丢弃。

运行实例

- https://192.168.159.1/a.htm
- **TCP三次链接**

```
68 5.618153   192.168.159.140   192.168.159.1     TCP    51981 > https [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=8 SACK_
71 5.618540   192.168.159.1     192.168.159.140   TCP    https > 51981 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0 MSS=1460
72 5.618591   192.168.159.140   192.168.159.1     TCP    51981 > https [ACK] Seq=1 Ack=1 Win=65536 Len=0
```

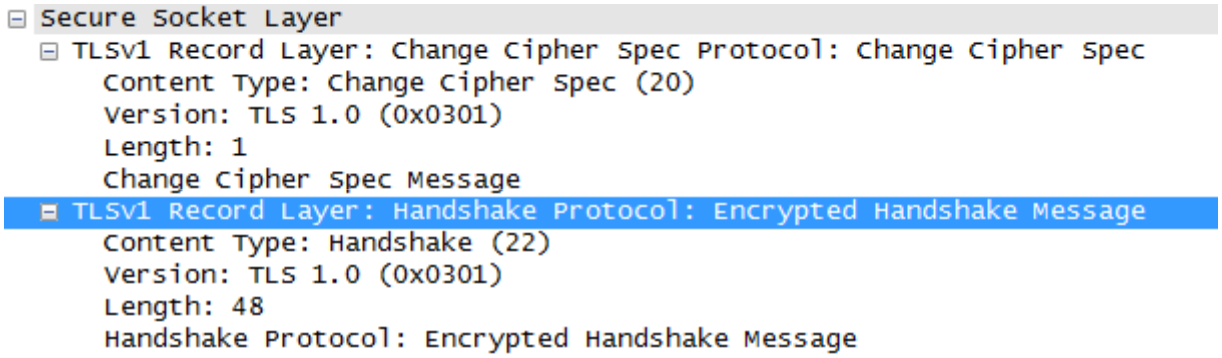
Client Hello



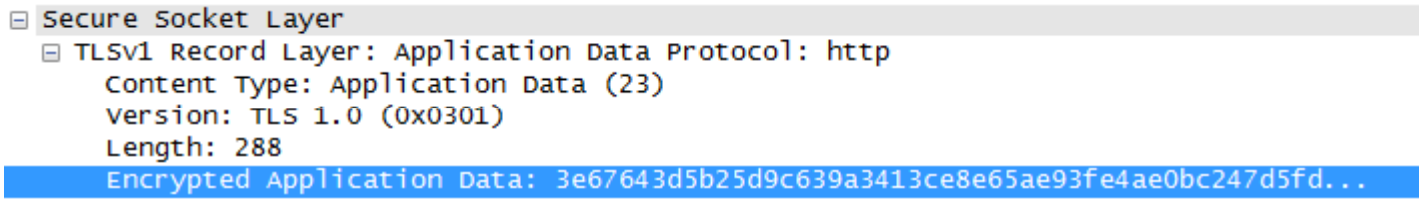
Server Hello

```
⊟ Secure Socket Layer
  ⊟ TLSv1 Record Layer: Handshake Protocol: Server Hello
      Content Type: Handshake (22)
      Version: TLS 1.0 (0x0301)
      Length: 81
    ⊟ Handshake Protocol: Server Hello
        Handshake Type: Server Hello (2)
        Length: 77
        Version: TLS 1.0 (0x0301)
      ⊟ Random
          gmt_unix_time: Dec  4, 2018 16:22:11.000000000 [illegible]
          random_bytes: 8eeaf1aab9698efd889b6f2a091e4cb1229e9cedc8d65448...
        Session ID Length: 32
        Session ID: d23a00006e75202691b6f3d3979ada5b4ad7f556501bddc6...
        Cipher Suite: TLS_RSA_WITH_AES_128_CBC_SHA (0x002f)
        Compression Method: null (0)
        Extensions Length: 5
      ⊞ Extension: renegotiation_info
  ⊟ TLSv1 Record Layer: Change Cipher Spec Protocol: Change Cipher Spec
      Content Type: Change Cipher Spec (20)
      Version: TLS 1.0 (0x0301)
      Length: 1
      Change Cipher Spec Message
  ⊟ TLSv1 Record Layer: Handshake Protocol: Encrypted Handshake Message
      Content Type: Handshake (22)
      Version: TLS 1.0 (0x0301)
      Length: 48
      Handshake Protocol: Encrypted Handshake Message
```

