WK01-WK03 Contents

WK01: Security Challenges in Wireless Networks

Challenges in Wireless Network Security

Wireless Networks are Unique

- Broadcast Media/Channel: easy to eavesdrop on, jam, overuse
- Mobile Users: can't be stopped via traditional firewall
  - Can roam across networks.

Standard Security requirements remains the same:

- authentication, confidentiality, integrity, availability

Large number of Mobile devices:

- Limited resources
- Can be lost/stolen: Lack of physical protection

Location privacy:

- Good if you wish rescue in emergency
- However, easy to be tracked !

What Is This Course About?

Threats, vulnerabilities, and security countermeasures of existing and upcoming wireless networks.

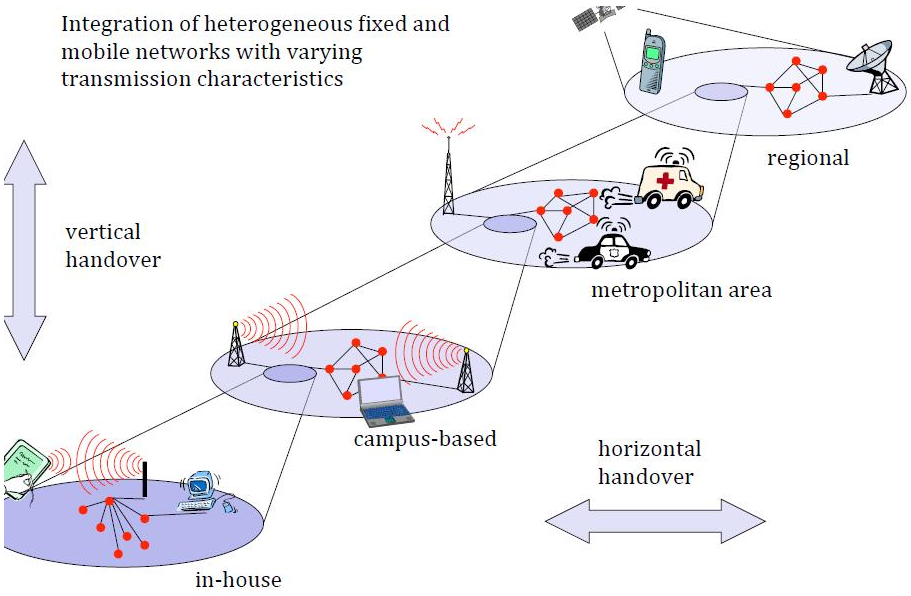
Security topics include a wide range of mainstream wireless technologies, including WLAN, VANETS and WSN, IoT.

- Mobile/Cellular Security a big area, we may have a quick glimpse .

We will be taking a systems approach.

Study crypto primitives and protocols applicable to wireless networks.

Learn practice of security in wireless networks



Integration:集成；heterogeneous:混杂的

Security Services – Wireless Network

- **Authentication（证明）**

– The most fundamental security service which ensures, that an entity has in fact the identity it claims to have.

- **Integrity（完整性）**

– In some kind, the “small brother” of the authentication service, as it ensures, that data created by specific entities may not be modified without detection(侦测).

- **Confidentiality（机密性）**

– The most popular security service, ensuring the secrecy of protected data.

- **Access Control（访问控制）**

– Controls that each identity accesses only those services and information it is entitled(有权利) to.

- **Non Repudiation(否认，拒绝)**

– Protects against that entities participating in a communication exchange can later falsely deny that the exchange occurred.(防止参与通信交换的实体随后错误地否认发生了交换.)

Wireless networks face all threats that does its wired counterpart:

– Masquerade, eavesdropping, authorization violation, loss or modification of transmitted information, repudiation of communication acts, forgery of information, sabotage(伪装、窃听、违反授权、传输信息丢失或修改、拒绝通信行为、伪造信息、破坏)

– Thus, similar measures like in fixed networks have to be taken.

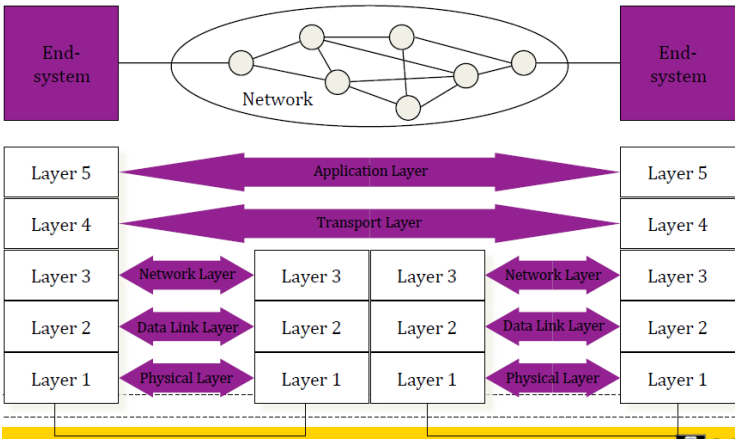
Different from wired network and wireless network

Wireless Network is more accessible for eavesdropping

• The lack of a physical connection makes it easier to access services

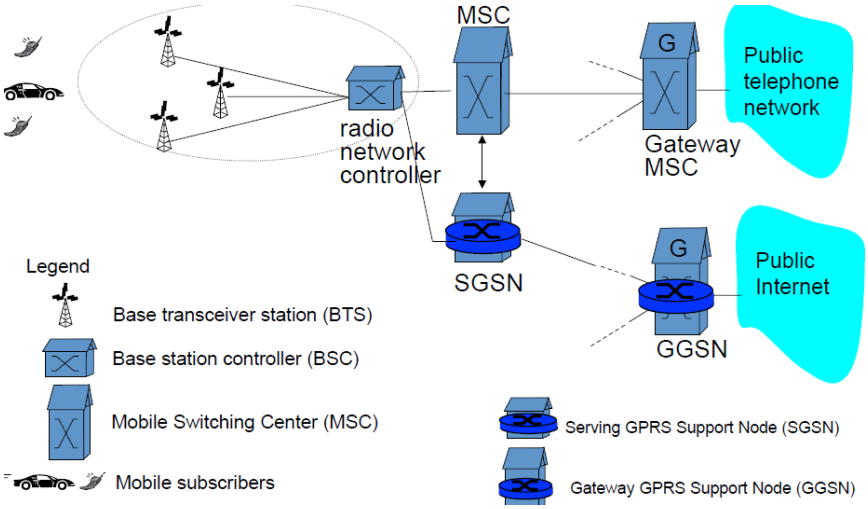
• Authentication has to be re-established when the mobile device moves
• Key management gets harder as peer identities cannot be pre-determined
• The location of a device / user becomes a more important information that is worthwhile to eavesdrop on and thus to protect
• Injecting(注入) bogus(假的) messages into the network is easy
• Replaying previously recorded messages is easy
• Illegitimate access to the network and its services is easy
• Denial of service is easily achieved by jamming

Communication in Layered Protocol Architectures



Types of Wireless Networks and Associated Security Challenges

Cellular network architecture

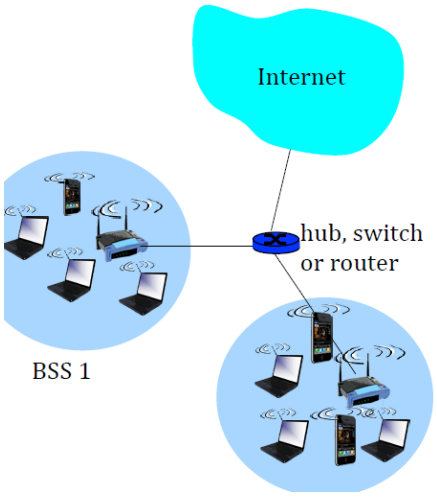


Cellular network security

• *2G had weak security*

– Possible attacks from a faked base station
– Cipher keys and authentication data transmitted in clear between and within networks
– Encryption not used in some networks -> open to fraud(骗子)
– Data integrity not provided
• *Some improvement with respect to 2nd generation*
– Cryptographic algorithms are published
– Integrity of the signalling messages is protected
• *Cellular Security not a focus but may explore a bit more*

Wifi - WLAN



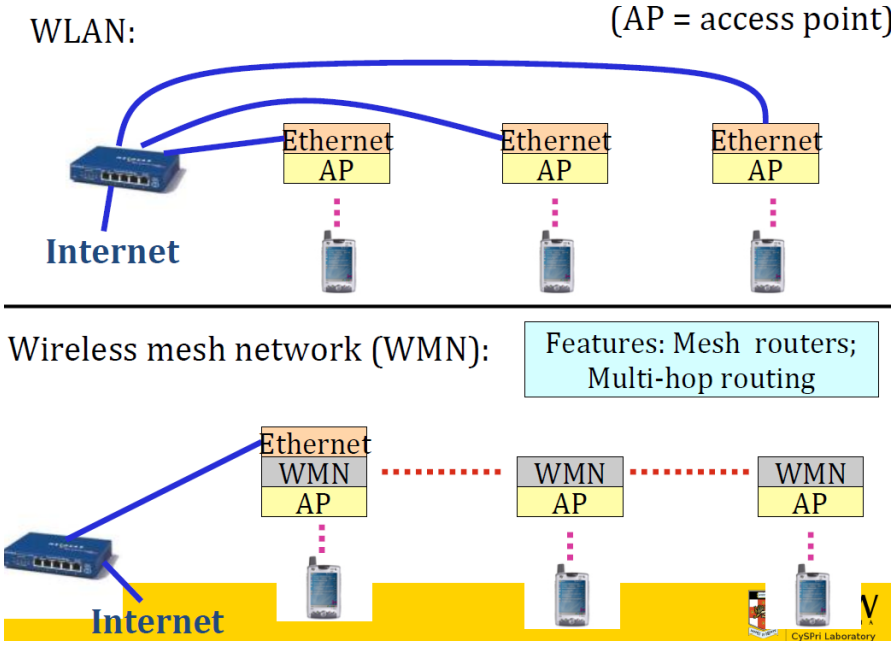
- *wireless host communicates with base station*
  - *base station = access point (AP)*
- *Basic Service Set (BSS) (aka "cell") in infrastructure mode contains:*
  - *wireless hosts*
  - *access point (AP): base station*
  - *ad hoc mode: hosts only*

ad hoc:特别地

Security in WLAN

– WEP, Why failed, what lesson did we learn
– 802.11i, Temporal Key Integrity Protocol (TKIP) and so on.

Wireless Mesh Networks: Extended WLAN coverage

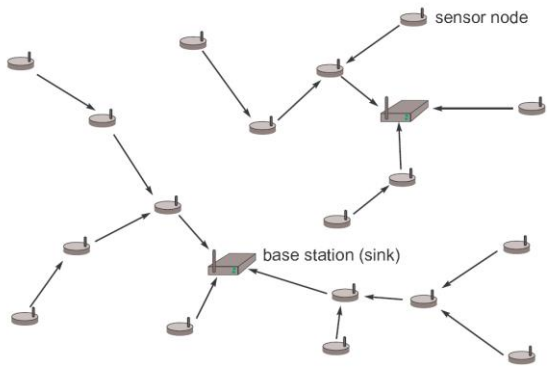


WMN Security

• Several verifications(查证) need to be performed:
– WAP (connected to internet) has to authenticate the user terminal.
– Each user has also to authenticate the next hop mesh router
– Each mesh router has to authenticate the other mesh routers in the WMN
– The data sent or received by user has to be protected (e.g., to ensure data integrity, non-repudiation and/or confidentiality).
– Denial of service attack possible
• Performing these verifications has to be efficient and lightweight(轻便), especially for the user terminal.

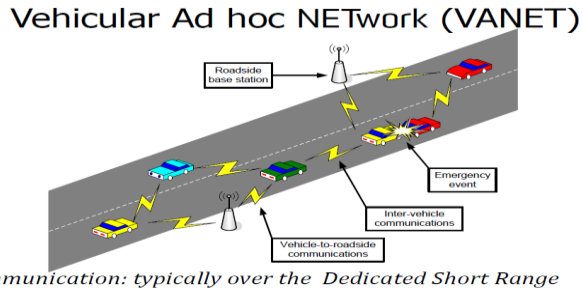
Sensor Networks

• Large number of sensor nodes, a few base stations
• Sensors are usually battery powered:
– Main design criteria: reduce the energy consumption
• Multi-hop communication reduces energy consumption:
– Overall energy consumption can be reduced, if packets are sent in several smaller hops instead of one long hop
– Fewer re-transmissions are needed due to collisions



Sensor Network Security

• Resource constraint
– Limited CPU processing power
– Limited Battery – attacker can deplete(耗尽).
– Need lightweight crypto protocols
• Physical Security
– Capture, Cloning, and Tampering(篡改) easy.
• Wireless Programming on Devices possible
– Additional security risk



Communication: typically over the Dedicated Short Range Communications (DSRC) (5.9 GHz)
Example of protocol: IEEE 802.11p

Why Security important?

• *Bogus Traffic Information*

• *Disruption(破坏) of road network/traffic movement*

• *Cheating with identity, speed, location*

• *Jamming(干扰)*

• *Location/privacy(隐私) issues*

• *Security requirements:*

– Sender authentication, Verification of data consistency, Availability, Non-repudiation, Privacy, Real-time constraints

802.15: Personal Area Network

less than 10 m diameter

replacement for cables (mouse, keyboard, headphones)

ad hoc: no infrastructure

master/slaves:

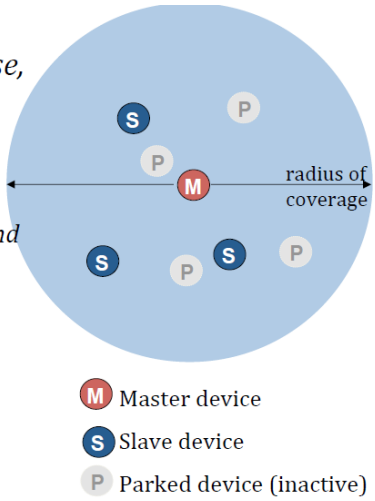
– *slaves request permission to send (to master)*

– *master grants requests*

802.15: evolved from Bluetooth specification

– *2.4-2.5 GHz radio band*

– *up to 721 kbps*



PAN Security

• *Short-range communications, master-slave principle*

• *Eavesdropping is difficult:*

– Frequency hopping

– Communication is over a few meters only

• *Security issues:*

– Authentication of the devices to each other

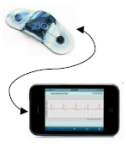
– Confidential channel

o *Based on secret link key*

IoT Devices and Security

- The market for wearable wireless sensors is projected to grow to more than 420 million devices by 2014.

- Fundamental applications in patient monitoring, personalized healthcare, telemedicine, and athlete training.



1. Apple iPhone SensorStrip

2. Nike + iPod Sports Kit

3. Nokia Sports Tracker

4. Toumaz Life Pebble

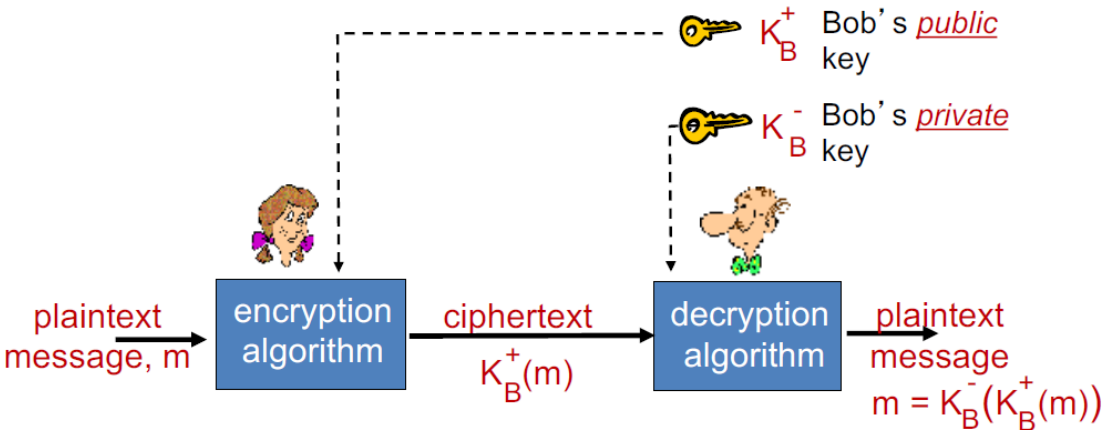
- Security is critical because these devices generate medical data, and challenging given that they have low power and computation capabilities.

