中山大學
SUN YAT-SEN UNIVERSITY

Module II. Internet Security

Chapter 4
Introduction to Internet Security

Web: Theory & Applications

School of Data & Computer Science, Sun Yat-sen University

Outline

- **4.1 Network Security Architectures**
  - Five Layers of Network Security Architectures
  - Information Security Models
  - OSI/ISO 7498-2
  - ISO Security Services
  - ISO Security Mechanisms
- **4.2 IPSec**
  - Introduction
  - Some Basic Concepts About IPSec
  - ESP protocol
  - Gateway and Road Warrior mode
  - Key management of IPSec

Outline

- **4.3 SSL/TLS**
  - Introduction
  - How TLS Works
  - Decryption of TLS Packet
- **4.4 VPN**
  - Introduction to IPsec VPN
  - OpenVPN

4.3 SSL/TLS

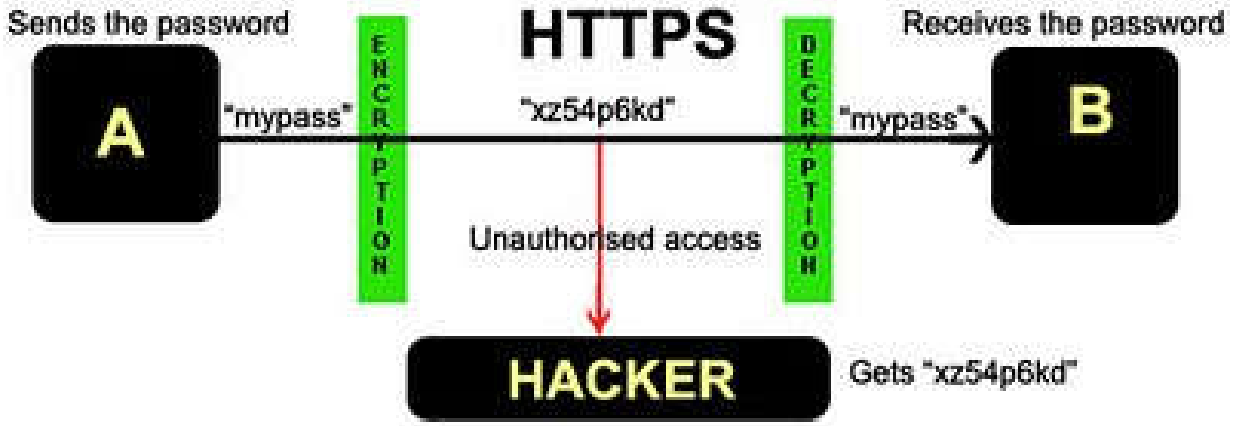
4.3.1 Introduction

- **Security in the Transport Layer**
  - Every thing will be transported in plain text!
    - ✧ *Example.* http
  - How to keep data in secret
    - ✧ Encrypt my data with *Symmetric Key* before sending it
    - ✧ Protect *Master Key* with my public key (PKI)
    - ✧ (Receiver) verify my *Public Key* with *Certificate* (by CA)
    - ✧ Prevent my data from tempering with *HMAC* (Hash-based Message Authentication Code)

4.3 SSL/TLS

4.3.1 Introduction

- **Security in the Transport Layer**
  - How to keep my data in secret
    - ✧ *Example*: https



4.3 SSL/TLS

4.3.1 Introduction

- **What is SSL/TLS**
  - SSL/TLS 是工作在传输层的协议，目的是防止数据被窃听和篡改。
    - ✧ SSL (Secure Sockets Layer) 最早由 NetScape 发明，其2.0版本被认为有安全缺陷， 3.0版本发布于1996年并得到广泛应用。后来 IEEE 对 SSL 进行标准化，成为 TLS (Transport Layer Security, 传输层安全协议)。
    - ✧ 最先发布的 TLS1.0 即是 SSL3.1。实际上 TLS1.0 并没有对 SSL3.0 做太大的改动，详细说明参见 RFC5246。
    - ✧ https 是 SSL/TLS 的一个应用。
      - ◌ https 使用443端口，区别于原来 http 的80端口
    - ✧ TLS 现在的公版是1.2版 (2008)，即 SSL3.3。更高级别的版本被禁止从美国出口。

4.3 SSL/TLS

4.3.1 Introduction

- **What is SSL/TLS**
  - A Simplified Picture of SSL
    1. Client sends out connection request.
    2. Server accepts the request and send back Server's public key PU(s).
    3. Client builds an E(PU(s), *K*) by encrypting a secret key *K* with PU(s) and send the E(PU(s), *K*) to the server.
    4. Server decrypts the E(PU(s), *K*) with its private key PR(s) and gets the secret key *K*.
    5. Client and serve start communicating with the secret key *K*.