Change Cipher Spec



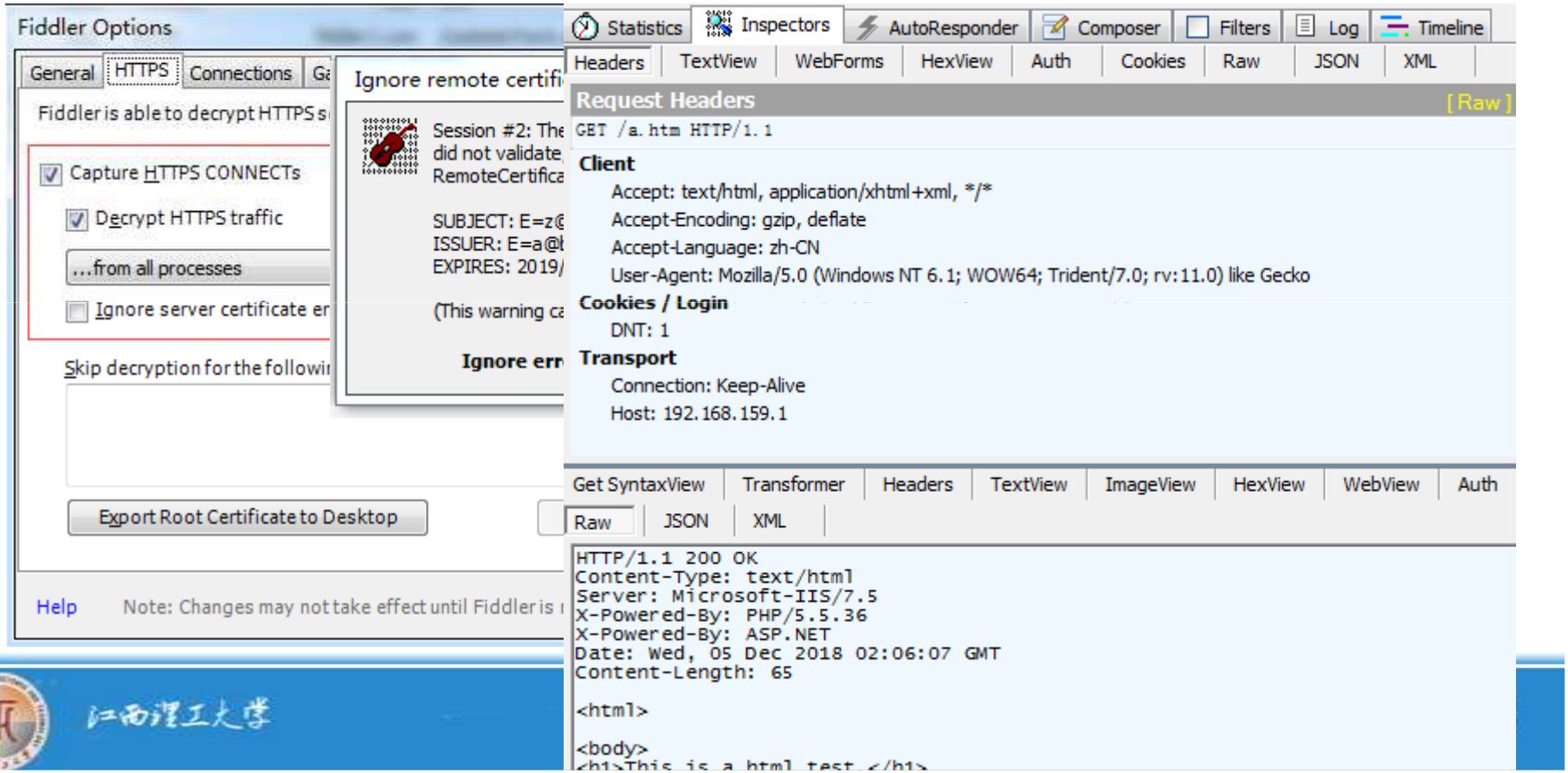
Data Transfer



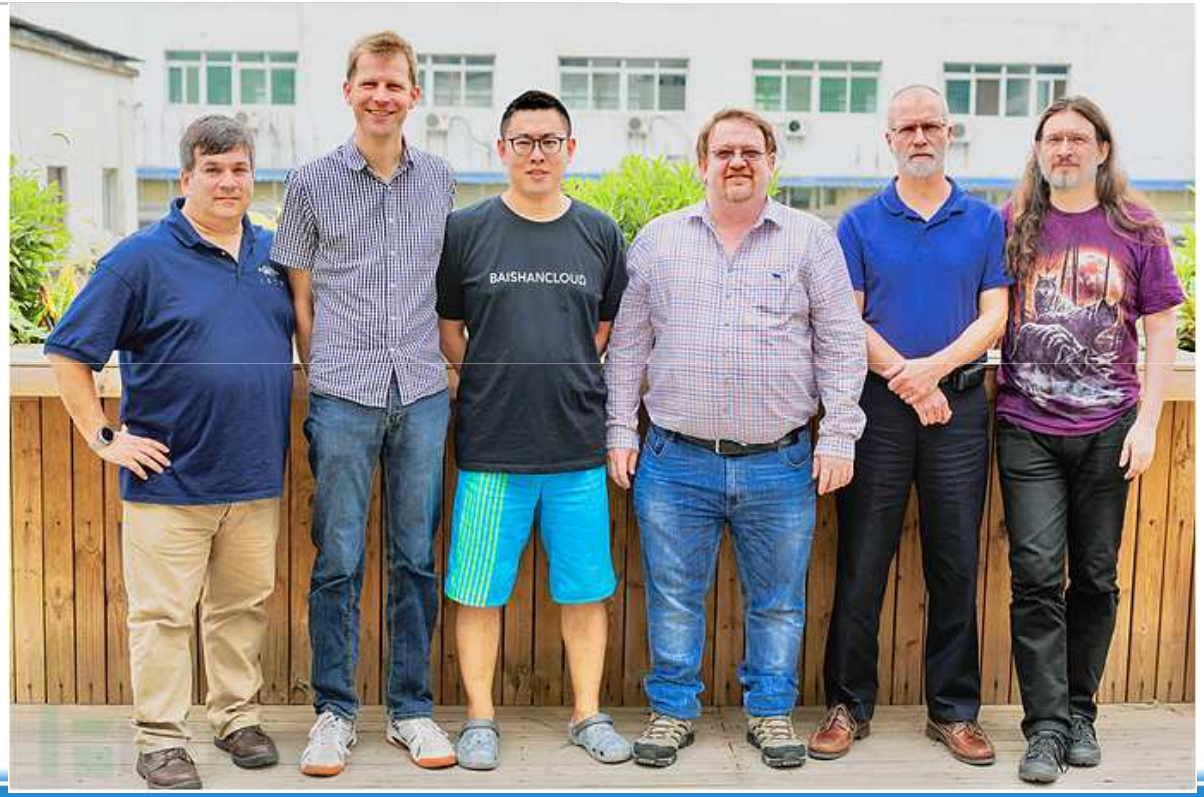
释放连接

```
83 5.686994   192.168.159.140   192.168.159.1     TCP    51981 > https [ACK] Seq=734 Ack=1772 Win=65536 Len=0
84 5.690822   192.168.159.140   192.168.159.1     TCP    51981 > https [FIN, ACK] Seq=734 Ack=1772 Win=65536 Len
85 5.692606   192.168.159.1     192.168.159.140   TCP    https > 51981 [FIN, ACK] Seq=1772 Ack=735 Win=65024 Len
86 5.692750   192.168.159.140   192.168.159.1     TCP    51981 > https [ACK] Seq=735 Ack=1773 Win=65536 Len=0
```

用fiddler做中间人



OpenSSL开发团队



05
Part

SSH协议

6.5 SSH协议

- 6.5.1 SSH传输协议
- 6.5.2 SSH身份认证协议
- 6.5.3 SSH连接协议
- 6.5.4 SSH协议的应用
- 6.5.5 SSH安全性分析

6.5 SSH协议

- **IETF RFCs（最初Tatu Ylönen开发）**
  - **RFC 4250, The Secure Shell (SSH) Protocol Assigned Numbers**
  - **RFC 4251, The Secure Shell (SSH) Protocol Architecture**
  - **RFC 4252, The Secure Shell (SSH) Authentication Protocol**
  - **RFC 4253, The Secure Shell (SSH) Transport Layer Protocol**
  - **RFC 4254, The Secure Shell (SSH) Connection Protocol**
  - **RFC 4255, Using DNS to Securely Publish Secure Shell (SSH) Key Fingerprints**
  - **RFC 4256, Generic Message Exchange Authentication for the Secure Shell Protocol (SSH)**
  - **RFC 4335, The Secure Shell (SSH) Session Channel Break Extension**
  - **RFC 4344, The Secure Shell (SSH) Transport Layer Encryption Modes**
  - **RFC 4345, Improved Arcfour Modes for the Secure Shell (SSH) Transport Layer Protocol**

目的

- 在非安全网络上提供安全的远程登录和其他安全网络服务
- SSH也是建立在应用层和传输层基础上的安全协议
- SSH主要解决的是密码在网络上明文传输的问题，通常用来替代Telnet、FTP等协议
  - 传统的Telnet、FTP和Rlogin等服务存在众多安全缺陷
    - 使用弱密码单一认证机制
    - 传输数据(包括账号和密码)为明文，容易被窃取、篡改和重放
    - 这些服务的安全验证机制容易引发各种欺骗，比如中间人攻击等