WK01: Crypto Building Blocks

Overview

Learn Crypto building blocks for
Confidentiality and Authentication :Secret Keys
Integrity: Message Digests (Hash)
Integrity and Authentication: HMAC, MAC
Non-Repudiation: Digital Signature
Block Ciphers

Public Key Cryptography



Public key encryption algorithms

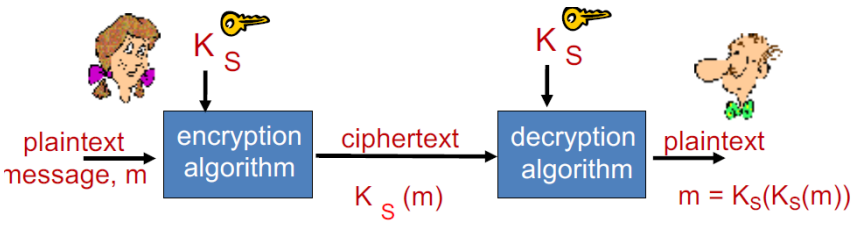
Need $K_B^+(.)$ and $K_B^-(.)$ such that

$$K_B^-(K_B^+(m)) = m$$

given public key, it should be impossible to compute private key

RSA: Rivest, Shamir, Adelson algorithm

Symmetric Key Cryptography



symmetric key crypto: Bob and Alice share same (symmetric) key: K_S

Session Keys

- Public-key cryptography is computationally intensive as compared to symmetric key crypto
- Exponentiation is computationally intensive
- Symmetric Algorithms much faster than RSA

Session key, K_S

- Bob and Alice use RSA to exchange a symmetric key K_S
- Once both have K_S, they use symmetric key cryptography

Confidentiality and Integrity

- Confidentiality: message private and secret
- Integrity: protection against message tempering
- Encryption alone may not guarantee integrity
  - Attacker can modify message under encryption without learning what it is
- Public Key Crypto Standards (PKCS)
  - "RSA encryption is intended primarily to provide confidentiality… It is not intended to provide integrity"
- Both Confidentiality and integrity are needed for security

Example on Integrity

• Software distribution protection:

– For a Good File and Hash (Good File), it is infeasible to find a Tampered File (containing rootkit or Trojan) such that Hash (Good File) = Hash (Tampered File)

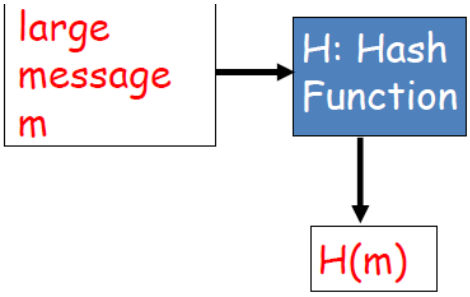
Hash/Message Digests (MD)(MD5)

Function H( ) that takes as input an arbitrary length message and outputs a fixed-length string: "message signature"

The values returned H( ) are called **hash values**, **hash codes**, **digests**, or simply **hashes**.

Note that H( ) is a many-to-1 function

H( ) is often called a "hash function"



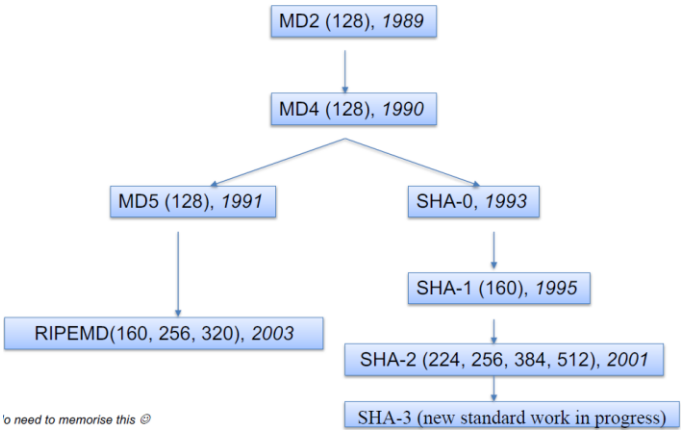
- Desirable properties:
  - Easy to calculate
  - Irreversibility: Can't determine m from H(m)
  - Collision resistance: Computationally difficult to produce m and m' such that H(m) = H(m')
  - Seemingly random output

MD Randomness

Message digest of a hashing should have a random pattern

- Randomness: any bit in digest is "1" half the time
- Change input only one bit, and the hash will change half of the digest bits
- Diffusion: if hash function does not exhibit the avalanche effect to a slight change of input, then it has poor randomization, and thus a cryptanalyst can make predictions about the input, being given only the output

Diffusion(扩散)，exhibit(展示)，avalanche:雪崩



Hash 函数之 MD5 算法介绍

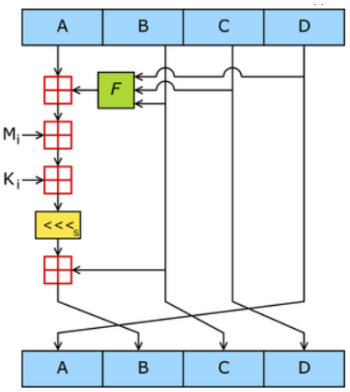
MD5：MD5 是输入不定长度信息，输出固定长度 128-bits 的算法。经过程序流程，生成四个 32 位数据，最后联合起来成为一个 128-bits 散列。基本方式为，求余、取余、调整长度、与链接变量进行循环运算。得出结果。即“Message-Digest Algorithm 5（信息-摘要算法）”，从名字来看就知道它是从 MD3、MD4 发展而来的一种加密算法，其主要通过采集文件的信息摘要，以此进行计算并加密。通过 MD5 算法进行加密，文件就可以获得一个唯一的 MD5 值，这个值是独一无二的，就像我们的指纹一样，因此我们就可以通过文件的 MD5 值来确定文件是否正确，密码进行加密后也会生成 MD5 值，论坛就是通过 MD5 值来验证用户的密码是否正确的。

MD5 算法实现

1、填充编码。

在 MD5 算法中，首先需要对信息进行填充，使其位长对 512 求余的结果等于 448。因此，信息的位长（Bits Length）将被扩展至 N*512+448，N 为一个非负整数，N 可以是零。填充的方法如下，在信息的后面填充一个 1 和无数个 0，直到满足上面的条件时才停止用 0 对信息的填充。然后，在这个结果后面附加一个以 64 位二进制表示的填充前信息长度。经过这两步的处理，现在的信息的位长=N*512+448+64=(N+1)*512，即长度恰好是 512 的整数倍。这样做的原因是为满足后面处理中对信息长度的要求。然后把消息分以 512 位为一分组进行处理，每一个分组进行 4 轮变换，以规定的 4 个常数为起始变量进行计算，重新输出 4 个变量，以这 4 个变量再进行下一分组的运算，如果已经是最后一个分组，则这 4 个变量为最后的结果，即 MD5 值。即是四轮循环运算：循环的次数是分组的个数（N+1）

F 一个非线性函数；一个函数运算一次。Mi 表示一个 32-bits 的输入数据，Ki 表示一个 32-bits 常数，用来完成每次不同的计算。



主循环有四轮（MD4 只有三轮），每轮循环都很相似。第一轮进行 16 次操作。每次操作对 a、b、c 和 d 中的其中三个作一次非线性函数运算，然后将所得结果加上第四个变量，文本的一个子分组和一个常数。再将所得结果向左环移一个不定的数，并加上 a、b、c 或 d 中之一。最后用该结果取代 a、b、c 或 d 中之一。以下是每次操作中用到的四个非线性函数（每轮一个）。

F(X, Y, Z) =(X&Y) | ((~X) & Z)

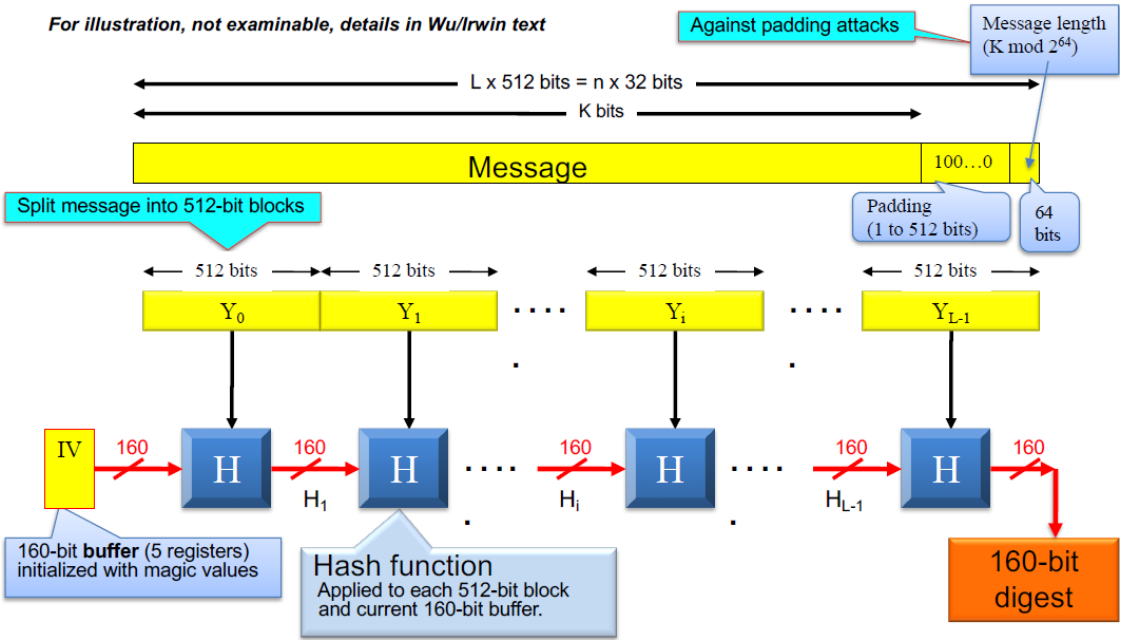
G(X, Y, Z) =(X&Z) | (Y & (~Z))

H(X, Y, Z) =X^Y^Z

I(X, Y, Z)=Y^(X|(~Z))

（&；是与，|是或，~是非，^是异或）

SHA-1



Algorithm Name	Max. Message Size (bits)	Block Size (bits)	Word size (bits)	Digest size (bits)	Collision resistance (bits)
SHA-1	2^{64}	512	32	160	80
SHA-256	2^{64}	512	32	256	128
SHA-384	2^{128}	1024	64	384	192
SHA-512	2^{128}	1024	64	512	256

https://blog.csdn.net/u010536615/article/details/80080918

SHA(Security Hash Algorithm)是美国的 NIST 和 NSA 设计的一种标准的 Hash 算法，SHA 用于数字签名的标准算法的 DSS 中，也是安全性很高的一种 Hash 算法，该算法的输入消息长度小于 2^64bit，最终输出的结果值是 160bit，SHA 与 MD-4 相比较而言，主要增加了扩展变换，将前一轮的输出也加到了下一轮，这样增加了雪崩效应，而且由于其 160bit 的输出（其实对于计算机来说，这个长度的输出不能算是很安全），对穷举攻击更具有抵抗性。

Sha-1 算法实现的基本步骤

1、将消息摘要转换成位字符串

因为在 Sha-1 算法中，它的输入必须为位，所以我们首先要将其转化为位字符串，我们以“abc”字符串来说明问题，因为'a'=97, 'b'=98, 'c'=99，所以将其转换为位串后为：01100001 01100010 01100011

2、对转换后的位字符串进行补位操作

Sha-1 算法标准规定，必须对消息摘要进行补位操作，即将输入的数据进行填充，使得数据长度对 512 求余的结果为 448，填充比特位的最高位补一个 1，其余的位补 0，如果在补位之前已经满足对 512 取模余数为 448，也要进行补位，在其后补一位 1 即可。总之，补位是至少补一位，最多补 512 位，我们依然以“abc”为例，其补位过程如下：

初始的信息摘要：01100001 01100010 01100011

第一步补位：　　01100001 01100010 01100011 1

....

补位最后一位：　01100001 01100010 01100011 10.......0(后面补了 423 个 0)

而后我们将补位操作后的信息摘要转换为十六进制，如下所示：

61626380 00000000 00000000 00000000

00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000

00000000 00000000

3、附加长度值

在信息摘要后面附加 64bit 的信息，用来表示原始信息摘要的长度，在这步操作之后，信息报文便是 512bit 的倍数。通常来说用一个 64 位的数据表示原始消息的长度，如果消息长度不大于 2^64，那么前 32bit 就为 0，在进行附加长度值操作后，其“abc”数据报文即变成如下形式：

61626380 00000000 00000000 00000000

00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000018

因为“abc”占 3 个字节，即 24 位 ，换算为十六进制即为 0x18。

4、初始化缓存

一个 160 位 MD 缓冲区用以保存中间和最终散列函数的结果。它可以表示为 5 个 32 位的寄存器(H0,H1,H2,H3,H4)。初始化为：

H0 = 0x67452301

H1 = 0xEFCDAB89

H2 = 0x98BADCFE

H3 = 0x10325476

H4 = 0xC3D2E1F0

如果大家对 MD-5 不陌生的话，会发现一个重要的现象，其前四个与 MD-5 一样，但不同之处为存储为 big-endien format.

5、计算消息摘要

在计算报文之前我们还要做一些基本的工作，就是在我们计算过程中要用到的方法，或定义。

(1)、循环左移操作符 Sn(x),x 是一个字，也就是 32bit 大小的变量，n 是一个整数且 0<=n<=32。Sn(X) = (X<<n)OR(X>>32-n)

(2)、在程序中所要用到的常量，这一系列常量字 k(0)、k(1)、...k(79)，将其以十六进制表示如下：

Kt = 0x5A827999　(0 <= t <= 19)

Kt = 0x6ED9EBA1 (20 <= t <= 39)

Kt = 0x8F1BBCDC (40 <= t <= 59)

Kt = 0xCA62C1D6 (60 <= t <= 79)

(3)、所要用到的一系列函数

Ft(b,c,d) ((b&c)|((~b)&d)) (0 <= t <= 19)

Ft(b,c,d) (b^c^d) (20 <= t <= 39)

Ft(b,c,d) ((b&c)|(b&d)|(c&d)) (40 <= t <= 59)

Ft(b,c,d) (b^c^d) (60 <= t <= 79)

(4)、计算

计算需要一个缓冲区，由 5 个 32 位的字组成，还需要一个 80 个 32 位字的缓冲区。第一个 5 个字的缓冲区被标识为 A，B，C，D，E。80 个字的缓冲区被标识为 W0, W1,..., W79

另外还需要一个一个字的 TEMP 缓冲区。

为了产生消息摘要，在第 4 部分中定义的 16 个字的数据块 M1, M2,..., Mn

会依次进行处理，处理每个数据块 Mi 包含 80 个步骤。

现在开始处理 M1, M2, ... , Mn。为了处理 Mi,需要进行下面的步骤

(1). 将 Mi 分成 16 个字 W0, W1, ... , W15, W0 是最左边的字

(2). 对于 t = 16 到 79 令 Wt = S1(Wt-3 XOR Wt-8 XOR Wt- 14 XOR Wt-16).

(3). 令 A = H0, B = H1, C = H2, D = H3, E = H4.

(4) 对于 t = 0 到 79，执行下面的循环

TEMP = S5(A) + ft(B,C,D) + E + Wt + Kt;

E = D; D = C; C = S30(B); B = A; A = TEMP;

(5). 令 H0 = H0 + A, H1 = H1 + B, H2 = H2 + C, H3 = H3 + D, H4 = H4 + E.

在处理完所有的 Mn, 后，消息摘要是一个 160 位的字符串，以下面的顺序标识

H0 H1 H2 H3 H4.

对于 SHA256,SHA384,SHA512。你也可以用相似的办法来计算消息摘要。对消息进行补位的算法完全是一样的。

Integrity vs Authentication

suppose Alice creates msg m, and calculates hash H(m) using a hash function (e.g. SHA-1)

Alice send the extended message (m, H(m)) to Bob

Bob upon receipt of this message independently calculates H(m') from the message

If H(m) = H(m'), message integrity verified.

What if Trudy intercepts the original message, replaces it with a new message n and its hash H(n) ?