4.3 SSL/TLS

4.3.1 Introduction

- **What is SSL/TLS**
  - 一个简化的 SSL/TLS 模型：
    1. 客户端向服务器发出连接请求。
    2. 服务器接收请求，向客户端发送服务器公钥 PU(s)。
    3. 客户将随机生成的密钥 *K* 用服务器的公钥加密，并将加密结果 E(PU(s), *K*) 发给服务器。
    4. 服务器用自己的私钥 PR(s) 解密 E(PU(s), *K*) 得到 *K*
    5. 通信的双方都已经得到了密钥 *K*，接下来的通信使用刚才协商的密钥 *K* 进行加密。

4.3 SSL/TLS

4.3.1 Introduction

- **What is SSL/TLS**
  - 一个简化的 SSL/TLS 模型：
    - ✧ 从上面的过程可以看出，SSL/TLS 其实与 IPsec 类似，两者处理的都是加密通信前密钥交换的问题，交换好密钥后再使用协商密钥来保护接下来的通信。
    - ✧ 在进行真正的 SSL/TLS 通信时，情况要复杂一些，例如，服务器不会简单的向客户发送一个公钥，而是发送一份数字证书。SSL/TLS 不仅仅要保证数据的保密性，还要保证传输数据的完整性，所以会用到 HMAC。
    - ✧ 考虑到 SSL 与 TLS 的密切关系，我们主要以 RFC5246 为基础，讨论 TLS 的实现。

4.3 SSL/TLS

4.3.2 How TLS Works

- **Some Basic Concepts**
  - Session
  - Connection
  - Write State
  - Read State
  - Cipher Suites
  - Pre-master Secret
  - Record Layer Protocol
  - Handshake Protocol
  - ChangeCipherSpec Protocol
  - Alert Protocol
  - Application Data Protocol

4.3 SSL/TLS

4.3.2 How TLS Works

- **一些基本概念**
  - Session 会话
    - ✧ 每一次 SSL 握手初始化都会创建一个 SSL 会话，一次握手恢复不会创建新的会话，而是继续沿用之前已经建立的会话。通常进行 SSL 通信的双方只会创建一个 SSL 会话。
  - Connection 连接
    - ✧ 每一个连接归属于某一个会话，而一个会话可以包含多个连接。通信的双方进行新的数据交换时只需要新建一个连接，而不需要重新建立会话，这样的设计可以减少握手开销。

4.3 SSL/TLS

4.3.2 How TLS Works

- **一些基本概念**
  - Write State 写模式
    - ✧ 在一次 SSL 会话中，通信双方都会分别纪录自己的写模式的信息。对于 *Alice* 与 *Bob* 的通信来说，*Alice* 的写模式描述了由 *Alice* 到 *Bob* 的通信方向，即*Alice* 所发送的消息所遵循的与安全通信相关的约定，例如加密算法和密钥。

4.3 SSL/TLS

4.3.2 How TLS Works

- **一些基本概念**
  - Read State 读模式
    - ✧ 同样，在一次 SSL 会话中，通信双方都会维护自己的读模式的信息。*Alice* 的读模式描述了由 *Bob* 到 *Alice* 的通信方向，即 *Alice* 所接收的消息所遵循的约定。
    - ✧ 读模式和写模式描述的是不同的通信方向，但读写模式是对应的。对于 *Alice* 与 *Bob* 的通信，*Alice* 的写模式就是 *Bob* 的读模式，两者纪录的信息 (密钥，算法) 完全一样。同样的，*Alice* 的读模式就是 *Bob* 的写模式。
    - ✧ *Alice* 的写模式和 *Bob* 的写模式可以不同，这意味着通信的两个方向可以采用不同的密钥和加密、散列算法。

4.3 SSL/TLS

4.3.2 How TLS Works

- **一些基本概念**
  - Cipher Suites 安全套件
    - ✧ 安全套件是通信过程中使用的各种加密算法与散列算法的集合，用于表示客户端所支持的加密与散列函数。安全套件中的每一个条目表示一种组合 (SSL 选项)，每种组合用两个字节表示，定义了会话的某个方向所使用的密钥交换算法、对称加密算法、密钥长度以及散列算法。
    - ✧ 例如 0x00, 0x3D 代表的就是

      TLS_RSA_WITH_AES_256_CBC_SHA256
  - Pre-Master Secret 预主密钥
    - ✧ 当通信双方协商选用 RSA 加密算法后，客户端会先生成一个预主密钥 PMS，用于计算后续的密钥。

4.3 SSL/TLS

4.3.2 How TLS Works

- **一些基本概念**
  - Record Layer Protocol 记录层协议
    - ✧ 记录层协议是 TLS 中所有协议的基础，所有的 TLS 通信都是通过这个协议来传输的，但它只是一个“外套”，其协议头包含了很少的信息，而数据段则负载了 TLS 的各种子协议。
    - ✧ 负载部分可以是用于通信过程控制的三个子协议之一，即握手协议、更改秘钥规格协议或者告警协议。
    - ✧ 当记录层协议的类型字段为23 (应用数据的标识) 时，负载部分是已加密的数据，这时记录层协议就是一个应用数据协议。

4.3 SSL/TLS

4.3.2 How TLS Works

- **一些基本概念**
  - TLS 的三个子协议
    - ✧ Handshake Protocol 握手协议
      - ◌ 握手协议主要用于正式通信前的密钥协商。
    - ✧ ChangeCipherSpec Protocol 更改秘钥规格协议
      - ◌ 更改秘钥规格协议是一个简单协议，它的负载部分只含有一个字节，其值取0或1，用于激活通信双方的读写模式。
    - ✧ Alert Protocol 告警协议
      - ◌ 告警协议用于传输通信过程中的错误信息。