6.5 SSH协议（续）

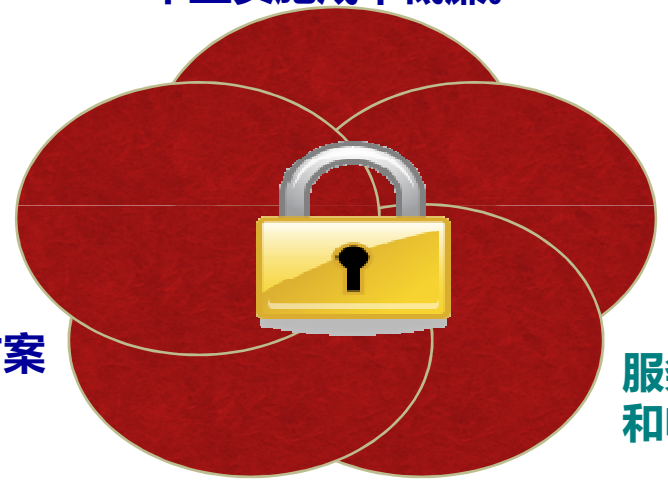
用于安全网络通信的协议，设计为相对简单且实施成本低廉。

SSH客户端和服务器应用程序可广泛用于大多数操作系统。

- 已成为远程登录和X隧道的首选方法。
- 正迅速成为嵌入式系统之外最普遍的加密技术应用之一。

SSH-2修复了原始方案中的许多安全漏洞。

- 被记录为IETF RFC 4250至4256中的建议标准。



最初的版本，SSH-1专注于提供安全的远程登录工具来取代TELNET和其他没有安全保障的远程登录方案。

SSH还提供更通用的客户端/服务器功能，可用于文件传输和电子邮件等网络功能。

SSH层次结构

- 每层提供不同类型的安全保护，并且可以与其它方式一起使用
  - SSH传输层协议（Transport Layer Protocol）
  - SSH用户认证协议（User Authentication Protocol）
  - SSH连接协议（Connection Protocol）

<table>
<tr><td colspan="3">SSH应用层协议</td></tr>
<tr><td colspan="3">SSH连接协议</td></tr>
<tr><td colspan="3">SSH用户认证协议</td></tr>
<tr><td colspan="3">SSH传输层协议</td></tr>
<tr><td>服务器认证</td><td>数据机密性</td><td>数据完整性</td></tr>
<tr><td colspan="3">TCP over IP</td></tr>
</table>

密钥机制

- 对于SSH以提供安全通讯为目标的协议，其中必不可少的就是一套完备的密钥机制
- SSH协议3个主要的密钥：主机密钥、服务器密钥和用户密钥

6.5.1 SSH传输协议

- SSH传输层协议提供加密主机认证、数据保密性和数据完整性保护
  - SSH 2.0几乎支持所有的公开密钥格式、编码和算法
- SSH传输层协议需要经过下列3个步骤
  - 密钥协商
    - 双方发送自己能支持的算法(压缩、加密、验证)，根据接收到的对方的算法进一步协商出一致的算法
  - 进行密钥交换(Diffie-Hellman)
  - 开始服务请求

SSH传输协议

- 主机密钥
  - 服务器认证发生在传输层，基于公共/私有密钥的服务器
  - 一个服务器会有多个主机密钥运用多重不同的非对称加密算法
  - 多个主机会公用一个主机密钥
  - 服务器的主机密钥在密钥交换时被用来确认主机的身份
- 分组交换
  - 图6.9说明了SSH传输层级协议中的事件序列。首先，客户端向服务器建立一个TCP连接。当连接建立后，客户端和服务器交换数据。
  - 分组的格式如图6.10所示。