HMAC

https://blog.csdn.net/chengqiuming/article/details/82822933

HMAC– Keyed-hash message authentication code: a message authentication code that uses a cryptographic key in conjunction with a hash function

o Runs data and authentication key through hash function twice

– Hashing is faster than encryption in software

– Allow the use of any hash function, e.g., SHA-256, or SHA-512

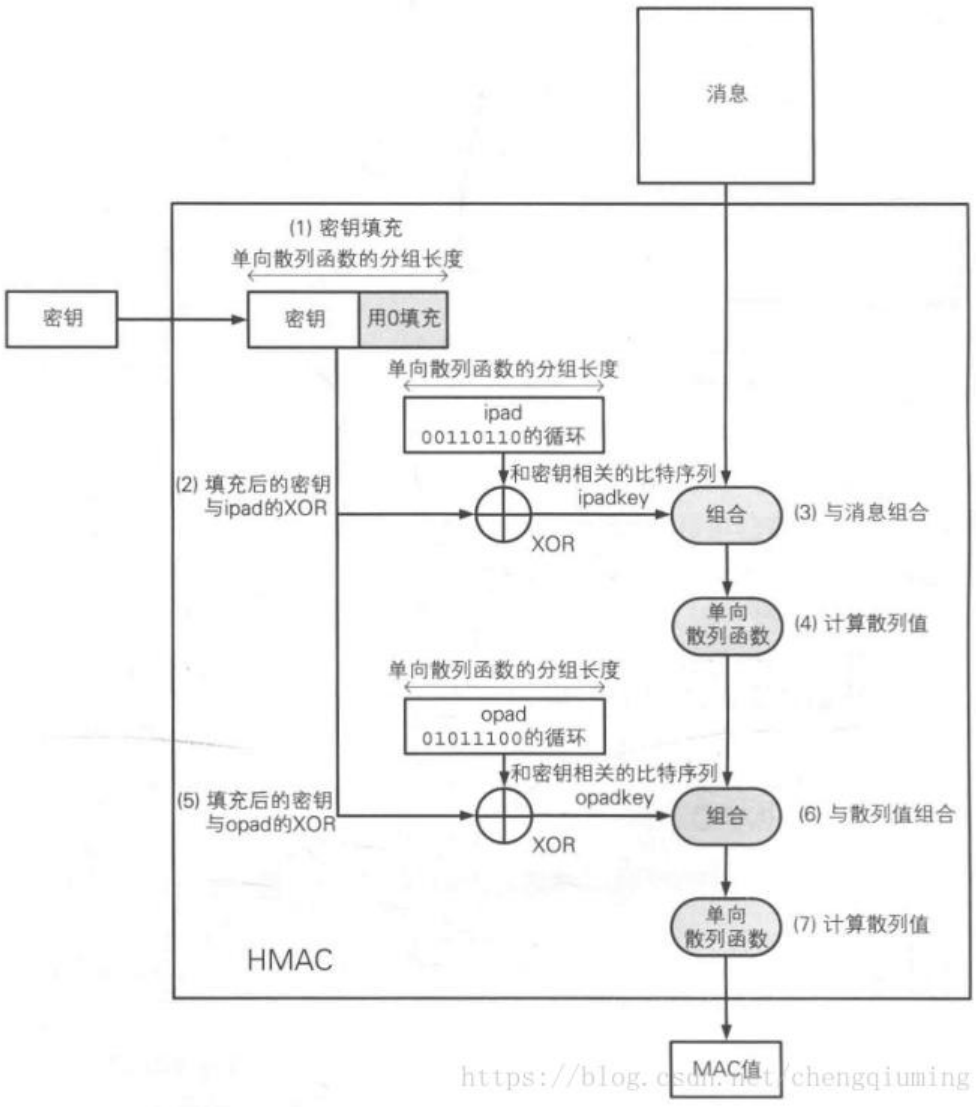
– No US export restrictions on HMAC and hash

• Used in TLS and IPSec (later weeks)

HMAC 是一种使用单向散列函数来构造消息认证码的方法，其中 HMAC 中的 H 就是 Hash 的意思。HMAC 中所使用的单向散列函数并不仅限于一种，任何高强度的单向散列函数都可以被用于 HMAC，如果将来设计出的新的单向散列函数，也同样可以使用。

使用 SHA-1、SHA-224、SHA-256、SHA-384、SHA-512 所构造的 HMAC，分别称为 HMAC-SHA1、HMAC-SHA-224、HMAC-SHA-384、HMAC-SHA-512。

图解及过程如下：



1　密钥填充

如果密钥比单向散列函数分组长度要短，就需要在末尾填充 0，直到其长度达到单向散列函数的分组长度为止。

如果密钥比分组长度要长，则要用单向散列函数求出密钥的散列值，然后将这个散列值用作 HMAC 的密钥。

2　填充后的密钥与 ipad 的 XOR

将填充后的密钥与被称为 ipad 的比特序列进行 XOR 运算。ipad 是将 00110110 这一比特序列不断循环反复直到达到分组长度所形成的比特序列，其中 ipad 的 i 是 inner 的意思。

XOR 运算所得到的值，就是一个和单向散列函数的分组长度相同，且和密钥相关的比特序列。这里将这个比特序列称为 ipadkey。

3　与消息组合

随后，将 ipadkey 与消息组合，也就是将和密钥相关的比特序列（ipadkey）附加在消息的开头。

4　计算散列值

将 3 的结果输入单向散列函数，并计算出散列值。

5　填充后的密钥与 opad 的 XOR

将填充后的密钥与被称为 opad 的比特序列进行 XOR 运算，opad 是将 01011100 这一比特序列不断循环反复直到达到分组长度所形成的比特序列，其中 opad 的 o 是 outer 的意思。

XOR 运算所得到的结果也是一个和单向散列函数的分组长度相同，且和密钥相关的比特序列。这里将这个比特序列称为 opadkey。

6　与散列值组合

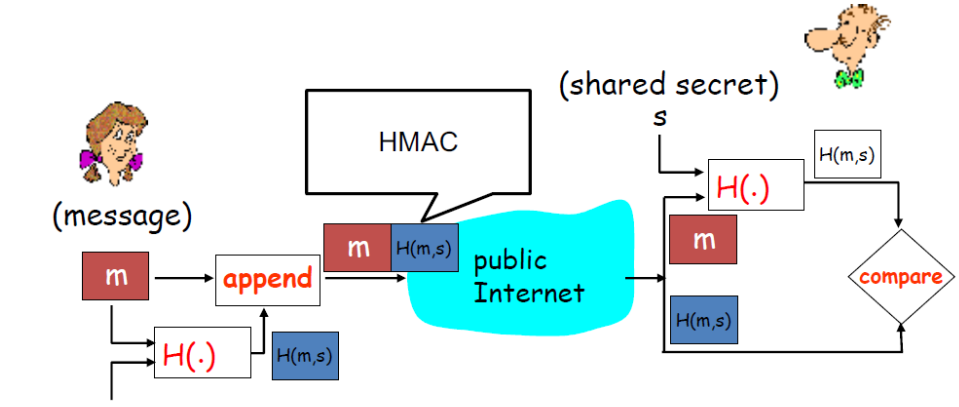
将 4 的散列值拼在 opadkey 后面。

7　计算散列值

将 6 的结果输入单向散列函数，并计算出散列值，这个散列值就是最终的 MAC 值。

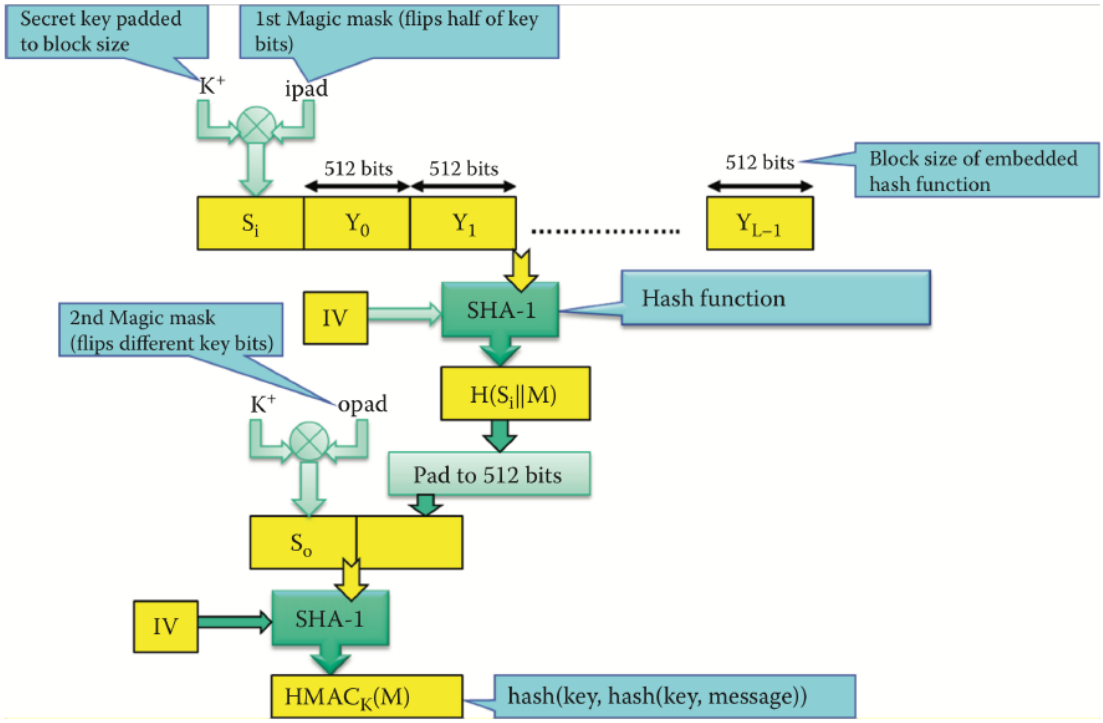
通过上述流程可以看出，最后得到的 MAC 值，一定是一个和输入的消息以及密钥都相关的长度固定的比特序列。

HMAC: Integrity and Authentication



- ***Authenticates sender***
- ***Verifies message integrity***
- No encryption ! – Confidentiality not always needed
- Also called “keyed hash”
- Please note that HMAC take both message and key as input, not a simple concatenation.

HMAC 工作原理



rc: Irwin Wu: figure 20.6

Hash, MAC and HMAC

• One way Hash Function(单向散列函数): Doesn't take any secret key or its operation

– Easy to compute (both hardware and software solutions possible)

– Concatenation(串联) of a Secret Key and Message can be passed through a hash function without any need for encryption/decryption for efficiency.

• Message Authentication Code: MAC

– Can use functions like DES, last bits of the cipher-text can be used as code. However, encryption software is slow, hence may be avoided.

在密码学中，消息认证码（MAC）（有时称为标签）是用于认证消息的一小段信息 - 换句话说，用于确认消息来自所声明的发送者（其真实性）并且没有被改变了。MAC 值允许验证者（也拥有密钥）检测消息内容的任何变化，从而保护消息的数据完整性及其真实性。虽然 MAC 功能类似于加密哈希函数，但它们具有不同的安全性要求。为了被认为是安全的，MAC 功能必须抵抗选择明文攻击下的存在性伪造。这意味着即使攻击者可以访问拥有密钥并为攻击者选择的消息生成 MAC 的 oracle，攻击者也无法在不执行不可行数量的情况下猜测其他消息（不用于查询 oracle）的 MAC 计算

MAC 与数字签名不同，因为 MAC 值都是使用相同的密钥生成和验证的。这意味着在发起通信之前，消息的发送方和接收方必须就相同的密钥达成一致，如对称加密的情况。出于同样的原因，在网络范围的共享密钥的情况下，MAC 不提供由签名提供的不可否认性的属性：任何可以验证 MAC 的用户也能够为其他消息生成 MAC。相反，使用密钥对的私钥生成数字签名，该密钥对是公钥密码术。由于此私钥只能由其持有者访问，因此数字签名证明文档是由该持有者签署的。因此，数字签名确实提供了不可否认性。

• HMAC: Key is XORed with ipad (part of pad) and then concatenated with the message m. This mix is run through a hash function. The result is then concatenated with XOR of Key and opad (part of pad). Now, this whole mix is run through hash function one more time.

XOR

相同为 0，不同为 1

a	b	a⊕b
1	0	1
1	1	0
0	0	0
0	1	1

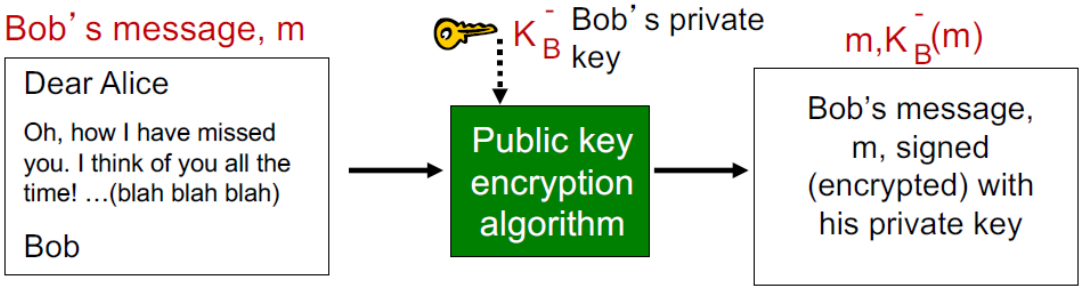
Digital signatures

cryptographic technique analogous to hand-written signatures:

- sender (Bob) digitally signs document, establishing he is document owner/creator.
- *verifiable, nonforgeable:* recipient (Alice) can prove to someone that Bob, and no one else (including Alice), must have signed document

simple digital signature for message m:

- Bob signs m by encrypting with his private key K_B^-, creating "signed" message, $K_B^-(m)$



- suppose Alice receives msg m, with signature: m, to $K_B^-(m)$
- Alice verifies m signed by Bob by applying Bob's public key K_B^+ to $K_B^-(m)$ then checks $K_B^+(K_B^-(m)) = m$.
- If $K_B^+(K_B^-(m)) = m$, whoever signed m must have used Bob's private key.

Alice thus verifies that:

- Bob signed m
- no one else signed m
- Bob signed m and not m'

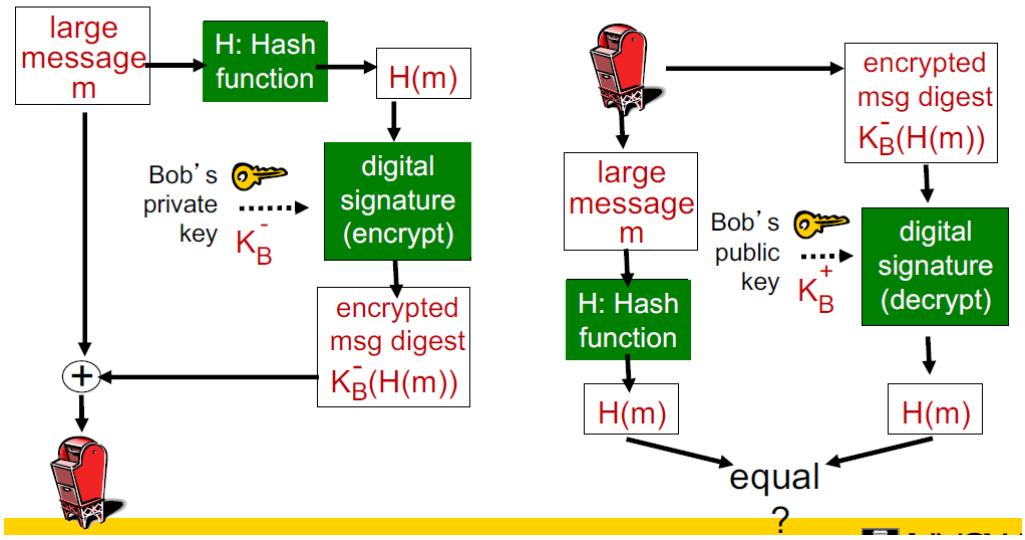
non-repudiation:

✓ Alice can take m, and signature $K_B^-(m)$ to court and prove that Bob signed m

Digital signature = signed message digest

Bob sends digitally signed message:

Alice verifies signature, integrity of digitally signed message:



Symmetric Key Cryptography (in detail)

Two types of symmetric ciphers

• Block ciphers

– Break plaintext message in equal-size blocks

– Encrypt each block as a unit

– Used in may Internet protocols (PGP-secure email, SSL (secure TCP), IPsec (secure net-transport layer)

• **Stream ciphers**

– encrypt one bit at time

– Used in secure WLAN

Block Cipher

Ciphertext processed as *k* bit blocks

1-to-1 mapping is used to map k-bit block of plaintext to k-bit block of ciphertext

E.g: k=3 (see table)

– 010110001111 => 101000111001

Possible permutations = 8! (40,320)

To prevent brute force attacks

– Choose large k (64, 128, etc)

Full-block ciphers not scalable

– E.g., for k = 64, a table with 2^{64} entries required

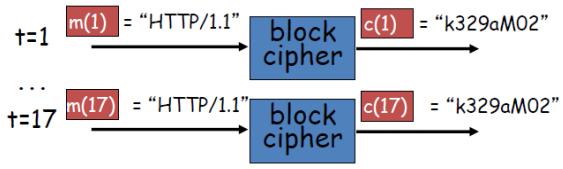
– instead use function that simulates a randomly permuted table

Input	Output
000	110
111	001
001	111
010	101
011	100
100	011
101	010
110	000

Permutation(置换)， scalable(可称量的)

Cipher Block Chaining (CBC)

cipher block: if input block repeated, will produce same cipher text:



Sender creates a random *k*-bit number *r(i)* for *i*th block and calculates

- $c(i) = K_S(m(i) \oplus r(i))$
- Sends ***c(1), r(1), c(2), r(2), c(3), r(3)***….
  - *r(i)* sent in clear but *Ks* not known to attackers.