4.3 SSL/TLS

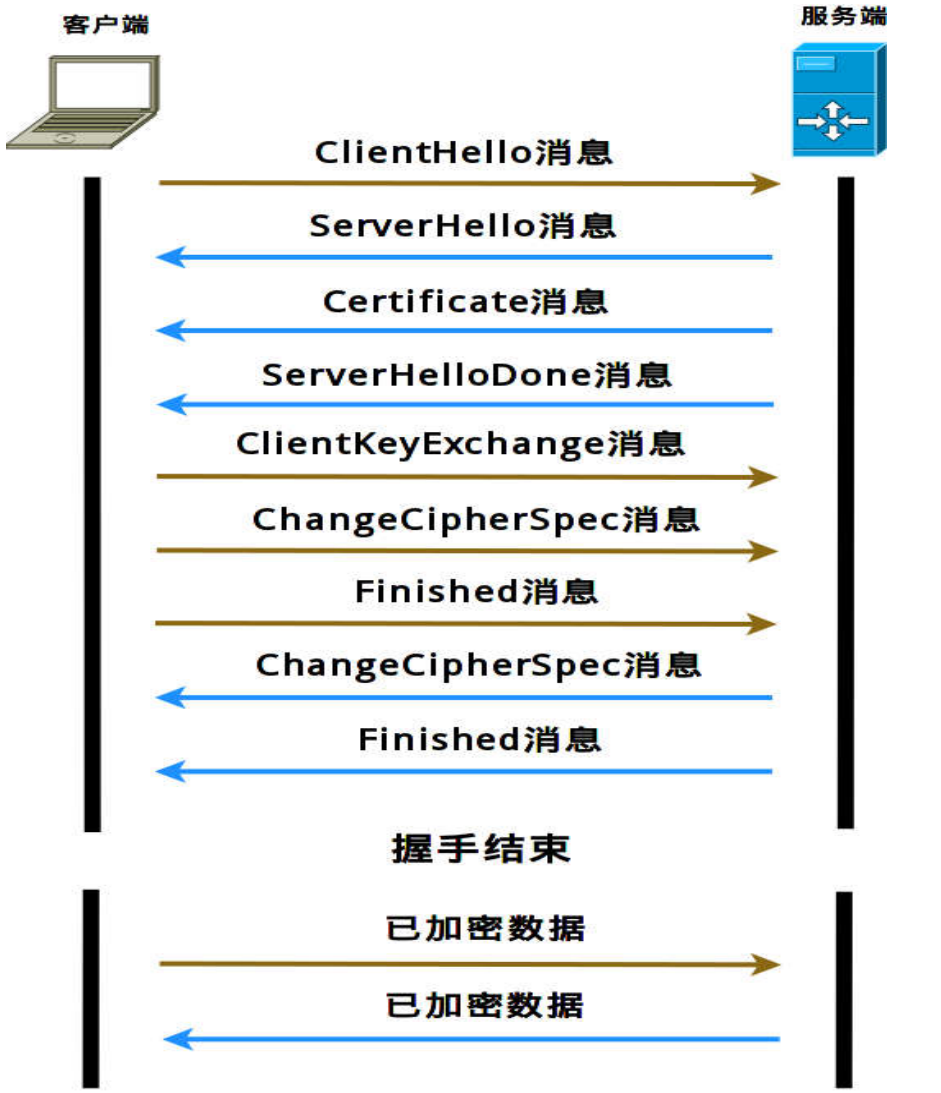
4.3.2 How TLS Works

- **一些基本概念**
  - Application Data Protocol 应用数据协议
    - ✧ 所有的受保护数据都将通过应用数据协议来传输，应用数据协议传输的是真正需要通信的数据。
    - ✧ 应用数据协议严格来讲不是一个单独的协议，而是记录层协议的一个类型。当记录层协议的类型字段填为23 (应用数据的标识)，而且负载部分填上已加密的数据后，它就是一个应用数据协议。
    - ✧ 应用数据协议在具体实现上与记录层协议同级，但在逻辑上又与其他3个子协议同级。因此逻辑上应用数据协议也可以理解为记录层协议的一个子协议。

4.3 SSL/TLS

4.3.2 How TLS Works

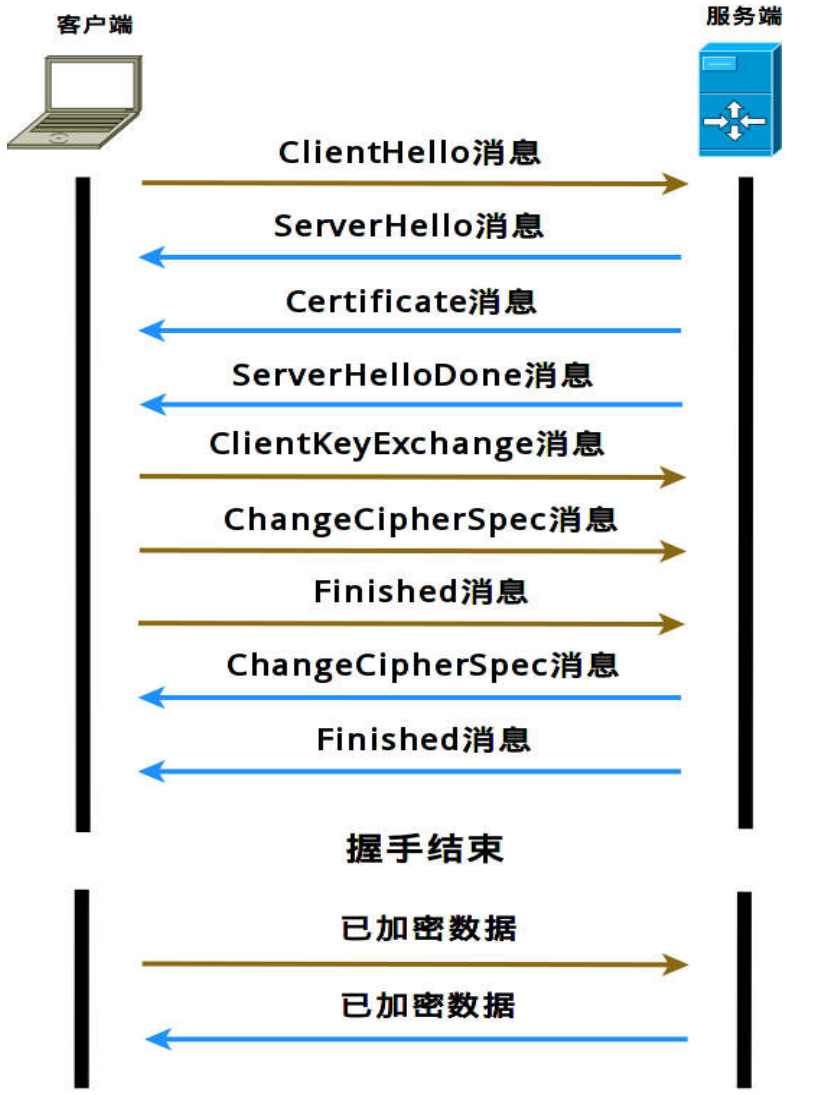
- **TLS Handshake**
  - ClientHello
  - ServerHello
  - Certificate
  - ServerHelloDone
  - ClientKeyExchange
  - ChangeCipherSpec
  - Finished
  - ChangeCipherSpec
  - Finished