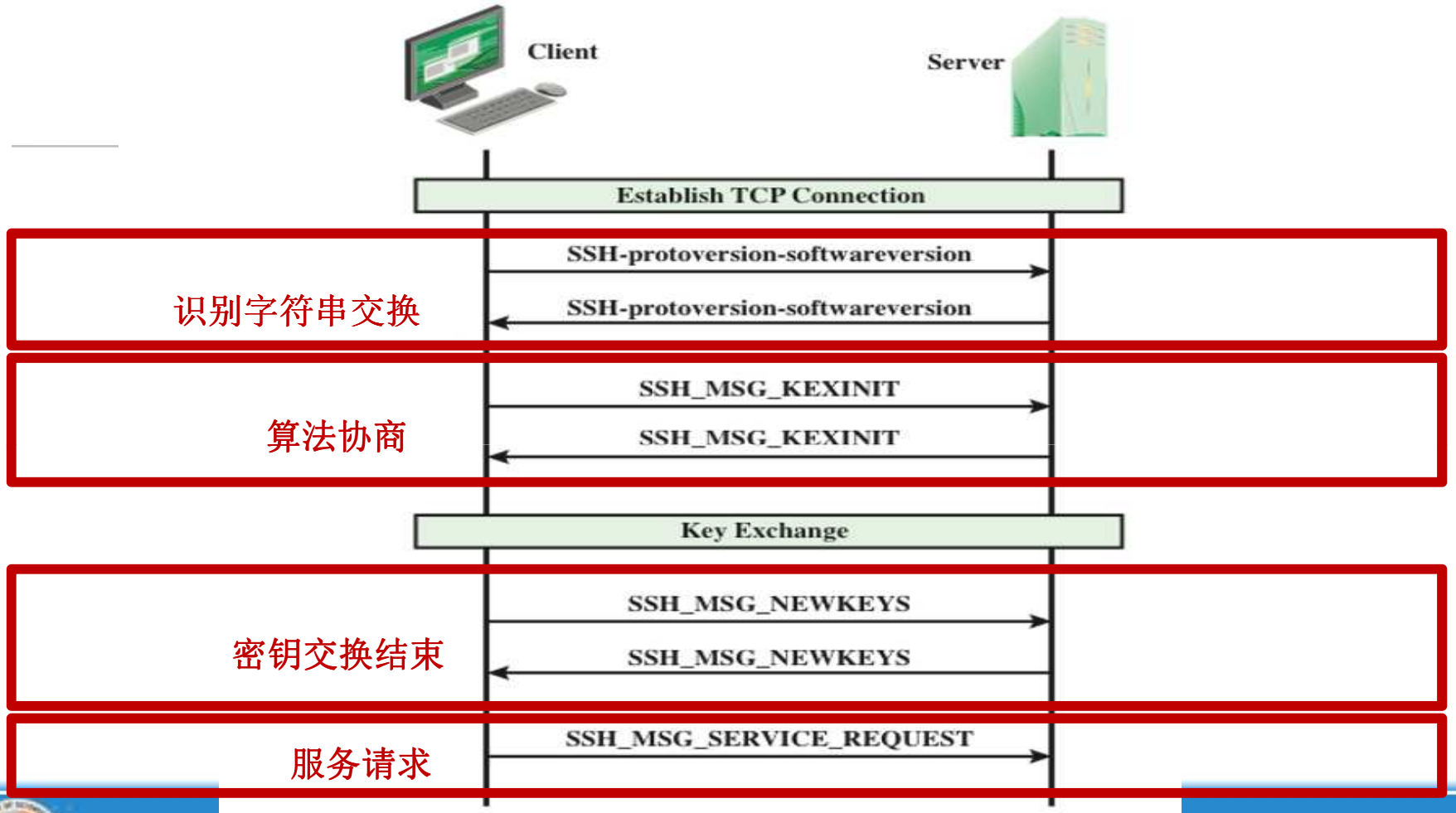


图6.9 SSH传输层协议分组交换

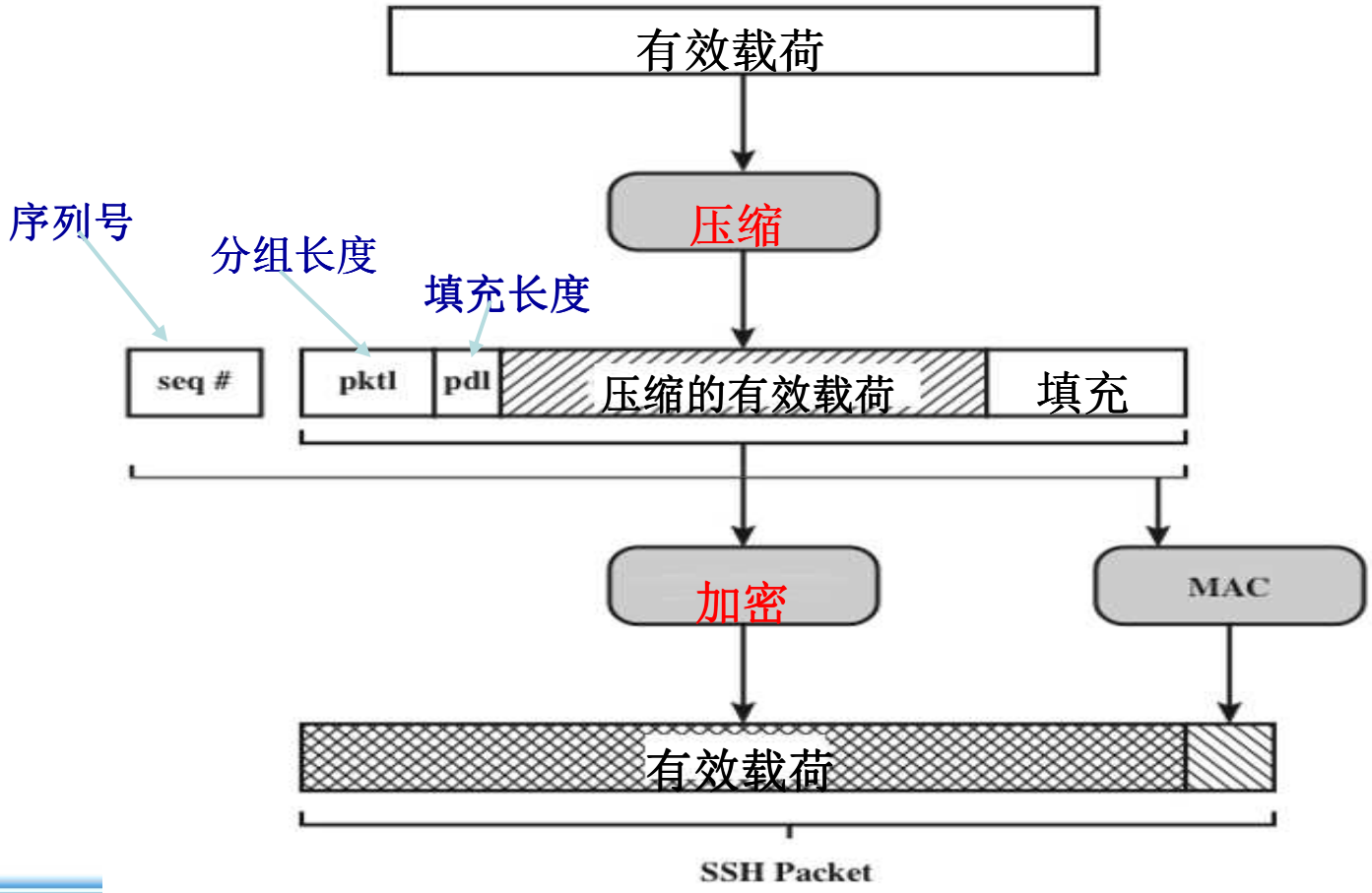


图6.10 SSH传输层协议分组信息

6.5.2 SSH身份认证协议

- 身份认证协议在传输层协议所建立的安全通道上运行
- 客户向服务器提出用户身份认证服务请求
  - 若服务器接受这个请求，双方即可开始执行SSH身份认证协议
  - 当一次身份认证失败时，客户端可以再次提出认证请求，但重试的时间间隔和次数有限
  - 如果在10分钟之内没有成功完成认证，或重试次数已经超过20次，服务器会返回SSH_MSG_DISCONNECT消息并断开连接
- 在认证成功后的通信过程中，客户端也可以随时提出新的认证请求

6.5.2 SSH身份认证协议（续）

- 在认证时，客户端发送SSH_MSG_USERAUTH_REQUEST消息，后面跟随下列内容：
  - 用户名
  - 服务名
  - 方法名
  - 其他和方法相关的域

身份认证方法

公开密钥

- 客户端向服务器发送包含客户端公钥的消息，消息由客户端私钥签名
- 当服务器收到此消息时，它会检查提供的密钥是否可用于身份验证，如果是，则检查签名是否正确

口令密码

- 客户端发送包含明文形式的密码消息，该密码受传输层协议加密保护

基于主机

- 身份验证在客户端的主机上执行，而不是在客户端本身上执行。
- 此方法的工作原理是让客户端发送使用客户端主机的私钥创建的签名。
- SSH服务器不是直接验证用户的身份，而是验证客户端主机的身份。

6.5.3 SSH连接协议

- SSH连接协议提供交互的登录会话、执行远程命令、转发TCP/IP连接和转发X11连接。这个协议的服务名字是ssh-connection。
- 由于连接协议的目的是把已经加密的隧道提供给多个应用程序复用，因此它需要一个能区分不同应用程序的方法
  - 引入了通道（channel）的机制，所有的终端会话、转接连接都是通道。
  - 多个通道被复用成一个连接。对于每一端来说，通道用数字来标识。在两端标明同一个通道的数字可能不同。
  - 当一个通道打开时，请求打开通道的消息同时会包含发送方的通道号。接收方也给新的通道分配一个自己的通道号。
  - 在以后的通信过程中，只要让这两个通道号一一对应就可以了。

6.5.3 SSH连接协议（续）

- 如果向对方请求打开一个通道，则需要发送一个SSH_MSG_CHANNEL_OPEN消息，同时还要告诉对方自己的通道号和初始的窗口大小，因此会有如下内容：
  - SSH_MSG_CHANNEL_OPEN。
  - 通道类型。
  - 发送方通道号。
  - 初始窗口大小。
  - 最大包大小。
  - 和通道类型相关的其他内容。