CBC 模式的加密首先也是将明文分成固定长度（64 位）的块（P0，P1…），然后将前面一个加密块输出的密文与下一个要加密的明文块进行 XOR（异或）操作计算，将计算结果再用密钥进行加密得到密文。第一明文块加密的时候，因为前面没有加密的密文，所以需要一个初始化向量（IV）。跟 ECB 方式不一样，通过连接关系，使得密文跟明文不再是一一对应的关系，破解起来更困难，而且克服了只要简单调换密文块可能达到目的的攻击。但是该加密模式的缺点是不能实时解密，也就是说，必须等到每 8 个字节都接受到之后才能开始加密，否则就不能得到正确的结果。这在要求实时性比较高的时候就显得不合适了。

CBC Example:

- Example: sent text 010010010 if no CBC, sent txt = *101101101*
  - *The sent text is result of 1-to-1 mapping is used to map k-bit block of plaintext to k-bit block of ciphertext (see table earlier)*
- *Lets use the following random bits*
  - *r(001), r2(111), r3(100)*
- *First we XOR the plain text with the above random bits:*
  - *E.g 010 XOR 001 = 011*
  - *Now do table lookup for 011 -> 100*
- Use above technique to generate cipher text *c(1) = 100, c(2) = 010, c(3) = 000*: NOTE all three output different even though same plain text 010.
- Inefficient: twice as many bits sent

NOTE: Try other bits in the example to make sure you understand how this works.

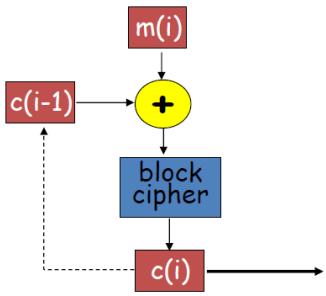
CBC Sender

- *cipher block chaining:* XOR *i*th input block, *m(i)*, with previous block of cipher text, *c(i–1)*
  - *c(0)* is an Initialisation Vector transmitted to receiver in clear
  - First block:

$c(1) = K_S(m(i) \oplus c(0))$

  - *Subsequent blocks:*

$c(i) = K_S(m(i) \oplus c(i-1))$



CBC Receiver

- How to recover *m(i)*?
  - Decrypt with *Ks* to get $s(i) = Ks(c(i)) = m(i) \oplus c(i-1)$
  - Now the receiver knows *c(i–1)*, it can get $m(i) = s(i) \oplus c(i-1)$
  - **IV** sent only once
  - Intruder can't do much with **IV** since it doesn't have *Ks*
  - CBC has important consequence for designing secure network protocols

Block Cipher

• Block cipher needs to wait for one block before processing it

– Suitable for storage

• Operates on a single block of plaintext

– 64 bits for Data Encryption Standard (DES), 128 bits for Advanced Encryption Standard (AES)

§ AES much (6x) faster than DES

• Computationally infeasible(不可实行的) to break block cipher by brute-force by cracking the key

– Brute force decryption (try each key)

Symmetric key crypto: DES

DES: Data Encryption Standard (US encryption standard [NIST 1993])

- 56-bit symmetric key, 64-bit plaintext input
- Block cipher with cipher block chaining
- How secure is DES?
  - DES Challenge: 56-bit-key-encrypted phrase decrypted (brute force) in less than a day
    - DES Challenge III: Distributed. Net worked with EFF's supercomputer and a world-wide network of 100,000 PCs to crack it within 22 hours and 15 minutes, Testing 2.45 billion keys per second
  - no known good analytic attack
- Making DES more secure:
  - 3DES: encrypt 3 times with 3 different keys

Symmetric key crypto: DES

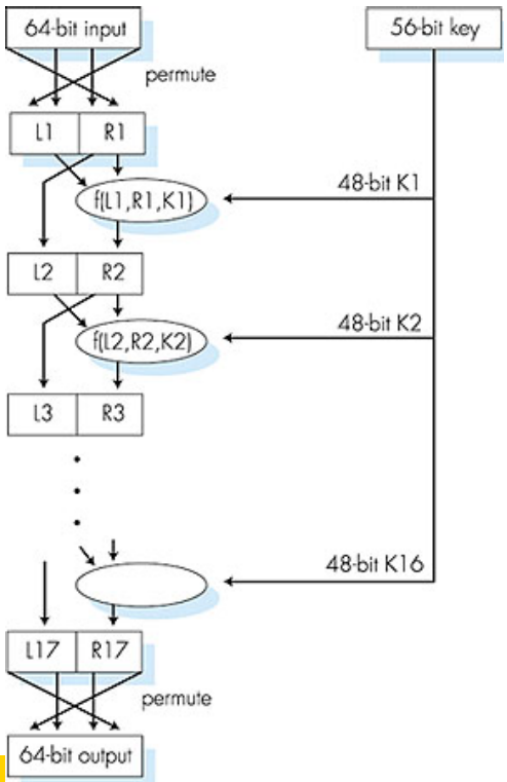
DES operation

initial permutation

16 identical "rounds" of function application, each using different 48 bits of key

final permutation

No need to memorise this



How does it work?

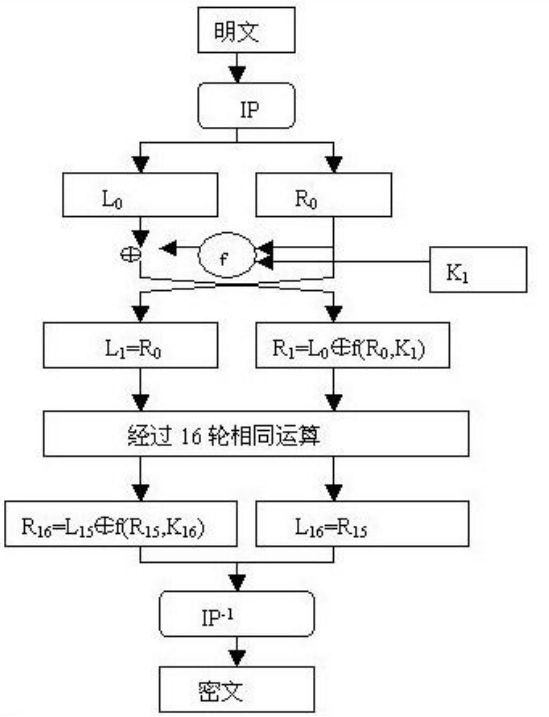
DES 算法为密码体制中的对称密码体制，又被称为美国数据加密标准。DES 是一个分组加密算法，典型的 DES 以 64 位为分组对数据加密，加密和解密用的是同一个算法。

DES 还是一种分组加密算法，该算法每次处理固定长度的数据段，称之为分组。DES 分组的大小是 64 位，如果加密的数据长度不是 64 位的倍数，可以按照某种具体的规则来填充位。

从本质上来说，DES 的安全性依赖于虚假表象，从密码学的术语来讲就是依赖于“混乱和扩散”的原则。混乱的目的是为隐藏任何明文同密文、或者密钥之间的关系，而扩散的目的是使明文中的有效位和密钥一起组成尽可能多的密文。两者结合到一起就使得安全性变得相对较高。

DES 算法具体通过对明文进行一系列的排列和替换操作来将其加密。过程的关键就是从给定的初始密钥中得到 16 个子密钥的函数。要加密一组明文，每个子密钥按照顺序（1-16）以一系列的位操作施加于数据上，每个子密钥一次，一共重复 16 次。每一次迭代称之为一轮。要对密文进行解密可以采用同样的步骤，只是子密钥是按照逆向的顺序（16-1）对密文进行处理。

密钥长 64 位，密钥事实上是 56 位参与 DES 运算（第 8、16、24、32、40、48、56、64 位是校验位，使得每个密钥都有奇数个 1），分组后的明文组和 56 位的密钥按位替代或交换的方法形成密文组。DES 算法的主要流程如下图所示，本文按照流程依次介绍每个模块。



IP 置换

IP 置换目的是将输入的 64 位数据块按位重新组合，并把输出分为 L0、R0 两部分，每部分各长 32 位。置换规则如下表所示：

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

表中的数字代表新数据中此位置的数据在原数据中的位置，即原数据块的第 58 位放到新数据的第 1 位，第 50 位放到第 2 位，……依此类推，第 7 位放到第 64 位。置换后的数据分为 L0 和 R0 两部分，L0 为新数据的左 32 位，R0 为新数据的右 32 位。

密钥置换

不考虑每个字节的第 8 位，DES 的密钥由 64 位减至 56 位，每个字节的第 8 位作为奇偶校验位。产生的 56 位密钥由下表生成（注意表中没有 8,16,24，32,40,48,56 和 64 这 8 位）：

57	49	41	33	25	17	9	1	58	50	42	34	26	18
10	2	59	51	43	35	27	19	11	3	60	52	44	36
63	55	47	39	31	23	15	7	62	54	46	38	30	22
14	6	61	53	45	37	29	21	13	5	28	20	12	4

在 DES 的每一轮中，从 56 位密钥产生出不同的 48 位子密钥，确定这些子密钥的方式如下：

1).将 56 位的密钥分成两部分，每部分 28 位。

2).根据轮数，这两部分分别循环左移 1 位或 2 位。每轮移动的位数如下表：

轮数	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
位数	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

移动后，从 56 位中选出 48 位。这个过程中，既置换了每位的顺序，又选择了子密钥，因此称为压缩置换。压缩置换规则如下表（注意表中没有 9，18，22，25，35，38，43 和 54 这 8 位）：

14	17	11	24	1	5	3	28	15	6	21	10
23	19	12	4	26	8	16	7	27	20	13	2
41	52	31	37	47	55	30	40	51	45	33	48
44	49	39	56	34	53	46	42	50	36	29	32

E 扩展置换

扩展置置换目标是 IP 置换后获得的右半部分 R0，将 32 位输入扩展为 48 位(分为 4 位×8 组)输出。

扩展置换目的有两个：生成与密钥相同长度的数据以进行异或运算；提供更长的结果，在后续的替代运算中可以进行压缩。

之后只写置换的目的，不写具体过程

S 盒代替

压缩后的密钥与扩展分组异或以后得到 48 位的数据，将这个数据送人 S 盒，进行替代运算。替代由 8 个不同的 S 盒完成，每个 S 盒有 6 位输入 4 位输出。48 位输入分为 8 个 6 位的分组，一个分组对应一个 S 盒，对应的 S 盒对各组进行代替操作。

P 盒置换

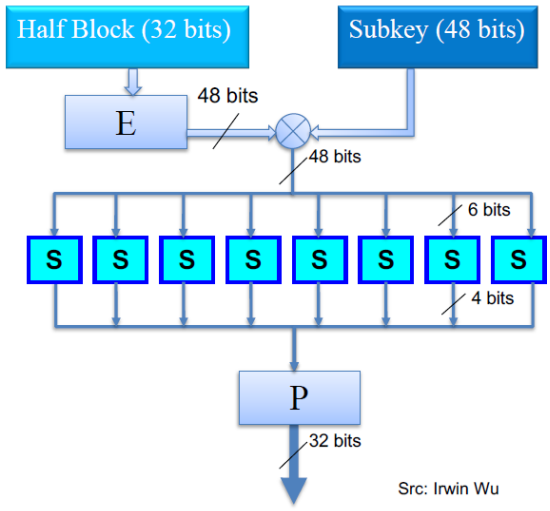
S 盒代替运算的 32 位输出按照 P 盒进行置换。该置换把输入的每位映射到输出位，任何一位不能被映射两次，也不能被略去。

IP^{-1} 末置换

末置换是初始置换的逆过程，DES 最后一轮后，左、右两半部分并未进行交换，而是两部分合并形成一个分组作为末置换的输入。

Feistel (F) function

- Expansion (E): the 32-bit half-block is expanded to 48 bits using the expansion permutation
- Key mixing:
  - Sixteen 48-bit subkeys: one for each round from 56 bit key
  - Derived from the main key using the key schedule
  - E is combined with a subkey using an XOR operation
- P: permutation function
  - P yields a 32-bit output from a 32-bit input by permuting the bits of the input block
- S-box: Substitution
  - Transforms input bits using substitution tables to provide diffusion
  - Spread plaintext bits throughout ciphertext
  - Small change in either the key or the plaintext should cause a drastic change in the ciphertext (avalanche effect)



No need to memorise this ☺

S-box(S 盒代替)

o 使用替换表转换输入位以提供扩散

o 在密文中传播明文位

o 密钥或明文的微小变化都会导致密文发生剧烈变化

（雪崩效应）

DES 的两种模式

数据补位

DES 数据加解密就是将数据按照 8 个字节一段进行 DES 加密或解密得到一段 8 个字节的密文或者明文，最后一段不足 8 个字节，按照需求补足 8 个字节（通常补 00（十六进制：\x00）或者 FF，根据实际要求不同）进行计算，之后按照顺序将计算所得的数据连在一起即可。这里有个问题就是为什么要进行数据补位？主要原因是 DES 算法加解密时要求数据必须为 8 个字节。

1. ECB 模式

DES ECB（电子密本方式）其实非常简单，就是将数据按照 8 个字节一段进行 DES 加密或解密得到一段 8 个字节的密文或者明文，最后一段不足 8 个字节，按照需求补足 8 个字节进行计算，之后按照顺序将计算所得的数据连在一起即可，各段数据之间互不影响。

2. CBC 模式

DES CBC（密文分组链接方式）有点麻烦，它的实现机制使加密的各段数据之间有了联系。其实现的机理如下：

1）首先将数据按照 8 个字节一组进行分组得到 D1D2.....Dn（若数据不是 8 的整数倍，用指定的 PADDING 数据补位）

2）第一组数据 D1 与初始化向量 I 异或后的结果进行 DES 加密得到第一组密文 C1（初始化向量 I 为全零）

3）第二组数据 D2 与第一组的加密结果 C1 异或以后的结果进行 DES 加密，得到第二组密文 C2

4）之后的数据以此类推，得到 Cn

5）按顺序连为 C1C2C3.....Cn 即为加密结果。

解密是加密的逆过程，步骤如下：

1）首先将数据按照 8 个字节一组进行分组得到 C1C2C3.....Cn

2）将第一组数据进行解密后与初始化向量 I 进行异或得到第一组明文 D1（注意：一定是先解密再异或）

3）将第二组数据 C2 进行解密后与第一组密文数据进行异或得到第二组数据 D2

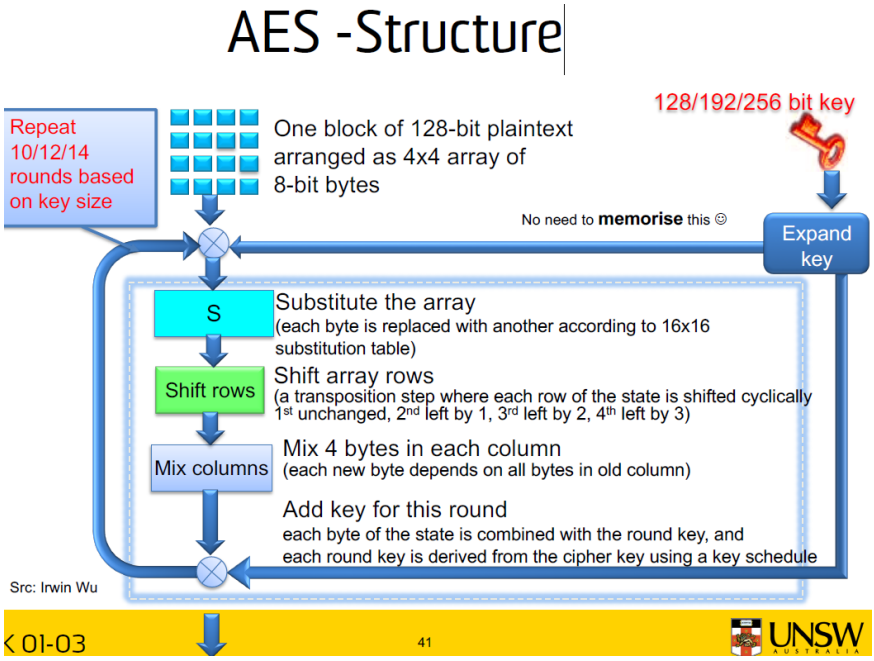
4）之后依此类推，得到 Dn

5）按顺序连为 D1D2D3.....Dn 即为解密结果。

这里注意一点，解密的结果并不一定是我们原来的加密数据，可能还含有你补得位，一定要把补位去掉才是你的原来的数据。

AES: Advanced Encryption Standard

- Symmetric-key NIST standard, replaced DES (Nov 2001)
- processes data in 128 bit blocks
- 128, 192, or 256 bit keys
- Brute force decryption (try each key) taking few secs on DES, takes 149 trillion years for AES
  - Universe lifetime: 100 billion years