4.3 SSL/TLS

4.3.2 How TLS Works

- **TLS Handshake**
  - Note.
    - ✧ 从简化的 SSL/TLS 模型可以看出，建立会话的很大部分耗费用在了密钥协商这一部分，这一过程被称为 SSL/TLS 握手。一旦握手成功，意味着通信双方都已经安全的得到了共享密钥。



4.3 SSL/TLS

4.3.2 How TLS Works

- **TLS 握手机制 (Examples on RSA)**
  - 客户端：ClientHello
    - ✧ 客户端向服务端器发出连接请求。ClientHello 消息里面还包含了客户端生成的随机数 (后面计算主密钥要用到)、会话 ID (Session ID) 以及客户端所支持的 SSL 选项。初次握手时，会话 ID 为0，表示客户端要求建立新的会话。
  - 服务端：ServerHello
    - ✧ 服务端回应请求，从客户端提供的 SSL 选项中选取一个并告知客户端。
    - ✧ 由于 ClientHello 消息的会话 ID 为0，所以服务端会返回一个新的会话 ID，用于标识本次会话。同时服务端也会生成一个自己的随机数并发送给客户端。

4.3 SSL/TLS

4.3.2 How TLS Works

- **TLS 握手机制 (Examples on RSA)**
  - 服务端：Certificate
    - ✧ 在发送 ServerHello 消息后服务端会发送自己的公钥信息。当选用 RSA 算法时，服务端会发送自己的证书。
  - 服务端：ServerHelloDone
    - ✧ 服务端通知客户端它已经完成密钥交换的准备工作，本身不含任何特殊的信息。注意到直到这一步，所有的信息都是没有加密的。客户端收到这个消息后可以开始与服务端交换密钥。
  - 客户端：ClientKeyExchange
    - ✧ 由于选用的是 RSA 算法，客户端在这一步会随机生成一个预主密钥 PMS，用服务端的公钥加密并将加密结果发给服务端。至此，SSL 的初期协商结束。

4.3.2 How TLS Works

- **TLS 握手机制 (Examples on RSA)**
  - 客户端：ChangeCipherSpec
    - ✧ 客户端为之后的会话激活自己的写模式选项，并通知服务端激活其读模式的选项，包括密钥、加密算法、HMAC 算法等。
    - ✧ 这些选项在 Hello 消息和 ClientkeyExchange 消息已经协商过，协商结果在这之前处于待定 (Pending) 状态，只有在这个消息发出去后所有协商的选项才会被真正的激活。
    - ✧ 注意到 ChangeCipherSpec 信息由 ChangeCipherSpec 协议定义，它本身不属于握手协议的一部分，但是在握手过程中要用到。

4.3.2 How TLS Works

- **TLS 握手机制 (Examples on RSA)**
  - 客户端：Finished
    - ✧ 客户端告知服务端握手成功。Finished 消息通常紧接着 ChangeCipherSpec 消息被发送，同时它也是第一个使用协商好的密钥和加密算法加密的数据。该消息还包含对以下3个内容做的摘要：
      - ◌ 密钥信息
      - ◌ 以前所有握手信息的内容。攻击者插入或修改信息将导致的客户端和服务端两边的 hash 值不同
      - ◌ 一个指明发送方是服务端还是客户端的值
    - ✧ 这个摘要值可以保证握手过程的完整性，即中途没有被攻击者篡改。

4.3.2 How TLS Works

- **TLS 握手机制 (Examples on RSA)**
  - 服务端：ChangeCipherSpec
    - ✧ 与客户端发送的 ChangeCipherSpec 类似，服务端为之后的会话激活自己写模式的选项，也就是客户端读模式的选项。这里的服务端写模式选项可以与客户端的写模式，也就是服务端的读模式选项不同，即通信的两个方向可以选用不同的选项。
  - 服务端：Finished
    - ✧ 服务端告知客户端握手成功。附带的摘要信息如上所述。
  - 至此，SSL 握手结束，接下来的信息都可以使用握手协商得到的共享密钥来进行保护。

4.3 SSL/TLS

4.3.2 How TLS Works

- **Key Generation**
  - Client randomly generates a PMS
    - PMS (Pre-Master Secret) 预主密钥
  - Four keys are generated according to PMS
    - Encryption key for client to server
    - Encryption key for server to client
    - MAC key for client to server
    - MAC key for server to client