SSH连接协议

- SSH连接协议在SSH传输层协议之上运行，并假定正在使用安全身份验证连接。
  - 安全认证连接（称为隧道）由连接协议用于多路复用多个逻辑信道。
- 通道机制
  - 使用单独的通道支持使用SSH的所有类型的通信。
  - 任何一方都可以打开一个信道。
  - 对于每个信道，每一方都关联一个唯一的信道号。
  - 使用窗口机制对通道进行流量控制。
  - 在收到消息以指示窗口空间可用之前，不会向通道发送任何数据。
  - 信道的生命周期分为三个阶段：打开信道，数据传输和关闭信道。

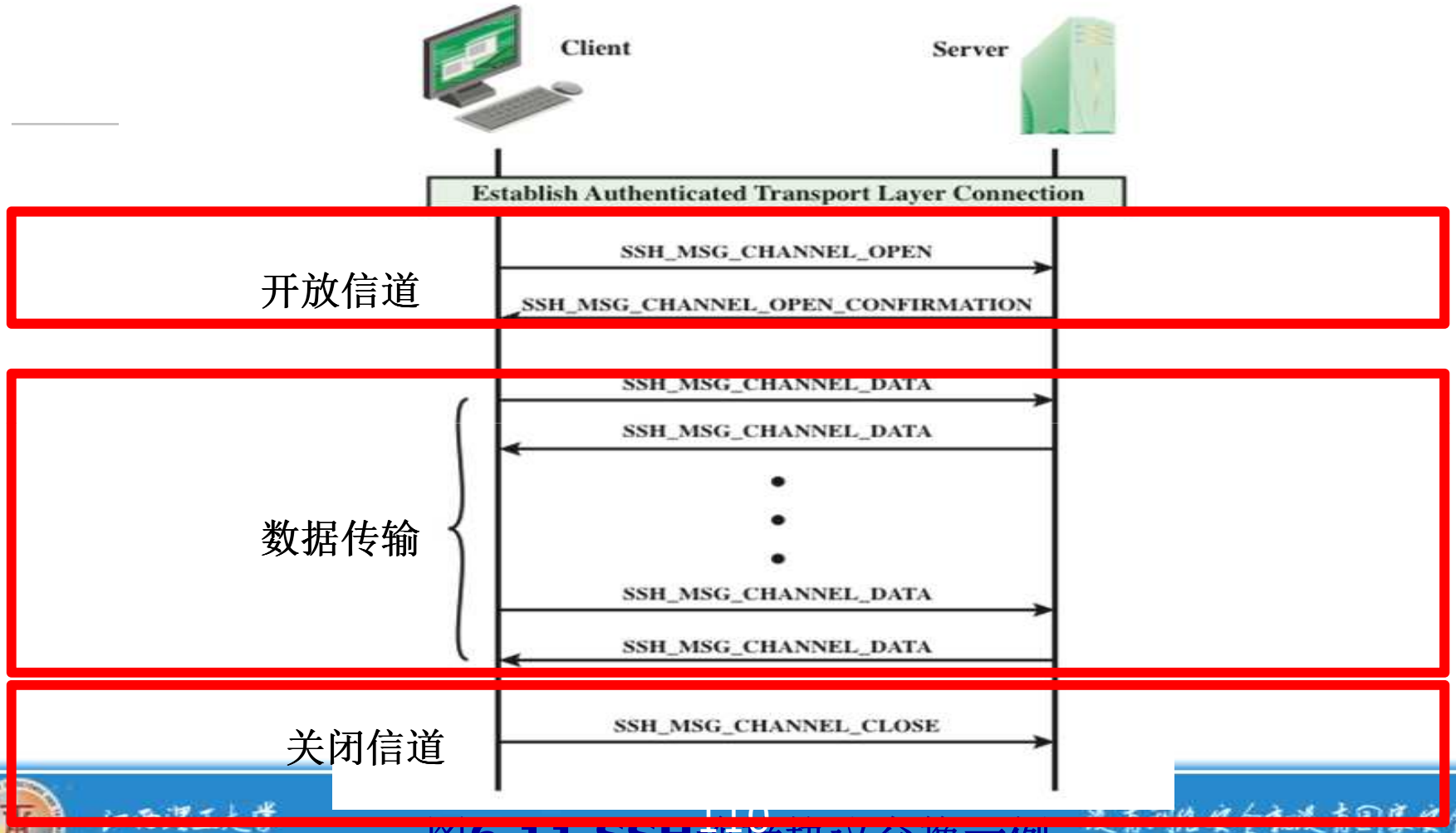


图6.11 SSH连接协议交换示例

信道类型

会话

- 远程执行程序。
- 该程序可以是shell，诸如文件传输或电子邮件的应用程序，系统命令等子系统。
- 打开会话通道后，后续请求将用于启动远程程序。

X11

- 指X 窗口系统，一种为网络计算机提供图形用户界面（GUI）的计算机软件系统和网络协议。
- X允许应用程序在网络服务器上运行，但可以在台式机上显示。

前向-tcpip

- 远程端口转发

直接-tcpip

- 本地端口转发

端口转发

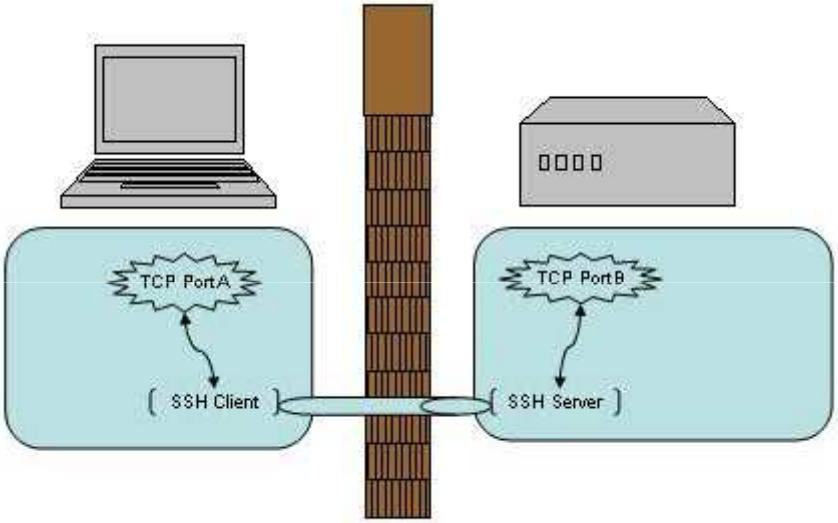
- **SSH最有用的特征之一就是端口转发**
  - 加密 SSH Client 端至 SSH Server 端之间的通讯数据
  - 突破防火墙的限制完成一些之前无法建立的 TCP 连接
  - **提供将任何不安全的TCP连接转换为安全SSH连接，这也称为SSH隧道技术**
  - **传入TCP流量根据端口号传送到适当的应用程序（端口是TCP用户的标识符）**
  - **应用程序可以使用多个端口号**