AES Confidentiality Modes

- Five confidentiality modes of operation for symmetric key block cipher algorithms, such as AES
- Electronic Codebook (ECB),
  - Split plaintext into blocks, encrypt each one separately using the block cipher
- Cipher Block Chaining (CBC) mode
  - Split plaintext into blocks, XOR each block with the result of encrypting previous blocks
- Cipher Feedback (CFB), Output Feedback (OFB), and Counter (CTR) modes: stream ciphers based on block cipher (will discuss as needed in future lectures)

AES 加密过程

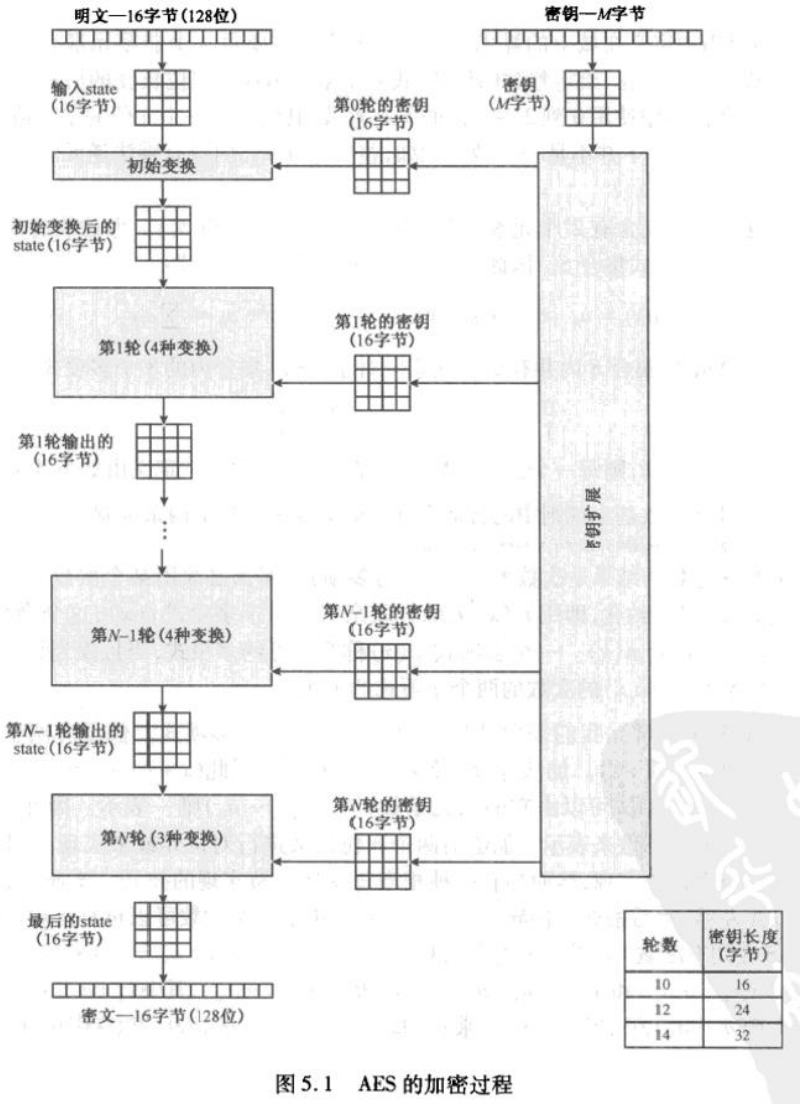


图 5.1　AES 的加密过程

加密解密算法的输入是一个 128 位分组。这些分组被描述成 4×4 的字节方阵，这个分组被复制到 state 数组中，并在加密和

解密的每一阶段都被修改。在字节方阵中，每一格都是一个字，包含了 4 字节。在矩阵中字是按列排序的。加密由 N 轮构成，轮数依赖于密钥长度：16 字节密钥对应 10 轮，24 字节密钥对应 12 轮，32 字节对应 14 轮。

AES 未使用 Feistel 结构。其前 N-1 轮由 4 个不同的变换组成：字节代替、行移位、列混淆和轮密钥加。最后一轮仅包含三个变换。而在第一轮前面有一个起始的单变换（轮密钥加），可以视为 0 轮。

字节代替（SubBytes）：用一个 S 盒完成分组的字节到字节的代替。

行移位（ShiftRows）：一个简单的置换。

列混淆（MixColumns）：利用域 GF(28)上的算术特性的一个代替。

轮密钥加（AddRoundKey）：当前分组和扩展密钥的一部分进行按位异或 XOR。

首尾使用轮密钥加的理由：若将其他不需要密钥的阶段放在首尾，在不知道密钥的情况下就能计算其逆，这就不能增加算法的安全性。

加密原理：轮密钥加实际是一种 Vernam 密码形式，其本身不难被破解。另外三个阶段一起提供了混淆、扩散和非线性功能。这三个阶段没有涉及密钥，就它们自身而言，并未提供算法的安全性。然而，该算法经历一个分组的 XOR 加密（轮密钥加），再对该分组混淆扩散（其他三个阶段），再接着又是 XOR 加密，如此交替进行，这种方式非常有效非常安全。

可逆原理：每个阶段均可逆。对字节代替、行移位和列混淆，在解密算法中用它们相对应的逆函数。轮密钥加的逆就是用同样的轮密钥和分组相异或，其原理就是 A⊕B⊕B = A。和大多数分组密码一样，AES 解密算法按逆序利用扩展密钥，然而其解密算法和加密算法并不一样，这是由 AES 的特定结构决定的。图 5.3 中加密和解密流程在纵向上是相反的，在每个水平点上，state 数组在加密和解密函数中都是一样的。

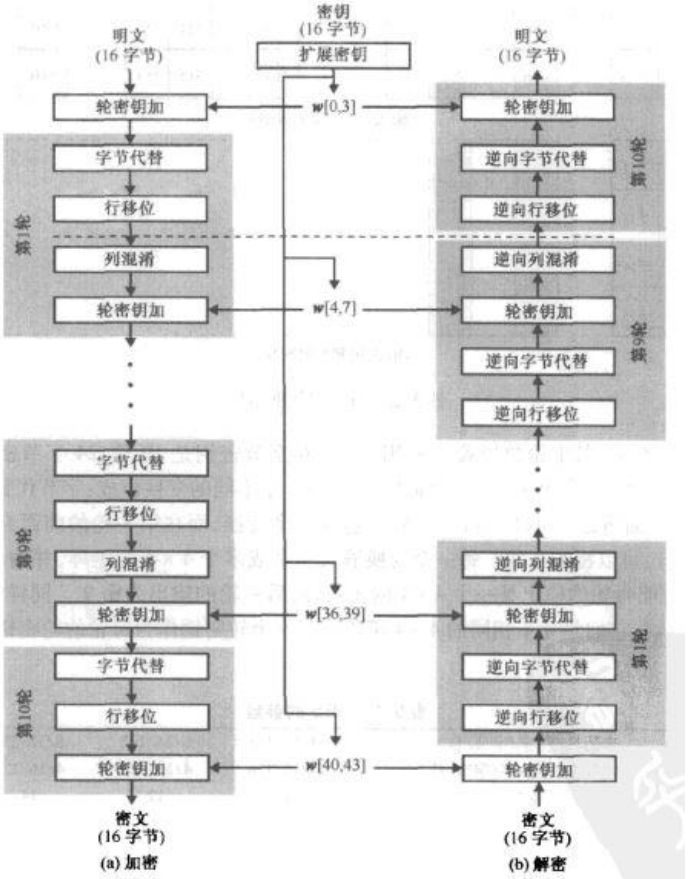
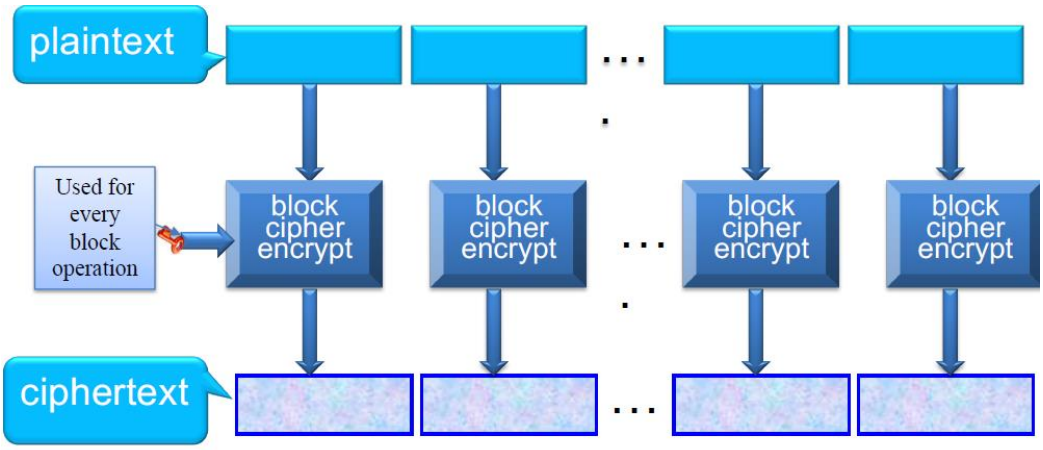


图 5.3　AES 加密和解密

ECB Mode Encryption



- Problem: Identical blocks of plaintext produce identical blocks of ciphertext
- No integrity checks: can mix and match blocks

Weakness of ECB

Message repetitions may show in ciphertext

Weakness is due to the encrypted message block uniquely mapped to the plaintext block

Main use is sending only a few blocks of short data

If longer messages, use Cipher Block Chaining (as discussed earlier)

DES 与 AES 的比较

自 DES 算法公诸于世以来，学术界围绕它的安全性等方面进行了研究并展开了激烈的争论。在技术上，对 DES 的批评主要集中在以下几个方面：

1、作为分组密码，DES 的加密单位仅有 64 位二进制，这对于数据传输来说太小，因为每个分组仅含 8 个字符，而且其中某些位还要用于奇偶校验或其他通讯开销。

2、DES 的密钥的位数太短，只有 56 比特，而且各次迭代中使用的密钥是递推产生的，这种相关必然降低密码体制的安全性，在现有技术下用穷举法寻找密钥已趋于可行。

3、DES 不能对抗差分和线性密码分析。

4、DES 用户实际使用的密钥长度为 56bit，理论上最大加密强度为 256。DES 算法要提高加密强度（例如增加密钥长度），则系统开销呈指数增长。除采用提高硬件功能和增加并行处理功能外，从算法本身和软件技术方面都无法提高 DES 算法的加密强度。

AES 的优势：

1、运算速度快，在有反馈模式、无反馈模式的软硬件中，Rijndael 都表现出非常好的性能。

2、对内存的需求非常低,适合于受限环境。

3、Rijndael 是一个分组迭代密码， 分组长度和密钥长度设计灵活。

4、AES 标准支持可变分组长度，分组长度可设定为 32 比特的任意倍数，最小值为 128 比特，最大值为 256 比特。

5、AES 的密钥长度比 DES 大， 它也可设定为 32 比特的任意倍数，最小值为 128 比特，最大值为 256 比特，所以用穷举法是不可能破解的。

7、AES 算法的设计策略是 WTS。WTS 是针对差分分析和线性分析提出的，可对抗差分密码分析和线性密码分析。

DES 与 RSA 的比较

RSA 算法的密钥很长，具有较好的安全性，但加密的计算量很大，加密速度较慢限制了其应用范围。为减少计算量，在传送信息时，常采用传统加密方法与公开密钥加密方法相结合的方式，即信息采用改进的 DES 对话密钥加密，然后使用 RSA 密钥加密对话密钥和信息摘要。对方收到信息后，用不同的密钥解密并可核对信息摘要。

采用 DES 与 RSA 相结合的应用，使它们的优缺点正好互补，即 DES 加密速度快，适合加密较长的报文，可用其加密明文；RSA 加密速度慢，安全性好，应用于 DES 密钥的加密，可解决 DES 密钥分配的问题。

目前这种 RSA 和 DES 结合的方法已成为 EMAIL 保密通信标准。

(RSA 在后面的章节有介绍具体的算法)

Lab 1 Questions (Compare all the algorithm)

§ Compare DES encryption and AES encryption. Explain your observations.

As you can see from the figure, AES is more efficient than DES, especially when the file size is large. This is probably because DES was originally designed to be implemented quickly with hardware, and then people designed AES that has efficiency on both software and hardware, and AES uses Rijndael algorithm which is more advanced. In addition, AES has a longer key length than DES, so AES performs better than DES in terms of efficiency and security.

从图中可以看出，AES 比 DES 更有效，尤其是当文件大小较大时。这可能是因为 DES 最初的设计是为了用硬件快速实现,然后人们设计了在软件和硬件上都具有效率的 AES,而 AES 使用了更高级的 Rijndael 算法。此外，AES 的密钥长度比 DES 长，因此在效率和安全性方面，AES 的性能优于 DES。

§ Compare DES encryption and RSA encryption. Explain your observations.

As can be seen from the graph, DES is more efficient than RSA in encryption efficiency. Because of the long key of RSA algorithm, the computation of encryption is very large, which results in the slow speed of encryption. However, its security is better than DES algorithm. In addition, DES is a Symmetric key cryptography algorithm, which means its encryption and decryption use the same key, but RSA is a Asymmetric key cryptography algorithm, which means that encryption and decryption use different keys.

从图中可以看出，DES 在加密效率上比 RSA 更有效。由于 RSA 算法密钥较长，加密计算量很大，导致加密速度较慢。但其安全性优于 DES 算法。另外，DES 是一种对称密钥加密算法，这意味着它的加密和解密使用相同的密钥，而 RSA 是一种非对称密钥加密算法，这意味着加密和解密使用不同的密钥。

§ Compare DES encryption and SHA-1 digest generation. Explain your observations.

SHA-1 and DES encryption have the similar efficiency. SHA-1 is slightly more efficient than DES, but SHA-1 is not an encryption algorithm, it is an algorithm used to confirm data integrity.

SHA-1 和加密具有相似的效率。SHA-1 比 DES 慢，但 SHA-1 不是加密算法，它是一种用于确认数据完整性的算法。

§ Compare HMAC signature generations and SHA-1 digest generation. Explain your observations.

SHA-1 and HMAC belong to hash algorithm, and SHA-1 has higher efficiency than HMAC. In fact, the built-in function used by HMAC in the experiment is MD5, so it is actually a comparison of efficiency between MD5 and SHA-1. Because the data digest length of SHA-1 algorithm is longer, MD5 is easier to collide than SHA-1. Finally, HMAC can also use hashing algorithms such as SHA-1, SHA-256 and so on.

sha-1 和 hmac 属于哈希算法，sha-1 的效率高于 hmac。实际上，HMAC 在实验中使用的内置函数是 MD5，所以它实际上是 MD5 和 SHA-1 之间效率的比较。由于 sha-1 算法的数据摘要长度较长，因此 MD5 比 sha-1 更容易碰撞。最后，HMAC 还可以使用散列算法，如 sha-1、sha-256 等。

§ Compare RSA encryption and decryption times. Can you explain your observations?

As can be seen from the figure, RSA encryption is faster than decryption. Because public key (e) can be chosen to encrypt manually, the encrypted ciphertext is a relatively small number, and to obtain plaintext from the ciphertext, the private key (d) is a larger number than the public key (e), so when calculating plaintext, a very large number will be obtained, which greatly reduces the efficiency of decryption.