4.3 SSL/TLS

4.3.2 How TLS Works

- **Key Generation**
  - TLS 需要的密钥
    - ✧ 通信双方密钥协商成功后，可以利用密钥来保护真正的通信。
    - ✧ 通信的一方会有一个写模式和读模式，而每一种模式会采用对应的对称密钥和 HMAC 密钥，所以一共需要 2x2=4 个密钥。
      - ◌ 注意到不是 2x2x2=8 个密钥，因为其中一方的写模式就是另一方的读模式，所以只需要计算两个模式。
    - ✧ 前面握手的时候只协商了一个预主密钥 PMS。
  - 伪随机数函数 PRF
    - ✧ TLS 利用前面协商得到的 PMS 来计算出所需要的4个密钥。这其中起关键作用的就是伪随机数函数 PRF。下面是 RFC5246 中给出的其中一种 PRF 定义：

4.3 SSL/TLS

4.3.2 How TLS Works

- **Key Generation**
  - RFC5246 中给出的一种 PRF 定义：

```
if (i=0)
   A(i) = seed ;
else
   A(i) = HMAC_hash(secret, A(i-1)) ①;
P_hash(secret, seed) = HMAC_hash(secret, A(1) + seed) + ②
    HMAC_hash(secret, A(2) + seed) +
HMAC_hash(secret, A(3) + seed) + ... ③;
PRF(secret, label, seed) = P_hash(secret, label + seed) ;
```

① 定义中用到的 HMAC_hash 函数在 RFC5246 中使用 SHA-256。

② “+” 号表示字符串的连接。

③ PRF 生成的随机数由多个散列值叠代连接而成，散列值的个数 (也就是叠代的次数) 由所需要的数据块长度决定。

4.3 SSL/TLS

4.3.2 How TLS Works

- **Key Generation**
  - 主密钥的计算：

key_block0 = PRF(PMS①, “master key” ②, 客户端随机数 + 服务端随机数)

MS = key_block0[0..47]③

① Key_block0 是以 PMS 为种子应用 PRF 生成的一个数据段。
② 这里的 “master key” 就是字符串 “master key” 的 ASCII 码串。
③ 取 key_block0 的前48个字节得到主密钥 MS。

4.3 SSL/TLS

4.3.2 How TLS Works

- **Key Generation**
  - 密钥块的计算：

key_block = PRF(MS①, “key expansion” ②, 客户端随机数 +服务端随机数)

① Key_block 是以 MS 为种子应用 PRF 生成的一个数据段。
② 这里的 “key expansion” 就是字符串 “key expansion” 的 ASCII 码串。
③ 从 key_block 中顺序划分出4个密钥和2个初始向量 IV (选用块加密算法时需要用到 IV)，剩余部分被舍弃。

客户端MAC写密钥 client_write_MAC	服务端MAC写密钥 server_write_MAC	客户端写密钥 client_write_KEY	服务端写密钥 server_write_KEY	客户端写IV client_write_IV	服务端写IV server_write_IV	剩余部分... remaining...

✧ 现在已经得到了所需的4个密钥，可以用以保护传输的数据。
✧ TLS 使用应用数据协议来进行加密数据的传输。

4.3.2 How TLS Works

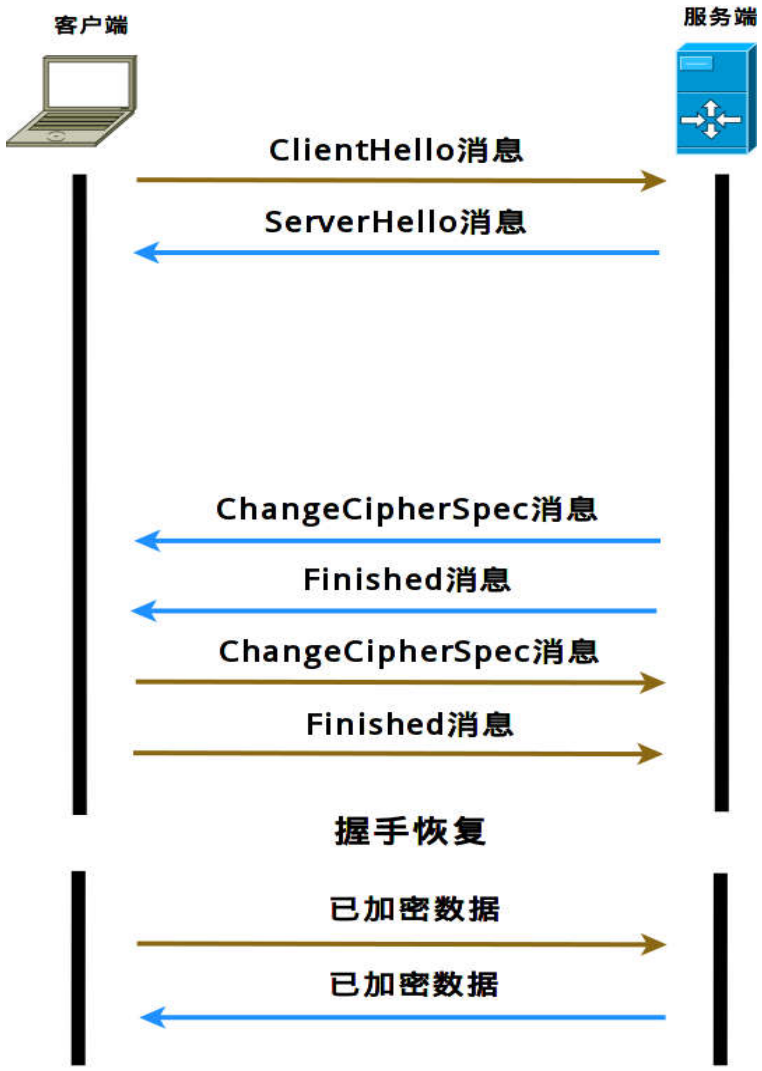
- 应用数据协议
  - 类型为23的 TLS 记录层协议就是应用数据协议，结构如下：