(a) TCP连接

(b) SSH隧道连接

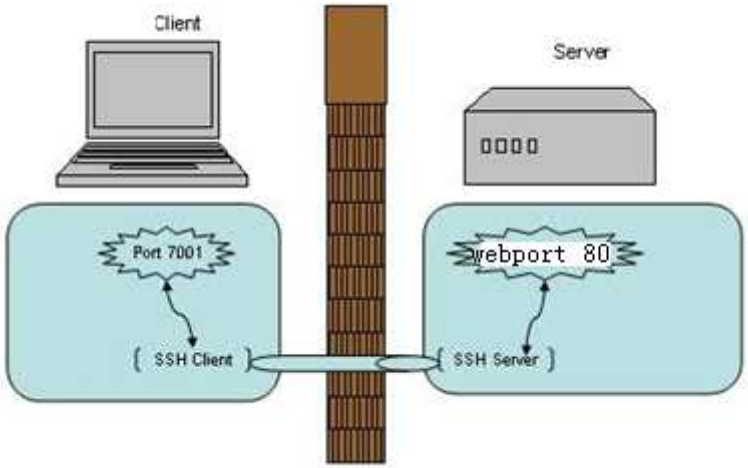
图6.12 SSH传输层包交换

端口转发



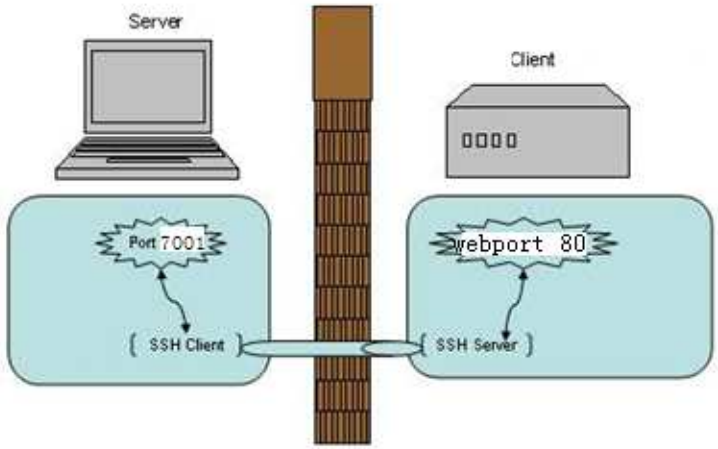
本地转发

- **例子：** Web服务器端口未对外网开放
  - **ssh -L [localhost]:<local port>:<remote host>:<remote port> <SSH hostname>**
  - ssh -L 7001:localhost:80 SSHServerHost
  - 将发送到本地7001的数据转发到服务器的80端口



远程转发

- **例子： Client 上开放Web服务（假设端口7001），但没有公网地址**
  - **ssh -R <remote host>:<remote port>:<local port> <SSH hostname>**
  - ssh -R 80:localhost:7001 SSHServerHost
  - 将发往远端SSH服务器80端口的数据转发本地7001端口



6.5.4 SSH协议的应用

- **SSH协议发布了两种版本**
  - 即版本1（SSH1.5协议）和版本2（SSH2.0协议）
  - 版本1是一个完全免费的软件包，包含几种专利算法（但其中有几种已经过期）且存在一些明显的安全漏洞（如允许在数据流中插入数据）
  - 而版本2安全性得到较大的提高，但在商业使用时则要付费
- **SSH最常见的应用**
  - 用它来取代传统的Telnet、FTP等网络应用程序，通过SSH登录到远方机器执行各种命令
  - 在不安全的网路通信环境中，它提供了验证机制与非常安全的通信环境。

6.5.4 SSH协议的应用（续）

- **安全远程登录**
  - 用户可以用SSH完成TELNET、RLOGIN能够完成的任何事情。登录后所有的通信数据都受到加密保护。
- **TCP端口转发**
  - 利用SSH既可以进行本地端口的流量转发，也可以进行远程端口的流量转发，甚至可以结合PPP协议组建虚拟专用网
- **安全远程执行命令**
  - 使用SSH协议，同样可调用shell程序，由于建立连接之后的所有数据都经过加密，因此在SSH建立连接后，远程执行命令时所有的通信都被加密
- **安全远程文件传输**
  - SSH允许通过客户端程序SCP进行文件的远程复制。在SSH协议版本2中更提供了SFTP的安全文件传输服务
- **X窗口连接转发**
  - SSH提供的一个重要功能就是X转发功能，它可以在客户端的显示屏上把服务器端X程序的运行结构以图形形式现实在客户端

6.5.5 SSH安全性分析

- **SSH是一种通用，功能强大的基于软件的网络安全解决方案**
  - **计算机每次向网络发送数据时，SSH都会自动对其进行加密**
  - **数据到达目的地时，SSH自动对加密数据进行解密**
- **整个过程都是透明的**
- **它使用了现代的安全加密算法，足以胜任大型公司的任务繁重的应用程序的要求**

6.5.5 SSH安全性分析（续）

1．SSH协议的主要安全特性和优点：

- 使用强加密技术来保证数据的私密性。端到端通信用随机密钥进行加密，随机密钥为会话进行安全协商，会话结束后被丢弃。支持的算法有DES，IDEA，3DES等。
- 通信完整性，确保通信不会被修改。SSH-2基于MD5 和 SHA-1的加密hash算法。
- 认证，即发送者和接收者的身份证明。客户机和服务器双向认证。
- 授权，即对账号进行访问控制。
- 使用转发或隧道技术对其它基于TCP/IP的会话进行加密。

6.5.5 SSH安全性分析（续）

2．SSH可以防止的攻击

- **网络窃听**，SSH通信是加密的，即使截获会话内容，也不能将其解密。
- **名字服务和IP伪装**，SSH通过加密验证服务器主机身份可避免这类风险。
- **连接劫持**，SSH的完整性检测负责确定会话在传输过程是否被修改，如果被修改过，就关闭连接。
- **中间人攻击**，SSH利用两种方法防止这种攻击，第一种是服务器主机认证。第二种是限制使用容易受到这种攻击的认证方法，密码认证容易受到中间人攻击。
- **插入攻击**，这种攻击可以客户和服务器之间发送的正文数据流之间插入任意数据。ssh1 1.2.25后和Openssh的所有版本都专门进行了设计，来检测并防止这种攻击。这种检测程序增大了插入攻击的难度，但是并不能完全防止。SSH2使用强加密完整性检测手段来防止这个问题。可以用3DES算法来防止这种攻击。