从图中可以看出，RSA 加密比解密快。由于公钥（E）可以选择手工加密，加密后的密文是一个相对较小的数字，而从密文中获取明文时，私钥（D）的数目比公钥（E）的数目大，因此在计算明文时，会得到一个很大的数字，这大大降低了解密的效率。

WK02: Stream Ciphers and WLAN Security

Overview

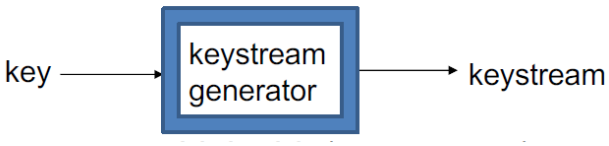
Stream Ciphers
How to design a flawed Security Protocol:
– WEP Case Study
Fixing a flawed Protocol: WPA, WPA2

Two types of symmetric ciphers:

① **Block ciphers：**
- Break plaintext message in equal-size blocks
- Encrypt each block as a unit
- Used in may Internet protocols (PGP-secure email, SSL (secure TCP), IPsec (secure net-transport layer)

② **Stream ciphers：**
- encrypt one bit at time
- Used in secure WLAN

Stream Ciphers



Process message bit by bit (as a stream)

- Ideal for real-time communication
- A keystream must not be reused; otherwise the encrypted messages can be recovered

How does it work?

combine each byte of keystream with byte of plaintext to get ciphertext:

- m(i) = ith unit of message
- ks(i) = ith unit of keystream
- c(i) = ith unit of ciphertext
- c(i) = ks(i) ⊕ m(i) (⊕ = exclusive or)
- m(i) = ks(i) ⊕ c(i)

Rivest Cipher 4 (RC4)

Rivest Cipher 4: Designed by Ron Rivest

- A proprietary cipher owned by RSA.com
- No longer a trade secret
- Ideal for software implementation, as it requires only byte manipulations

Variable key size (40 to 256 bits), byte-oriented stream cipher

Widely used

- SSL, Wireless WEP and WPA, Cellular Digital Packet Data, OpenBSD pseudo-random number generator

在密码学中，**RC4**（来自 Rivest Cipher 4 的缩写）是一种流加密算法，密钥长度可变。它加解密使用相同的密钥，因此也属于对称加密算法。RC4 是有线等效加密（WEP）中采用的加密算法

在介绍 RC4 算法原理之前。先看看算法中的几个关键变量:

1、密钥流：RC4 算法的关键是依据明文和密钥生成相应的密钥流，密钥流的长度和明文的长度是相应的。也就是说明文的长度是 500 字节，那么密钥流也是 500 字节。当然，加密生成的密文也是 500 字节。由于密文第 i 字节=明文第 i 字节^密钥流第 i 字节;

2、状态向量 S：长度为 256。S[0],S[1].....S[255]。每一个单元都是一个字节。算法执行的不论什么时候。S 都包含 0-255 的 8 比特数的排列组合，仅仅只是值的位置发生了变换；

3、暂时向量 T：长度也为 256，每一个单元也是一个字节。 假设密钥的长度是 256 字节。就直接把密钥的值赋给 T，否则，轮转地将密钥的每一个字节赋给 T。

4、密钥 K：长度为 1-256 字节。注意密钥的长度 keylen 与明文长度、密钥流的长度没有必定关系。通常密钥的长度趣味 16 字节（128 比特）。

RC4 的原理分为三步：

1、初始化 S 和 T

```
for i=0 to 255 do
    S[i]=i;
    T[i]=K[ imodkeylen ];
```

2、初始排列 S

```
j=0;
for i=0 to 255 do
    j= ( j+S[i]+T[i])mod256;
    swap(S[i],S[j]);
```

3、产生密钥流

```
i,j=0;
for r=0 to len do   //r 为明文长度，r 字节
    i=(i+1) mod 256;
    j=(j+S[i])mod 256;
    swap(S[i],S[j]);
    t=(S[i]+S[j])mod 256;
    k[r]=S[t];
```

WEP uses RC4 which is a stream cipher algorithm, meaning that for each bit of plaintext, it produces one bit of keystream and XOR them to produce the ciphertext. In the simplest operation of a stream cipher, the same key will always produce the same stream of random numbers. However, this is undesirable, because a known plaintext attack will be able to compute the keystream (c XOR d = k) and use it to decrypt new messages.

Thus, IV is introduced to solve this problem. IV in WEP is a 24-bit random value that changes periodically to prevent re-use of keystream. However, 24-bit length is too short and the IV will eventually be reused (which could be detected as IV is sent in cleartext). The reuse of the same IV will produce identical key streams.

Given that the attacker knows the IV, combined with a weakness in the RC4 key scheduling, an analytic attack could be done to recover the WEP key after intercepting a relatively small amount of traffic.

WEP 使用 RC4，这是一种流密码算法，也就是说，对于每一位明文，它生成一位密钥流，然后 XOR 生成密文。在流密码的最简单操作中，同一个密钥总是产生相同的随机数流。但是，这是不可取的，因为已知的明文攻击将能够计算密钥流（c xor d=k），并使用它来解密新消息。

因此，我们引入了 IV 来解决这个问题。在 WEP 中，IV 是一个 24 位随机值，它周期性地变化，以防止再次使用密钥流。但是，24 位长度太短，IV 最终将被重用（可以检测到，因为 IV 以明文形式发送）。重复使用相同的 IV 将产生相同的密钥流。

考虑到攻击者知道 IV，再加上 RC4 密钥调度中的一个弱点，可以进行分析攻击，在截获相对少量的流量后恢复 WEP 密钥。

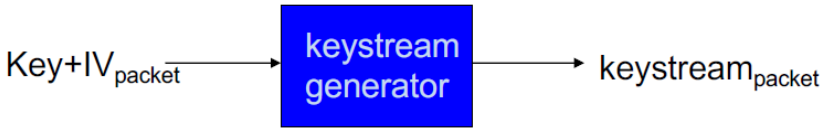
Wired Equivalent Privacy (WEP)

Provide security equivalent to Wired Network
- *Problem starts with this thinking!*

Symmetric key crypto
- confidentiality
- end host authorization
- data integrity

Efficient
- implementable in hardware or software

Stream cipher and packet independence:



Design goal: each packet separately encrypted

If for frame *n+1*, use keystream from where we left off for frame *n*, then each frame is not separately encrypted
- need to know where we left off for packet *n* (e.g Cipher Block chain approach)

WEP approach: initialize keystream with key + new IV for each packet:

WEP Pre-shared Key (WEP Key)

Enter a key (password) on access point and then then enter the key on all devices

- This is the pre-shared key, AKA WEP Key *(Shared Secret)*.

Not possible to authenticate individuals

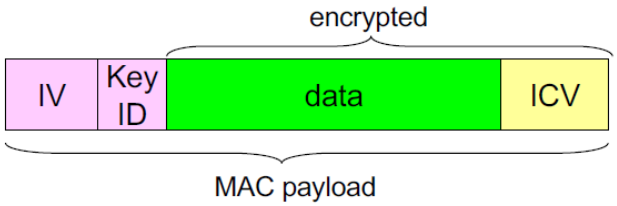
- hard to distinguish who is using service - needs extra work.

A key compromise for one user means that every device needs to change new key

- Must be distributed to all users securely

每个 AP 都会预先获得一个共享的对称 key（WEB Key），缺点就是如果有一个 AP 的密码被泄露，将会使得所有的 AP 密码都被泄露。并且没办法分辨是好人还是坏人在请求服务。

WEP encryption



sender calculates Integrity Check Value (ICV) over data

- four-bytes for data integrity: uses CRC-32

each side has 104-bit shared key(earlier only 40-bit)

sender creates 24-bit initialization vector (IV), appends to key: gives 128-bit key

sender also appends keyID (in 8-bit field)

128-bit key input into pseudo random number generator (PRNG) e.g. RC4 to get keystream

data in frame + ICV is encrypted with RC4:

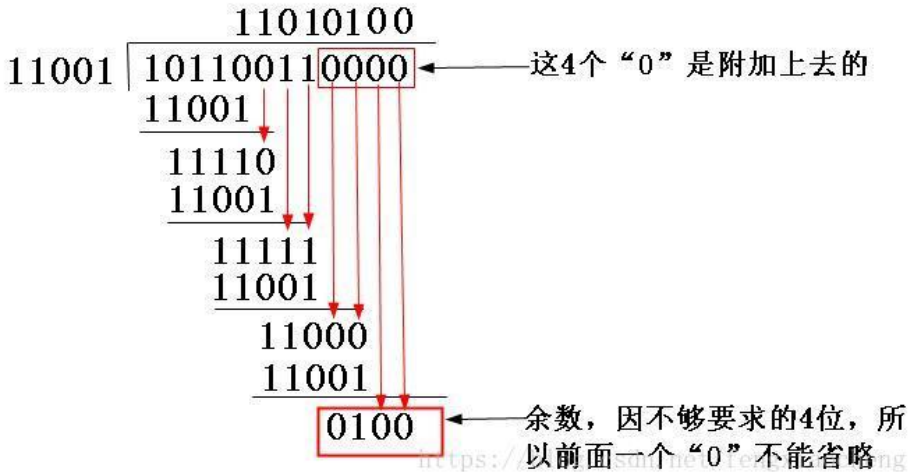
- Bytes of keystream are XORed with bytes of data & ICV
- IV & keyID are appended to encrypted data to create payload

IV：sender creates 24-bit initialization vector (IV), appends to key: gives 128-bit key

Key ID：sender also appends keyID (in 8-bit field)

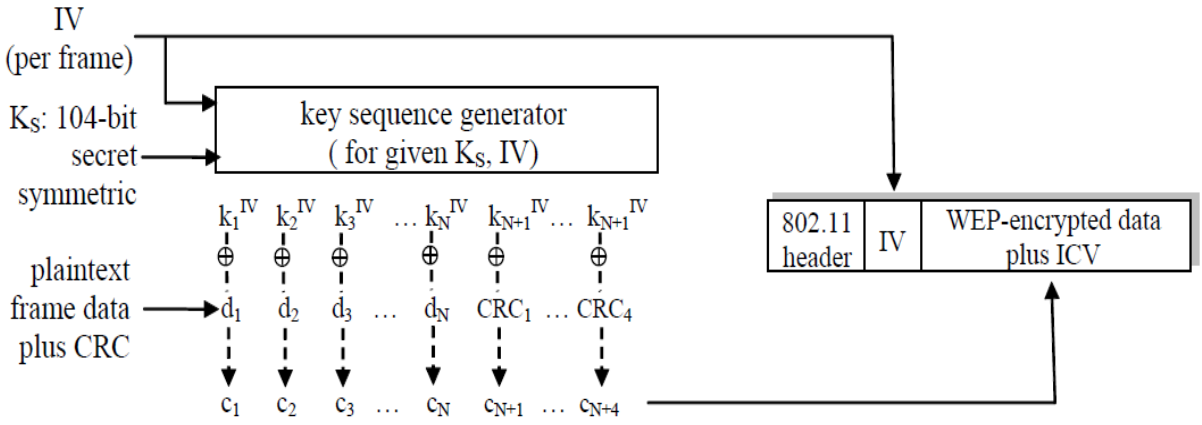
可能有两个设备中有很多密码，keyID用来暗示是用第几个密码加密，即是指示所使用的密码在设备database中的位置。

ICV 字段：CRC-32: 就是 CRC 校验码是 32 位（即生成多项式包含 x 的 32 次方）

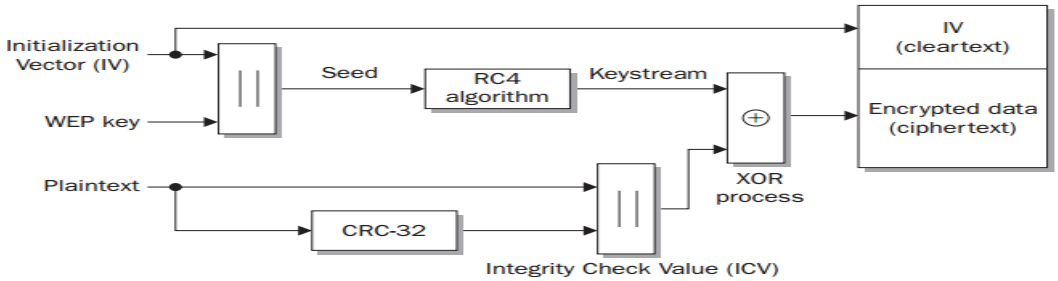


10110011 是数据，11001 是 CRC 校验码（它的生成多项式是：G（X） = X^4 + X^3 + 1），因为 CRC 校验码是五位，所以要在数据后面补齐四位再做除法（注意：这里的除法，减法不用借位!）所求余数就加入到原数据中去，即是 10110011 0100。

加密过程：



new IV for each frame



IV是动态生成的24bit随机数，标准没有指定应该怎么生成，而且在数据帧中以明文的方式进行发送，它和key结合生成随机种子（seed），然后运用CR4算法生成秘钥(keystream)

Example:

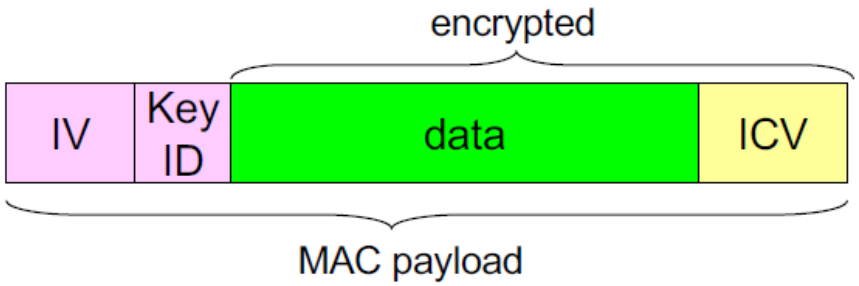
Consider the following pseudo-WEP protocol. The key is 4 bits and the IV is 2 bits. The IV is appended to the end of the key when generating the keystream. Suppose that the shared secret key is 1010. The keystreams for the four possible inputs are as follows:

101000: 00101011010101010011011010100100 . . .
101001: 10100110110010101110100100101101 . . .
101010: 00011010001111000101001001001111 . . .
101011: 11111010000000001010101000010111 . . .

Suppose all messages are 8-bits long. Suppose the ICV (integrity check) is 4-bits long, and is calculated by XOR-ing the first 4 bits of data with the last 4 bits of data. Suppose the pseudo-WEP packet consists of three fields: first the IV field, then the message field, and last the ICV field, with some of these fields encrypted.

- We want to send the message m = 10100000 using the IV = 11 and using WEP. What will be the values in the three WEP fields?

WEP decryption



- receiver extracts IV (*received in plaintext*)
- inputs IV, shared secret key into pseudo random generator, gets keystream
- XORs keystream with encrypted data to decrypt data + ICV
- verifies integrity of data with ICV
  - note: message integrity approach used here is CRC-32 different from MAC (message authentication code) and signatures (using PKI).

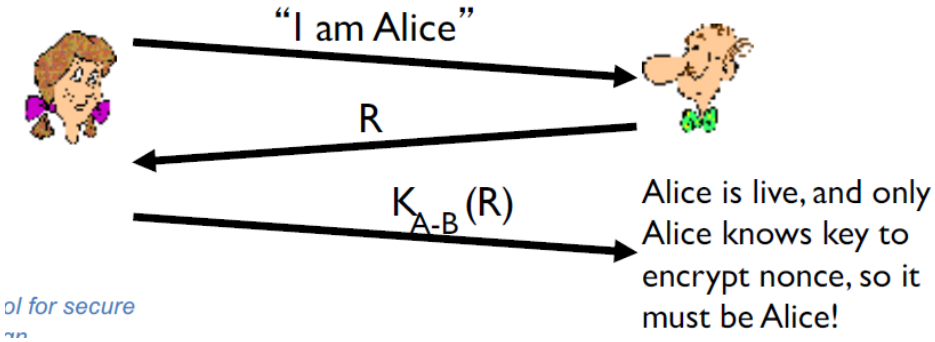
WEP authentication

End-point authentication w/ nonce：

Nonce: number (R) used only once –in-a-lifetime

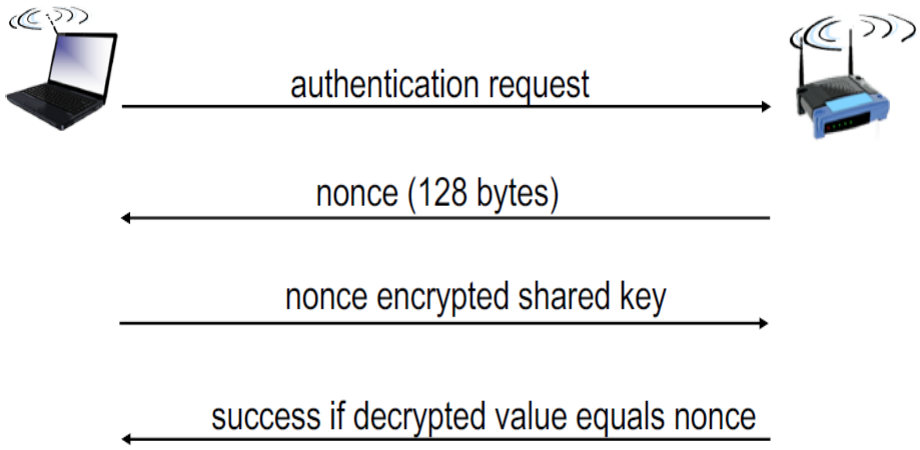
How to prove Alice "live": Bob sends Alice nonce, R.

Alice must return R, encrypted with shared secret key



ol for secure

WEP authentication



Notes:

- not all APs do it, even if WEP is being used
- AP indicates if authentication is necessary in beacon frame
- done before association

The AP may use Beacons which also show in the parameters in the AP to do this,

Beacons are the number of announcements packets sent by the AP. Each access point sends about ten beacons per second at the lowest rate (1M), so they can usually be picked up from very far.