0			5
23 (Content Type)	主版本号 Major version	子版本号 Minor version	长度 Length
数据内容 (已加密)			
HMAC (已加密)			

4.3 SSL/TLS

4.3.2 How TLS Works

- **Resume of TLS Handshake**
  - ClientHello
  - ServerHello
  - ChangeCipherSpec
  - Finished
  - ChangeCipherSpec
  - Finished



4.3 SSL/TLS

4.3.2 How TLS Works

- **TLS 握手恢复机制**
  - TLS 握手过程比较繁复，所以 TLS 提供了会话恢复机制以避免每次进行信息交换都要重复握手。
  - 客户端：ClientHello
    - ✧ 客户端发送会话请求，与首次握手类似，不同的是客户端会在会话 ID 字段填上首次握手建立的会话 ID。
  - 服务端：ServerHello
    - ✧ 服务端回复请求，确认会话 ID 的有效性。
  - 服务端：ChangeCipherSpec
    - ✧ 服务端发送 ChangeCipherSpec 消息来重新激活自己的写模式选项，并通知客户端重新激活它的读模式。

4.3.2 How TLS Works

- **TLS 握手恢复机制**
  - 服务端：Finished
    - ✧ 服务端告知客户端握手成功。
  - 客户端：ChangeCipherSpec
    - ✧ 客户端发送 ChangeCipherSpec 消息来重新激活自己的写模式选项，并通知服务端重新激活它的读模式。
  - 客户端：Finished
    - ✧ 客户端告知服务端握手成功。
  - 上述过程可见，恢复握手时省略了密钥协商部分，直接使用以前协商的密钥和算法，这样可以减少密钥计算的消耗。

4.3 SSL/TLS

4.3.3 Decryption of TLS Packet

- Reading …

Outline

- **4.1 Network Security Architectures**
  - Five Layers of Network Security Architectures
  - Information Security Models
  - OSI/ISO 7498-2
  - ISO Security Services
  - ISO Security Mechanisms
- **4.2 IPSec**
  - Introduction
  - Some Basic Concepts About IPSec
  - ESP protocol
  - Gateway and Road Warrior mode
  - Key management of IPSec

Outline

- 4.3 SSL/TLS
  - Introduction
  - How TLS Works
  - Decryption of TLS Packet
- **4.4 VPN**
  - Introduction to IPsec VPN
  - OpenVPN

4.4 VPN

4.4.1 Introduction to IPsec VPN

- **VPN**
  - Virtual Private Network
- **IPsec VPN on GNU/Linux**
  - FreeS/WAN
  - Openswan

4.4 VPN

4.4.1 Introduction to IPsec VPN

- **VPN**
  - 虚拟私有网络
- **IPsec VPN**
  - IPsec VPN 利用 IPsec 架构保护私有网络的通信，从而在逻辑上将私有网络与 Internet 隔离。
  - 基于 GNU/Linux 提供 IPsec VPN 服务的软件
    - ✧ FreeS/WAN
    - ✧ Openswan

4.4 VPN

4.4.2 OpenVPN

- **OpenVPN**
  - an open source software application under the GNU General Public License (GPL) by James Yonan
  - implementing virtual private network (VPN) techniques for creating secure point-to-point or site-to-site connections in routed or bridged configurations and remote access facilities.
  - using a custom security protocol that utilizes SSL/TLS for key exchange
  - capable of traversing Network Address Translators (NATs) and firewalls

4.4 VPN

4.4.2 OpenVPN

- **OpenVPN**
  - OpenVPN allows peers to authenticate each other using a pre-shared secret key, certificates, or username/password. When used in a multi-client-server configuration, it allows the server to release an authentication certificate for every client, using signature and Certificate authority. It uses the OpenSSL encryption library extensively, as well as the SSLv3/TLSv1 protocol, and contains many security and control features.

4.4 VPN

4.4.2 OpenVPN

- **OpenVPN**
  - OpenVPN 工作在传输层，借助的是 SSL 协议。
  - 在使用 OpenVPN 服务时，需要建立一张虚拟网卡。
  - OpenVPN 程序作为一个中间者，负责协调虚拟网卡和物理网卡之间的数据转换，所有接收到的数据都会先经过物理网卡，而所有发送的数据最终由物理网卡出去。

4.4 VPN

4.4.2 OpenVPN

- **OpenVPN**
  - 发送数据流程：
    - ✧ 应用程序将数据发送到 TCP/IP 协议栈；
    - ✧ TCP/IP 协议栈将数据发送到虚拟网卡；
    - ✧ OpenVPN 检查到达虚拟网卡的数据，将数据取出作加密保护；
    - ✧ OpenVPN 将保护好的数据发送至 TCP/IP 协议栈；
    - ✧ TCP/IP 协议栈将数据发送给物理网卡；
    - ✧ 物理网卡收到要发送的数据并真正将数据发出。