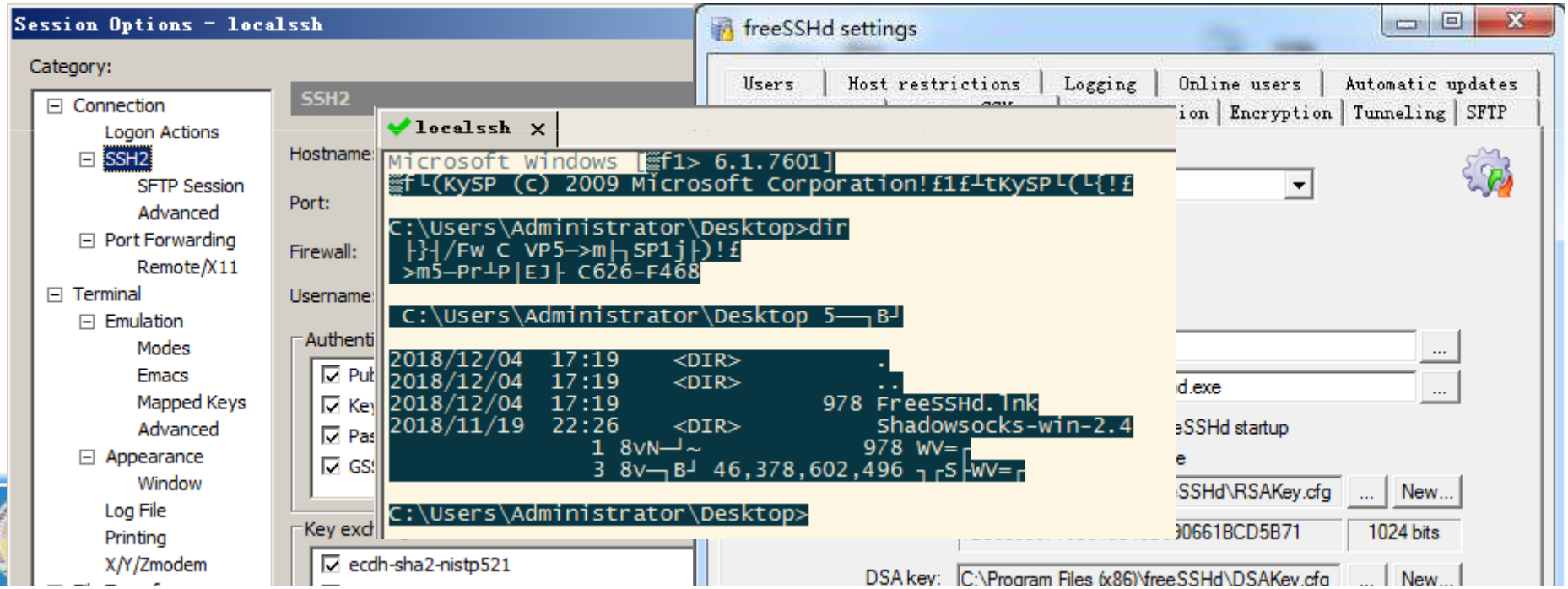
6.5.5 SSH安全性分析（续）

3 .SSH不能防止的攻击

- **密码崩溃**，密码认证是一种脆弱的认证形式，尽量使用公钥认证方式。如果必须要密码认证，可考虑使用S/Key之类的一次性密码机制。
- **IP AND TCP攻击**，由于SSH是在TCP之上进行操作的，因此容易受到针对TCP和IP缺陷而发起的攻击。SYN flood，TCP不同步和TCP劫持等。只能通过更低层的防护措施来保护。
- **流量分析。**
- **隐秘通道。**
- **粗心大意。**安全是一个过程，而不是一个产品，不要认为装上SSH就安全了。

SSH配置举例

- **SecureCRT(客户端)**
- **freesshd**



SSH抓包

```
192.168.159.1   192.168.159.142     TCP          66 37056 > ssh [SYN] Seq=4003559975 Win=8192 Len=0 MSS=1460 WS=
192.168.159.142 192.168.159.1       TCP          66 ssh > 37056 [SYN, ACK] Seq=2462115 Ack=4003559976 Win=8192 L
⊞ Frame 14: 750 bytes on wire (6000 bits), 750 bytes captured (6000 bits) on interface 0
⊞ Ethernet II, Src: Vmware_c7:fb:48 (00:0c:29:c7:fb:48), Dst: Vmware_c0:00:08 (00:50:56:c0:00:08)
⊞ Internet Protocol Version 4, Src: 192.168.159.142 (192.168.159.142), Dst: 192.168.159.1 (192.168.159.1)
⊞ Transmission Control Protocol, Src Port: ssh (22), Dst Port: 37056 (37056), Seq: 2462140, Ack: 4003560018, Len: 6
⊟ SSH Protocol
  ⊟ SSH Version 2 (encryption:aes128-cbc mac:hmac-sha2-256 compression:none)
      Packet Length: 692
      Padding Length: 7
    ⊟ Key Exchange
        Message Code: Key Exchange Init (20)
      ⊟ Algorithms
          Cookie: e2e066429942ecf107c9251a52d19285
          kex_algorithms length: 111
          kex_algorithms string: ecdh-sha2-nistp256,ecdh-sha2-nistp384,ecdh-sha2-nistp521,diffie-hellman-group1-sha
          server_host_key_algorithms length: 15
          server_host_key_algorithms string: ssh-rsa,ssh-dss
          encryption_algorithms_client_to_server length: 175
          encryption_algorithms_client_to_server string: aes128-cbc,aes128-ctr,3des-cbc,blowfish-cbc,aes192-cbc,aes
          encryption_algorithms_server_to_client length: 175
          encryption_algorithms_server_to_client string: aes128-cbc,aes128-ctr,3des-cbc,blowfish-cbc,aes192-cbc,aes
          mac_algorithms_client_to_server length: 64
          mac_algorithms_client_to_server string: hmac-sha2-256,hmac-sha2-512,hmac-sha1,hmac-sha1-96,hmac-md5,none
          mac_algorithms_server_to_client length: 64
          mac_algorithms_server_to_client string: hmac-sha2-256,hmac-sha2-512,hmac-sha1,hmac-sha1-96,hmac-md5,none
          compression_algorithms_client_to_server length: 9
          compression_algorithms_client_to_server string: none,none
```