AP可以使用在AP参数中显示的信标来完成此操作，信标是AP发送的公告包数。每个接入点每秒以最低速率（1米）发送大约10个信标，因此通常可以从很远的地方接收。

Breaking WEP encryption

Security hole:

- 24-bit IV, one IV per frame, -> IV's eventually reused
  ~16Million IVs at high speed exhausted in 2 hours
- IV transmitted in ***plaintext*** -> IV reuse detected

Attack:

- Trudy causes Alice to encrypt known plaintext d_1 d_2 d_3 d_4 …
- Trudy sees: $c_i = d_i$ XOR k_i^{IV}
- Trudy knows c_i d_i, so can compute $k_i^{IV} = d_i$ XOR c_i
- Trudy knows encrypting key sequence k_1^{IV} k_2^{IV} k_3^{IV} …
- Next time IV is used, Trudy can decrypt!

监听模式被动破解(这个就是有客户端并有大量有效通信):根据已知的的信息。我们知道要还原出WEP的密码关键是要收集足够的有效数据帧，从这个数据帧里我们可以提取IV 值和密文。与对于这个密文对应的明文的第一个字节是确定的他是逻辑链路控制的802.2 头信息。通过这一个字节的明文，还有密文我们做XOR运算能得到一个字节的WEP密钥流，由于rc4 流密码产生算法只是把原来的密码给打乱的次序。所以我们获得的这一次字节的密码就是IV+PASSWORD 的一部分。但是由于RC4 的打乱。不知道这一个字节具体的位置很排列次序。当我们收集到足够多的IV 值还有碎片密码时，就可以进行统计分析运算了。用上面的密码碎片重新排序配合IV使用RC4算法得出的值和多个流密码位置进行比较。最后得到这些密码碎片正确的排列次序。这样WEP的密码就被分析出来了。

主动攻击(有客户端。少量通信或者没有通讯): -3 ARP-request attack mode攻击抓取合法客户端的arp请求包。如果发现合法客户端发给AP的arp请求包，攻击者就会向AP重放这个包。由于802.11b 允许IV重复使用。所以AP接到这样的arp请求后就会回复客户端。这样攻击者就能搜集到更多的IV了。当捕捉到足够多的IV就可以按上面的2.9.1里的进行破解了。如果没有办法获取arp请求包我们就可以用-0 攻击使得合法客户端和AP断线后重新连接。-0 Deautenticate攻击实际就是无线欺骗。这样我们就有机会获得arp请求包了。

主动攻击(没有客户端的模式) :先和AP进行伪链接-1 fakeauth count attack mode。这样就能产生数据包了。收集两个IV 相同的的WEP 包，把这两个包里的密文做XOR 运算。得到一个XOR 文件。用这个XOR文件配合伪造arp包的工具。利用CRC-32的特点伪造一个arp包和原来的IV一起发给AP。这样就可以按上面2.9.2 里的进行破解了。其中-2 Interactive，-4 Chopchop，-5 Fragment 都是属于上面这个攻击类型的。

Problem with Linear Checksum

Encrypted CRC-32 used as integrity check Vector (ICV)

- Fine for random errors, but not malicious ones
- Bits can be changed in packet without decrypting

An attacker can change encrypted content (substitute by gibberish), compute a CRC over the substituted text and produce an 802.11 frame that will be accepted by the receiver.

Wi-Fi Protected Access (WPA)

Two versions WPA and WPA2

- WPA temporary solution to fix WEP while WPA2 developed

WPA compatible with existing hardware that supported WEP

WPA uses Temporal Key Integrity Protocol (TKIP)

- Used RC4 for compatibility
- Every packet encrypted with unique encryption key

802.11i: WPA – New Features

To provide stronger authentication than in WEP:

- Special purpose Message Integrity Code (MIC) as opposed to WEP CRC

To prevent Fluhrer, Mantin and Shamir (FMS) aka FMS-style attacks

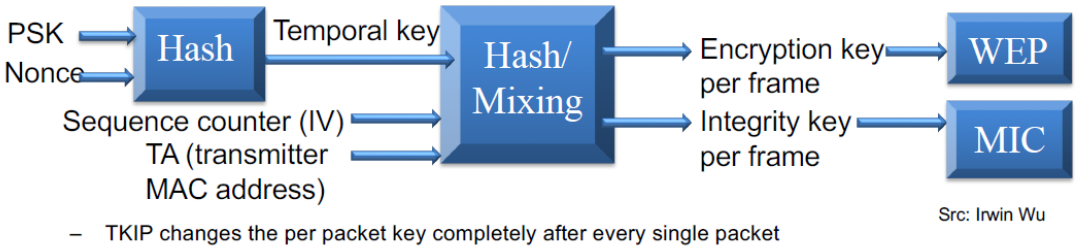
- a new *per-frame key is* constructed using a cryptographic hash

Temporal Key Integrity Protocol (TKIP) uses a cryptographic mixing function to combine a temporal key, the TA (transmitter MAC address), and the sequence counter into the WEP seed (128 bits)

- Pre Shared Key (PSK) AKA WPA-Personal similar to WEP-Key
  - However, it is not used for encryption
- Instead, PSK serves as the seed for hashing the per-frame key

要提供比WEP更强大的身份验证:
–专用消息完整性代码（MIC），而不是WEP CRC
•防止福勒、曼汀和沙米尔（fms）即fms式攻击
–使用加密哈希构造新的每帧密钥
•时态密钥完整性协议（TKIP）使用密码混合功能将时态密钥、TA（发送器MAC地址）和序列计数器组合到WEP种子（128位）中。
–预共享密钥（PSK），即与WEP密钥类似的WPA个人密钥。但是，它不用于加密
–相反，psk是散列每帧键的种子

How does it work?



Src: Irwin Wu

- TKIP changes the per packet key completely after every single packet
  - One key for encryption (128 bits)
  - One key for integrity (64 bits)
- The WEP IV is extended to 48 bits, and used as a packet sequence counter
  - A per packet sequence counter is used to prevent replay attacks
  - If a packet is received out of order, it is dropped by the receiving station

802.11 frame with WPA

FMS Attack

Fluhrer, Mantin and Shamir (FMS) attack

- For 50% success rate, capture around 5 Million packets on average
- Due to inherent weakness in RC4, output of encrypting with first few bytes of key not random
- Certain key values generate predictable pattern of encrypted data
  - Associated packets are IVs are “weak”
- Initial determine first bytes of key through IVs and then get the rest through statistical analysis
- Encrypted ARP packets can be captured and replayed to get encrypted ARP response

WEP vs WPA security

WPA temporary solution to fix WEP while WPA2 developed

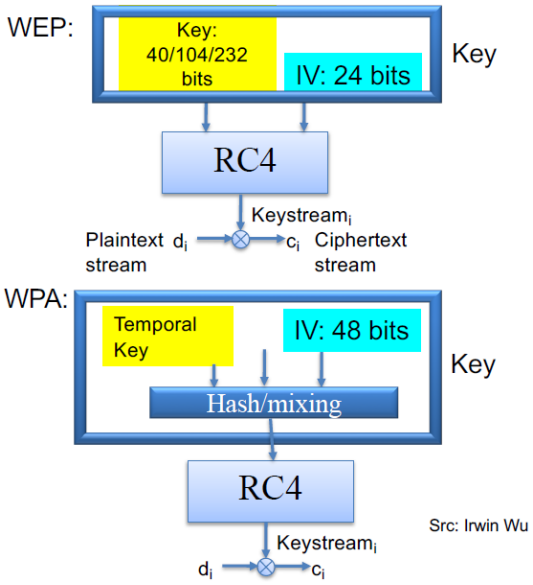
WEP IV extended to 48-bit IV

- Reuse > 100 years for replay of the same IV

RC4 key = Function(WPA Key||IV)

- Every packet encrypted with unique encryption key

IV used as a packet sequence space to prevent replay attack



WPA加密算法的两个版本介绍

WPA = 802.1x + EAP + TKIP + MIC

= Pre-shared Key + TKIP + MIC 802.11i(WPA2)

= 802.1x + EAP + AES + CCMP

= Pre-shared Key + AES + CCMP

这里802.1x + EAP，Pre-shared Key是身份校验算法（WEP没有设置有身份验证机制），TKIP和AES是数据传输加密算法（类似于WEP加密的RC4 算法），MIC和CCMP数据完整性编码校验算法（类似于WEP中CRC32算法）

WPA 认证方式

802.1x + EAP （工业级的，安全要求高的地方用。需要认证服务器）

Pre-shared Key （家庭用的，用在安全要求低的地方。不需要服务器）

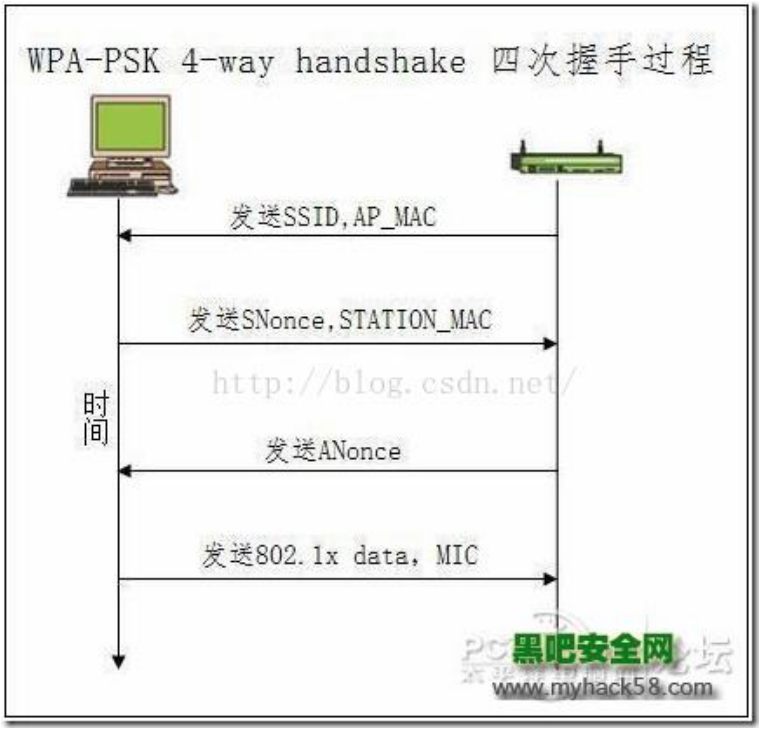
EAP 扩展认真协议，是一种架构。而不是定义了算法。常见的有LEAP，MD5，TTLS，TLS，PEAP，SRP，SIM，AKA 其中的TLS 和TTLS 是双向认证模式。这个和网络银行的安全方式差不多。这个认证方式是不怕网络劫持和字典攻击的。而md5 是单向认证的。不抗网络劫持，中间人攻击。关于企业级的如何破解就不讨论了。因为论坛上也很少提到。本身EAP模式是个协议，不是算法。

WPA-PSK

论坛上破解WPA 也主要是集中在这个模式上的。我们都知道破解WPA-PSK 不是和WEP一样抓很多包就能破解的。关键是要获取握手包，这个握手包叫4way-handshake 四次握手包。那么我们就从这个四次握手包开始。

四次握手

通信过程如图:



WPA-PSK 初始化工作

使用 SSID 和 passphares 使用以下算法产生PSK 在WPA-PSK 中PMK=PSK ，PSK=PMK=pdkdf2_SHA1(passphrase，SSID，SSID length，4096)

第一次握手

AP广播SSID，AP_MAC(AA)→STATION

STATION 端使用接受到的SSID，AP_MAC(AA)和passphares使用同样算法产生PSK

第二次握手

STATION 发送一个随机数SNonce，STATION_MAC(SA)→AP

AP端接受到SNonce，STATION_MAC(SA)后产生一个随机数ANonce，然后用PMK，AP_MAC(AA)，STATION_MAC(SA)，SNonce，ANonce 用以下算法产生PTK PTK=SHA1_PRF(PMK， Len(PMK)，"Pairwise key expansion"，MIN(AA，SA) || Max(AA，SA) || Min(ANonce，SNonce) || Max(ANonce，SNonce)) 提取这个PTK 前16 个字节组成一个MIC KEY

第三次握手

AP发送上面产生的ANonce→STATION

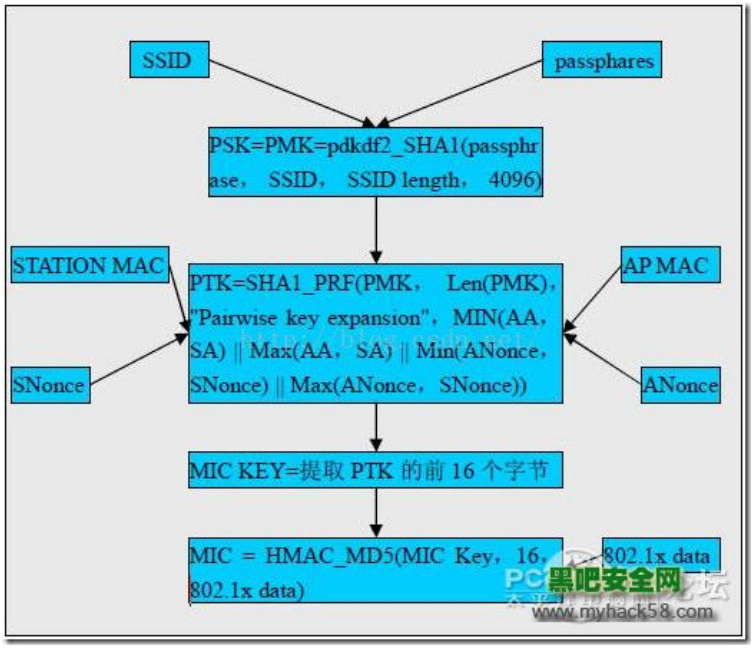
STATION 端用接收到ANonce 和以前产生PMK，SNonce，AP_MAC(AA)，STATION_MAC(SA) 用同样的算法产生PTK。提取这个PTK 前16 个字节组成一个MIC KEY使用以下算法产生MIC值用这个MIC KEY 和一个802.1x data 数据帧使用以下算法得到MIC值 MIC = HMAC_MD5(MIC Key，16，802.1x data)

第四次握手

STATION 发送802.1x data ， MIC→AP

STATION 端用上面那个准备好的802.1x 数据帧在最后填充上MIC值和两个字节的0（十六进制）让后发送这个数据帧到AP。

AP端 收到这个数据帧后提取这个MIC。并把这个数据帧的MIC部分都填上0（十六进制）这时用这个802.1x data 数据帧，和用上面AP产生的MIC KEY 使用同样的算法得出MIC'。如果MIC'等于STATION发送过来的MIC。那么第四次握手成功。若不等说明则AP 和STATION 的密钥不相同，或STATION 发过来的数据帧受到过中间人攻击，原数据被篡改过。握手失败了。



Breaking WPA

字典攻击

寻找可以攻击的信息元素字典攻击作为一种常用的攻击手段要明白的是从那里开始攻击。要寻找和密码有有联系的信息元素。在WPA 中和密码有联系的信息有数据的传送包和四次握手包。由于无法知道明文，和WPA的数据加密算法的复杂性。在数据传输包上要找到可以攻击的信息元素基本上很难实现。所以只能在握手包里寻找有密码有联系的信息。在上面的四次握手包的图片中很清楚的表明，在四川握手中主要传递的有如下数据：SSID，AP_MAC，STATION_MAC，SNonce，ANonce，802.1x data，MIC。前面6 个元素很清楚，一般不会和密码有联系的。只有最后一个MIC和密码有所联系。通过MIC的派生图我们知道，**MIC是通过上面六个信息元素和密码通过三个主要的算法派生出来的。**那么我们是不是只要找到这三个算法的逆反算法就可以根据上面的7 个信息元素把密码计算出来了呢。的确实这样。但是这三个算法有一个共同的名字叫HASH 函数。

HASH 函数

HASH 函数是不可能从生产的散列值来唯一的确定输入值。

a. 单向性(one-way)。HASH 函数是没有反函数的。

b. 抗冲突性(collision-resistant)。要寻找两个hash值相同的原值十分困难。

c. 映射分布均匀性和差分分布均匀性。不像普通函数那样数值分布有一定规律。

由于上面的pdkdf2_SHA1,SHA1_PRF,HMAC_MD5是HASH 函数。所以我们就基本上无法直接计算出密码。对于HASH 函数比较有效的攻击就是建立HASH 字典攻击。 HASH 字典就是把预先算好的HASH 值按照线性排列然后组成一个数据库。当我们知道 一个HASH 值时在这个数据库里能马上找到他的原值。当然这个过程是通过数据库实现的而不是HASH 的逆反函数。所以有些HASH 值在这样的数据库里是找不到原值的。