4.4 VPN

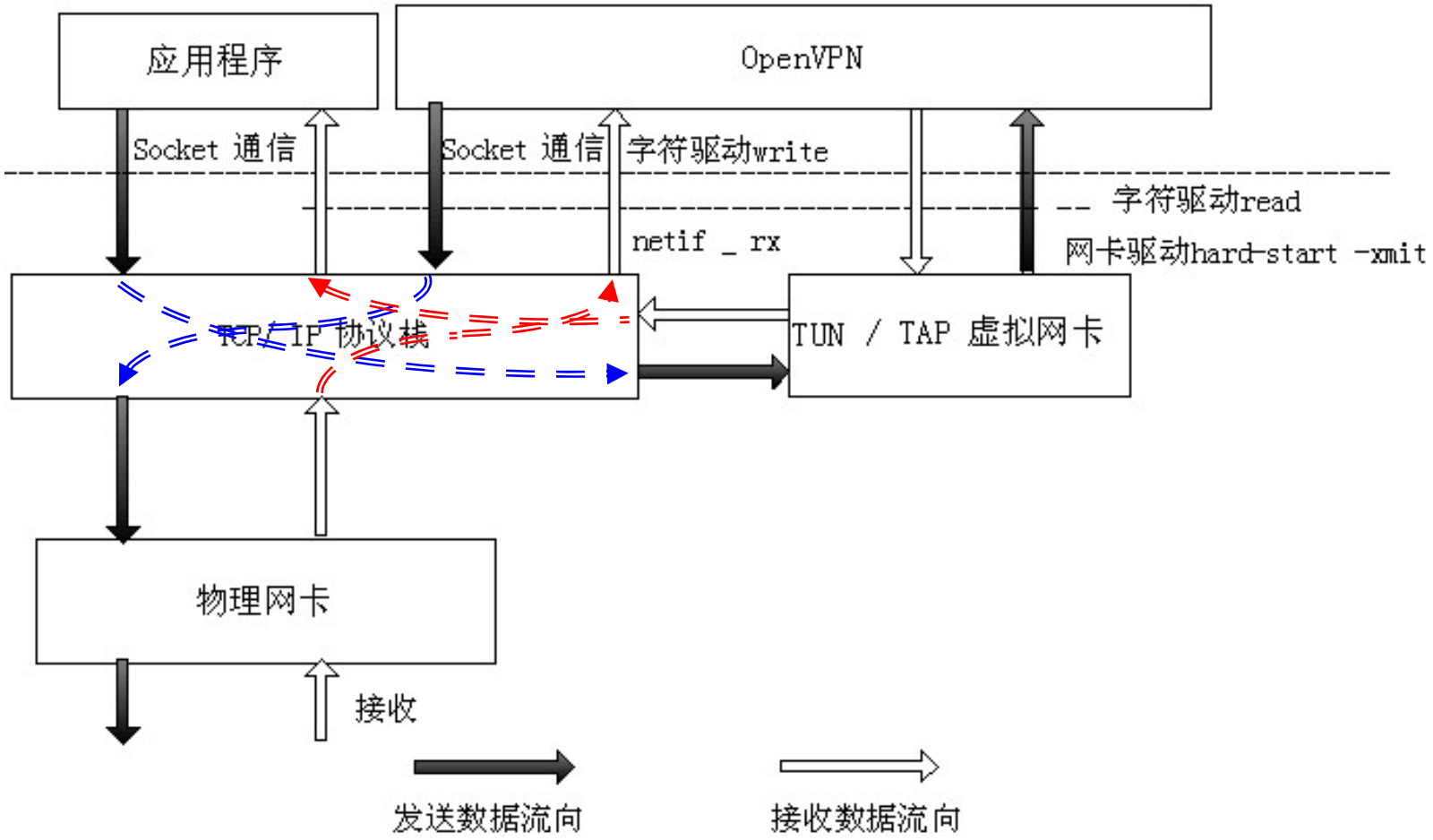
4.4.2 OpenVPN

- **OpenVPN**
  - 接收数据流程：
    - ✧ 物理网卡接收到传来的数据，交付 TCP/IP 协议栈；
    - ✧ OpenVPN 通过 Socket 通信检查接收到的数据，如果数据被加密过，则进行处理，否则抛弃；
    - ✧ OpenVPN 将解密后的明文报文发送至虚拟网卡；
    - ✧ 虚拟网卡将明文报文发送至 TCP/IP 协议栈；
    - ✧ 应用程序通过 Socket 从 TCP/IP 协议栈中读取信息。

4.4 VPN

4.4.2 OpenVPN

- **OpenVPN**



中山大學 SUN YAT-SEN UNIVERSITY

References

1. James F. Kurose and Keith W. Ross. Computer Networking - A Top-Down Approach 5th ed. Addison-Wesley, 2009.
2. James S. Tiller. A Technical Guide to IPsec Virtual Private Networks. Auerbach Publications, 2001.
3. Stephen A. Thomas. SSL and TLS Essentials. Wiley, 2000.
4. Wikipedia.Ipsec. http://en.wikipedia.org/wiki/IPsec
5. Steve Friedl. An Illustrated Guide to IPsec. http://www.unixwiz.net/techtips/iguide-ipsec.html
6. Martti Kuparinen and Oy L M Ericsson Ab. Isakmp and IKE. http://www.tml.tkk.fi/Opinnot/Tik-110.501/1998/papers/16isakmp/isakmp.html
7. Richard A. Deal. Isakmp/IKE Phase 1 Connections. http://fengnet.com/book/vpnconf/ch19lev1sec1.htm
8. Vicky. SSL Handshake. http://developer.connectopensource.org/display/CONNECTWIKI/SSL+Handshake