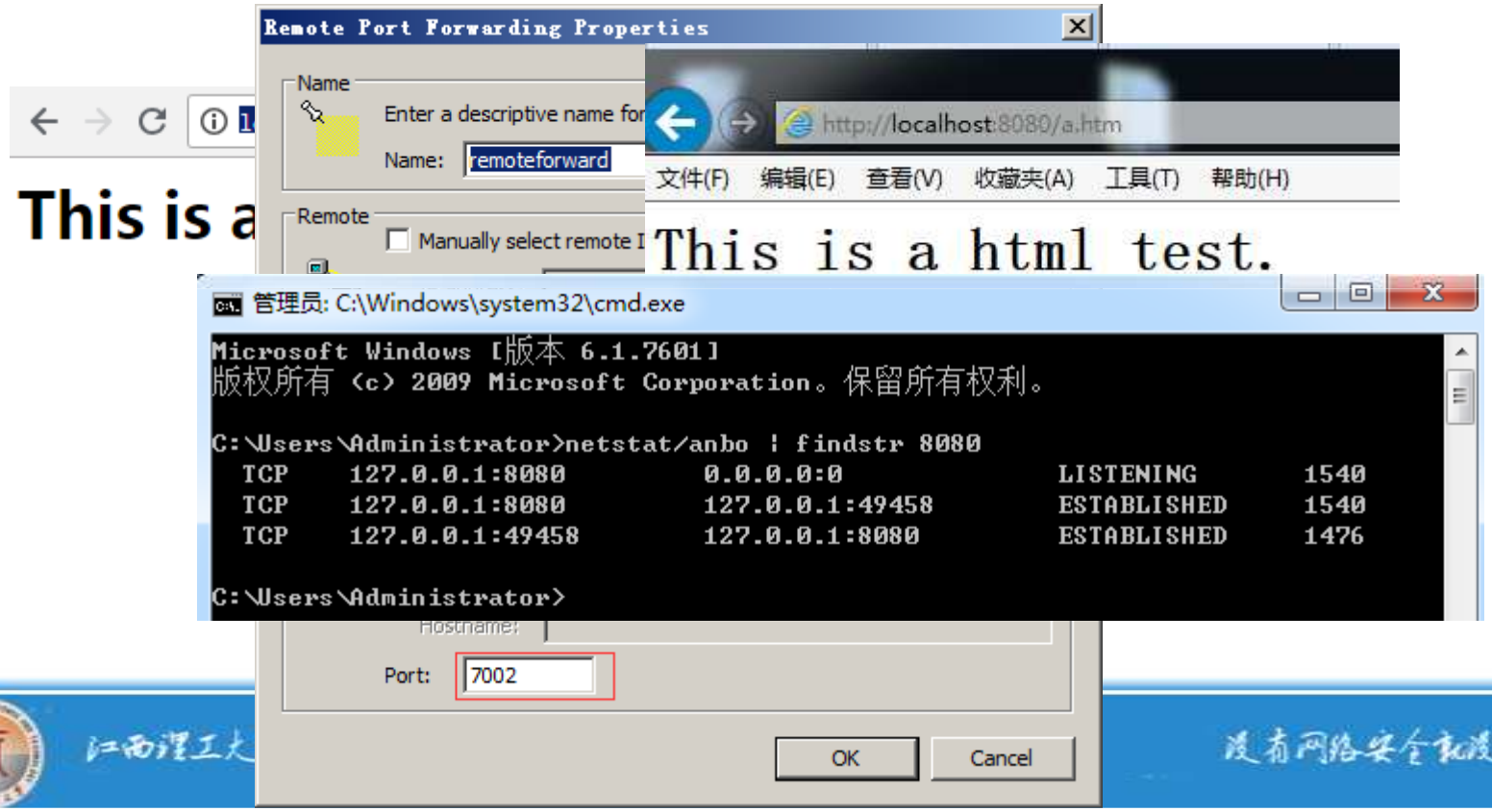
本地转发实例（服务器80端口被防火墙阻挡）

抓包

9	6.106759000	192.168.159.1	192.168.159.142	SSH	198 Encrypted request packet len=144
13	6.118742000	192.168.159.142	192.168.159.1	SSH	150 Encrypted response packet len=96
14	6.120236000	192.168.159.1	192.168.159.142	SSH	646 Encrypted request packet len=592
15	6.122591000	192.168.159.142	192.168.159.1	SSH	326 Encrypted response packet len=272
16	6.133828000	192.168.159.1	192.168.159.142	SSH	582 Encrypted request packet len=528
17	6.134443000	192.168.159.142	192.168.159.1	SSH	326 Encrypted response packet len=272

```
⊞ Frame 17: 326 bytes on wire (2608 bits), 326 bytes captured (2608 bits) on interface 0
⊞ Ethernet II, Src: Vmware_c7:fb:48 (00:0c:29:c7:fb:48), Dst: Vmware_c0:00:08 (00:50:56:c0:00:08)
⊞ Internet Protocol Version 4, Src: 192.168.159.142 (192.168.159.142), Dst: 192.168.159.1 (192.168.159.1)
⊞ Transmission Control Protocol, Src Port: ssh (22), Dst Port: 33836 (33836), Seq: 955355915, Ack: 4272876374, Len: 272
⊟ SSH Protocol
    Encrypted Packet: 7f453317a8768d6b38bd4d89d2157009922d41b5fa5e04f7...
```

远程转发实例


Remote Port Forwarding Properties
Name
Enter a descriptive name for
Name: remoteforward
Remote
Manually select remote
Port: 7002
OK
Cancel
This is a
http://localhost:8080/a.htm
文件(F) 编辑(E) 查看(V) 收藏夹(A) 工具(T) 帮助(H)
This is a html test.
管理员: C:\Windows\system32\cmd.exe
Microsoft Windows [版本 6.1.7601]
版权所有 (c) 2009 Microsoft Corporation。保留所有权利。
C:\Users\Administrator>netstat/anbo | findstr 8080
TCP 127.0.0.1:8080 0.0.0.0:0 LISTENING 1540
TCP 127.0.0.1:8080 127.0.0.1:49458 ESTABLISHED 1540
TCP 127.0.0.1:49458 127.0.0.1:8080 ESTABLISHED 1476
C:\Users\Administrator>

抓包

192.168.159.1	192.168.159.142	TCP	66 44869 > ssh [SYN] Seq=2313109311 Win=8192 Len=0 MSS=1460
192.168.159.142	192.168.159.1	TCP	66 ssh > 44869 [SYN, ACK] Seq=1070062041 Ack=2313109312 Win=
192.168.159.1	192.168.159.142	TCP	54 44869 > ssh [ACK] Seq=2313109312 Ack=1070062042 Win=65536
192.168.159.142	192.168.159.1	SSHv2	78 Server Protocol: SSH-2.0-WeOnlyDo 2.4.3\r
192.168.159.1	192.168.159.142	SSHv2	96 Client Protocol: SSH-2.0-SecureCRT_8.3.1 (x64 build 1537)
192.168.159.142	192.168.159.1	SSHv2	750 Server: Key Exchange Init
192.168.159.1	192.168.159.142	SSHv2	1030 Client: Key Exchange Init
192.168.159.1	192.168.159.142	SSHv2	206 Client: Diffie-Hellman Key Exchange Init
192.168.159.142	192.168.159.1	TCP	54 ssh > 44869 [ACK] Seq=1070062762 Ack=2313110482 Win=64512
192.168.159.142	192.168.159.1	SSHv2	502 Server: Diffie-Hellman Key Exchange Reply
192.168.159.1	192.168.159.142	SSHv2	70 Client: New Keys
192.168.159.142	192.168.159.1	SSHv2	70 Encrypted response packet len=16[Malformed Packet]
192.168.159.1	192.168.159.142	SSHv2	150 Encrypted request packet len=96[Malformed Packet]
192.168.159.142	192.168.159.1	SSHv2	150 Encrypted response packet len=96[Malformed Packet]
192.168.159.1	192.168.159.142	SSHv2	166 Encrypted request packet len=112[Malformed Packet]
192.168.159.142	192.168.159.1	SSHv2	182 Encrypted response packet len=128[Malformed Packet]

志存高远 责任为先

感谢聆听

网址：www.jxust. edu.cn
邮编：341000