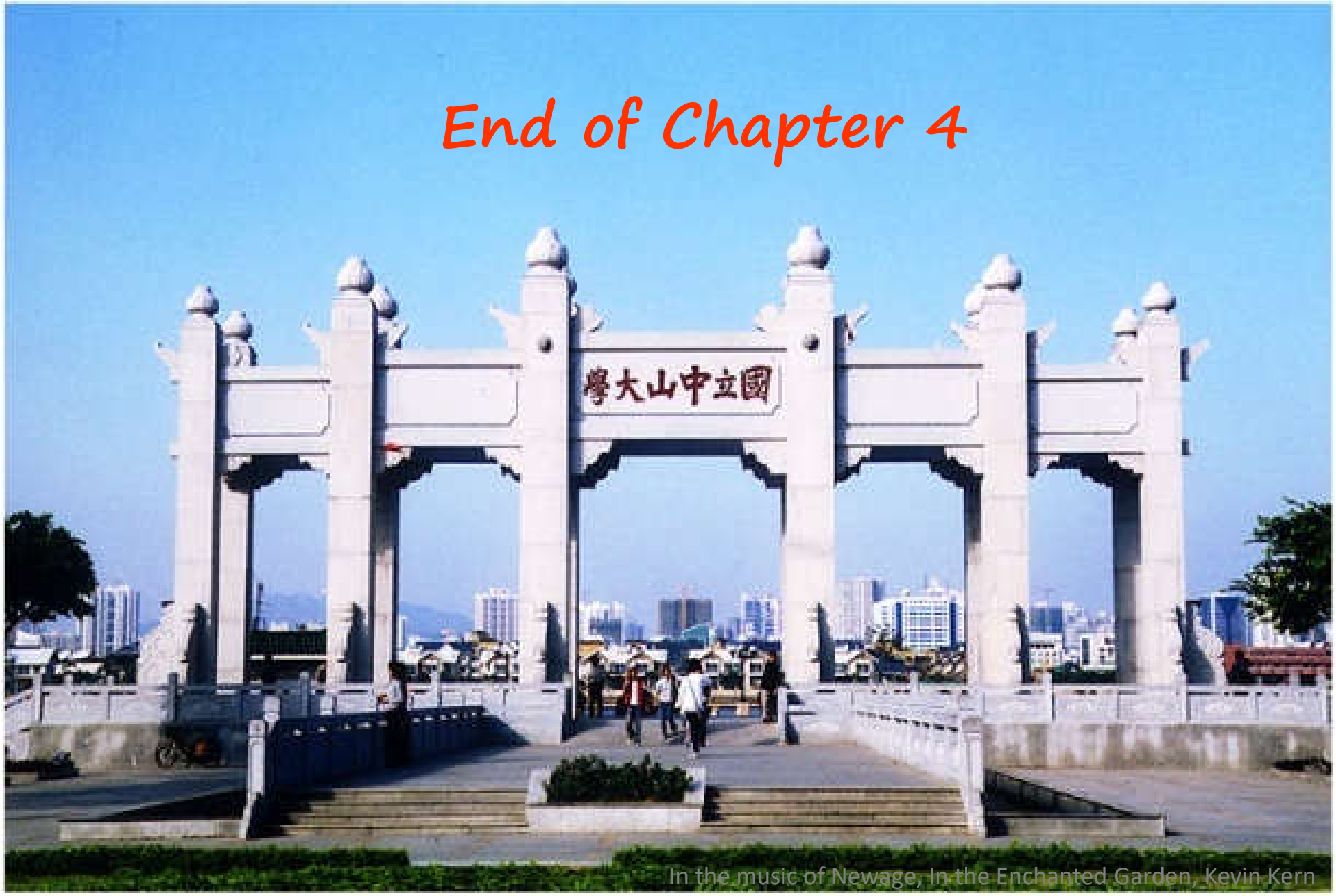
References

9. Adam Cain Eric Rescorla and Brian Korver: A Clustered SSL Accelerator. http://www.usenix.org/event/sec02/full_papers/rescorla/rescorla_html
10. Oracle. Overview of Secure Sockets Layer (ssl) in Oracle Application Server. http://download.oracle.com/docs/cd/b32110_01/core.1013/b32196/ssl_intro.htm http://www.hostyourspace.co.uk/category/addons
11. John. What is Secure Sockets Layer. http://hackhaholic.blogspot.com/2011/01/what-is-secure-sockets-layer-ssl.html
12. Romeoat. Wireshark Lab: SSL. http://blog.it.kmitl.ac.th/it51066527/10ssl/
13. IBM. The SSL Handshake. http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/index.jsp?topic=/com.ibm.itame2.doc_5.1/ss7aumst18.htm
14. Vimalanandan. Osi Security Architecture. http://www.classle.net/bookpage/osi-security-architecture

References

15. Jacob Appelbaum, Arjen Lenstra, David Molnar, Dag Arne, Osvik Benne de Weger, Alexander Sotirov, Marc Stevens. Md5 Considered Harmful Today - Creating a Rogue CA Certificate. http://www.win.tue.nl/hashclash/rogue-ca/
16. TCG. Trusted Computing Group - Developers - Trusted Network Connect. http://www.trustedcomputinggroup.org/developers/trusted_network_connect
17. INTEROP LABS. What is TCG's Trusted Network Connect. http://www.interop.com/archive/pdfs/TCG.pdf

End of Chapter 4



In the music of Newage, In the Enchanted Garden, Kevin Kern