由于HASH 库是线性的所以。所以在HASH 库里找数据是十分迅速的。还有一点有的人也知道的国内的王小云教授对于部分HASH 算法有突出贡献的。她的主要贡献是寻找碰撞值。暴力破解的话就是大概需要2^80量级的MD5 HASH 运算。被王教授提高到只需要2^69 量级的MD5 HASH 运算就能够找到一个碰撞。我们有这样两个对付HASH 函数的方法。那么对我们破解我怕密码是不是如虎添翼了呢？

4.1.2 HMAC (HASH Message Authentication Code)哈希消息校验算法

这里我承认我刚才有骗过你。pdkdf2_SHA1,SHA1_PRF,HMAC_MD5 不是HASH函数。当然我骗你我也有我的理由啦。第一我以前被别人骗过，某论坛上说建立HASH库然后进行WPA破解的。能建立HASH 库那三个函数不是HASH 函数那是什么啊。第二我不是故意的。了解HASH 函数，有助于你理解HMAC 算法。所以pdkdf2_SHA1,SHA1_PRF,HMAC_MD5是HMAC算法。不是HASH 函数。HMAC算法就是用一个密码，和一个消息。最后生成一个HASH值。由上面的介绍，我们可以看出，HMAC算法更像是一种加密算法，它引入了密钥，其安全性已经不完全依赖于所使用的HASH 算法。所以上面对HASH 的攻击，对于HMAC 是没有效果的。HMAC特别是象“挑战/响应”身份认证应用中，由于攻击者无法事先获得HMAC 的计算结果，对系统的攻击只能使用穷举或“生日攻击”的方法，但计算量巨大，基本不可行。所以，在目前的计算能力下，可以认为HMAC算法在“挑战/响应”身份认证应用中是安全的。

四次握手包

有上面的HMAC 的特性我们也不难得出SSID，AP_MAC，STATION_MAC，SNonce，ANonce，802.1x data，这些信息元素都是上面的HMAC算法里的消息。HMAC算法里的密码在pdkdf2_SHA1 算法里是WPA 的密码，在SHA1_PRF 算法里是PMK，在HMAC_MD5算法里是PTK。最后才得出MIC值。由于这些消息和这个MIC值都有关联性。所以四次握手吧的后面三次是缺一不可的。而且是有时效性的。不能把不是同一次的握手包拼起来使用的。当然第一次握手包的SSID 和AP-MAC是可以后获取的。这里你也明白了四次握手中根本是不是在传递一个简单的HASH 值。而是要传递一个HMAC 值。如果是传递一个简单的HASH 值，那么我们只要获取后重播这个值就可以欺骗AP 获得认证了。都不要知道这个HASH 值对应的原值。但我的这么好的想法被HMAC给打破了。

面向于四次握手包的字典攻击

字典攻击，就是把密码的可能性罗列起来组成一个密码字典。然后把字典里的密码和SSID，AP_MAC，STATION_MAC，SNonce，ANonce，802.1x data，这些信息元素。通过pdkdf2_SHA1,SHA1_PRF,HMAC_MD5 这些算法最后生成MIC’（具体过程看上面MIC派生图）。当在字典里找到一个密码他的MIC’等于握手包中的MIC。这时字典破解成功。这就是我们要的那个密码。如果把字典里的所有密码都找遍了还有没有符合上述条件的。那么破解失败。

WPA2 和 WPA 的区别

WPA 标准于2006年正式被 WPA2 取代。 WPA 和 WPA2 之间最显着的变化之一是强制使用 AES 算法和引入 CCMP （计数器模式密码块链消息完整码协议）替代 TKIP 。

802.11i: WPA2

WPA2 2004

- New AP hardware, 30 Million Instructions/sec, RC4 off-load hardware doesn't do AES or CCMP
- AES-CCMP 128-bit AES
  - CCMP (Counter Mode with Cipher Block Chaining Message Authentication Code Protocol)
- Improved 4-way handshake and temporary key generation

We may cover some details of WPA2 but for now, if you have WPA2, this would be the safest option to use.

WPA- Enterprise network security (802.1X) in later weeks

Hotspot Security

For most hotspots: Unfortunately almost none!

If you do not have to configure any security parameters besides typing in a username and password in a web page, expect the following:

- The hotspot operator checks your authenticity at logon time (often protected with SSL to protect against eavesdropping on your password)
- Only authenticated clients will receive service as packet filtering is deployed to only allow accessing the logon page until successful authentication
- Once logon authentication has been checked: no further security measures
- No protection for your user data:

Everything can be intercepted and manipulated

- However, you can deploy your own measures, e.g. VPN or SSL, but configuration is often tedious or not even supported by communication partner and performance is affected because of additional (per-packet-) overhead
  - Plus: your session can be stolen by using your MAC & IP addresses!

对于大多数热点：不幸的是几乎没有！

•如果除了在网页中键入用户名和密码之外，您不必配置任何安全参数，则应满足以下要求：

- Hotspot操作员在登录时检查您的真实性（通常使用SSL进行保护，以防止窃听您的密码）

- 只有经过身份验证的客户机才能接收服务，因为数据包筛选部署为仅允许访问登录页，直到成功身份验证为止。

- 检查登录验证后：无进一步的安全措施

- 不保护您的用户数据：

•一切都可以被拦截和操纵

- 但是，您可以部署自己的度量，例如VPN或SSL，但是配置通常很冗长，甚至不受通信伙伴的支持，并且由于额外的（每个数据包）开销，性能会受到影响。

Plus：您的会话可以通过使用您的MAC&IP地址被盗！

Message Forgery

CRC-32 is *linear*, which means that it is possible to compute the bit difference of two CRCs based on the bit difference of the messages over which they are taken.

Flipping bit n in the message results in a deterministic set of bits in the CRC that must be flipped to produce a correct checksum on the modified message.

Because flipping bits carries through after an RC4 decryption, this allows the attacker to flip arbitrary bits in an encrypted message and correctly adjust the checksum so that the resulting message appears valid.
Implications:

- "Integrity check" does not prevent packet modification
- Can maliciously flip bits in packets
  - Modify active streams!
  - Bypass access control

Partial knowledge of packet is sufficient

CRC-32是线性的，这意味着可以根据接收到的信息的位差来计算两个CRC的位差。

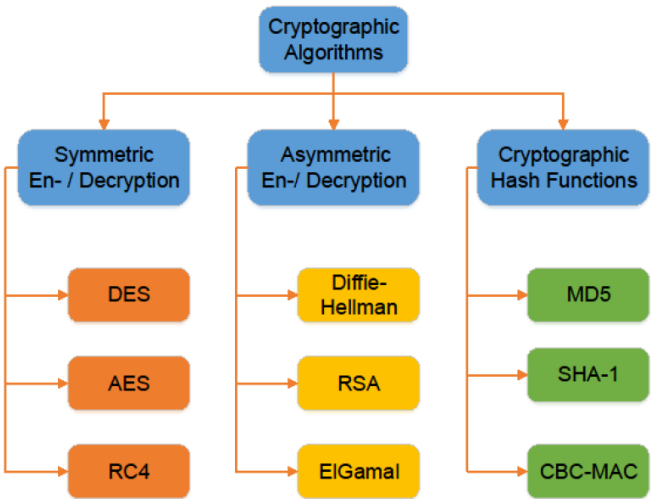
•翻转消息中的位n会在CRC中产生一组确定的位，必须翻转该位才能对修改后的消息生成正确的校验和。由于翻转位在RC4解密后进行，因此攻击者可以翻转加密消息中的任意位，并正确调整校验和，以使结果消息看起来有效。应用：完整性检查不会阻止数据包修改；可以恶意翻转数据包中的位；修改活动流；旁路访问控制；部分信息包知识足够

WK03: Authentication, Key Distribution (Asymmetric), Certification Authority

Overview

Authentication Recap
Key distribution using asymmetric encryption
  Public-key certificates
  Public-key distribution of secret keys
  Certification Authority and X.509
Symmetric key distribution using symmetric encryption
Kerberos
  Version 4
  Version 5

Asymmetric:不对称的

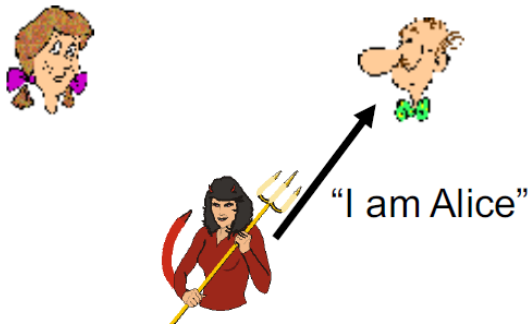


Authentication

端点鉴别

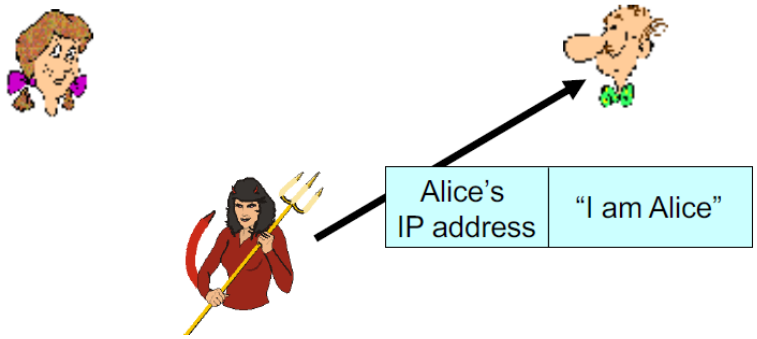
Goal: Bob wants Alice to "prove" her identity to him

Protocol ap1.0: Alice says "I am Alice"



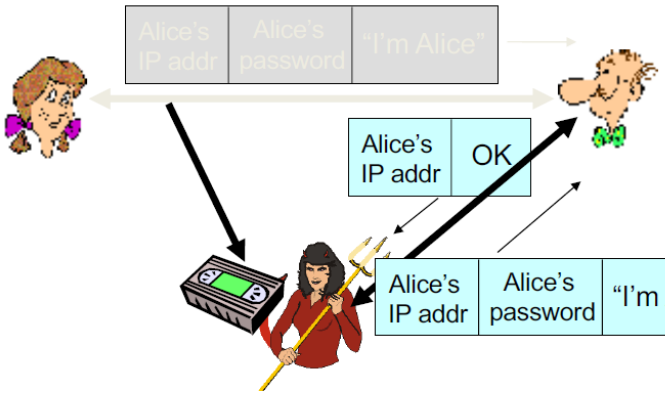
In a network,
Bob can not "see" Alice, so
Eve simply declares
herself to be Alice

Protocol ap2.0: Alice says "I am Alice" in an IP packet containing her source IP address



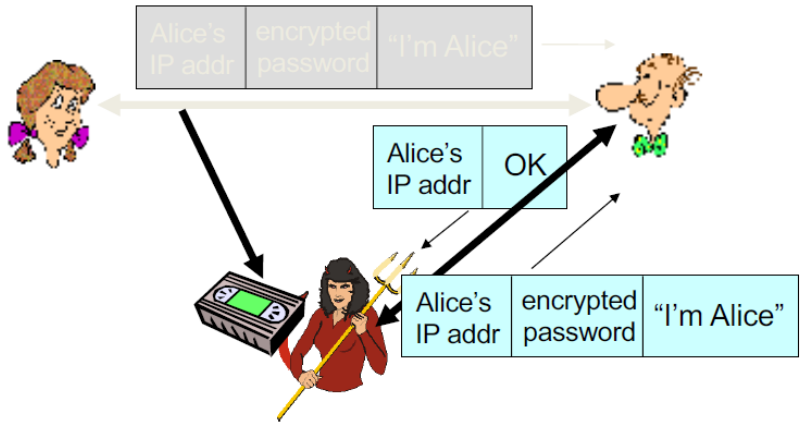
Eve can create
a packet "spoofing"
Alice's address

Protocol ap3.0: Alice says "I am Alice" and sends her secret password to "prove" it.



playback attack: Eve records Alice's packet and later plays it back to Bob

Protocol ap3.1: Alice says "I am Alice" and sends her *encrypted* secret password to "prove" it.

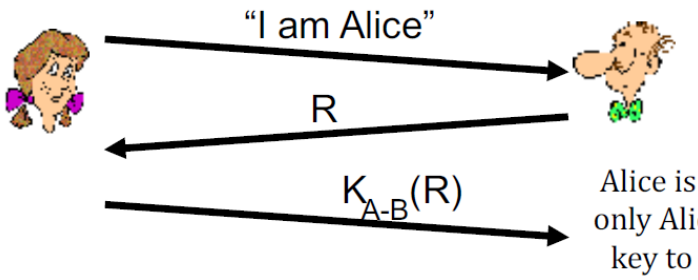


record and playback *still* works!

Goal: avoid playback attack

nonce: number (R) used only *once-in-a-lifetime*

ap4.0: to prove Alice "live", Bob sends Alice *nonce*, R. Alice must return R, encrypted with shared secret key



Alice is live, and only Alice knows key to encrypt nonce, so it must be Alice!

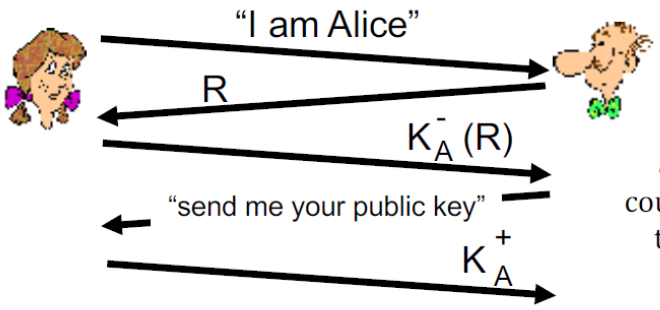
Failures, drawbacks?

Authentication: ap5.0

ap4.0 requires shared symmetric key

- can we authenticate using public key techniques?

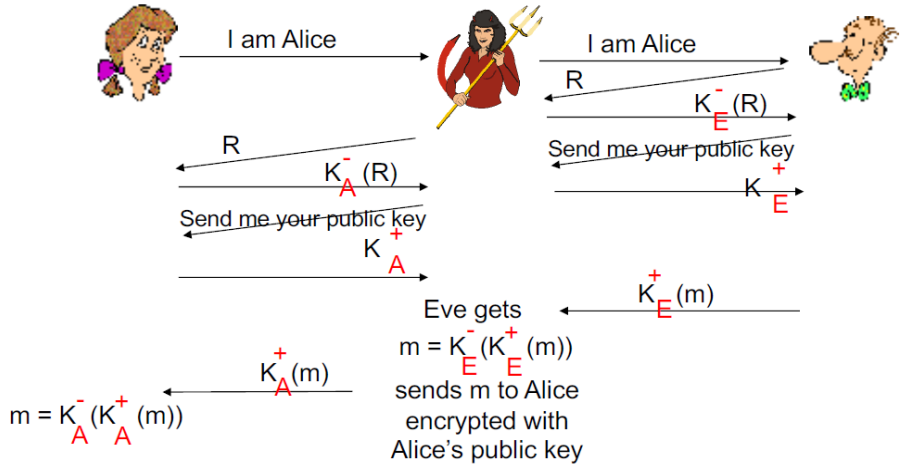
ap5.0: use nonce, public key cryptography



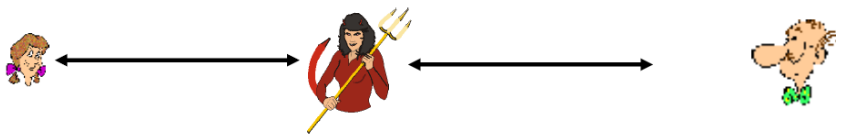
Bob computes
$K_A^+(K_A^-(R)) = R$
and knows only Alice could have the private key, that encrypted R such that
$K_A^+(K_A^-(R)) = R$

ap5.0: security hole

man (or woman) in the middle attack: Eve poses as Alice (to Bob) and as Bob (to Alice)



man (or woman) in the middle attack: Eve poses as Alice (to Bob) and as Bob (to Alice)



difficult to detect:

- Bob receives everything that Alice sends, and vice versa. (e.g., so Bob, Alice can meet one week later and recall conversation!)
- problem is that Eve receives all messages as well!

Public key encryption algorithms

Requirements:

1. need $K_B^+(\cdot)$ and $K_B^-(\cdot)$ such that

$$K_B^-(K_B^+(m)) = m$$

2. given public key K_B^+, it should be impossible to compute private key K_B^-

RSA: Rivest, Shamir, Adelson algorithm

Public Key Cryptography

symmetric key crypto

- requires sender, receiver know shared secret key
- Q: how to agree on key in first place (particularly if never "met")?

public key crypto

- radically different approach [Diffie-Hellman76, RSA78]
- sender, receiver do *not* share secret key
- *public* encryption key known to *all*
- *private* decryption key known only to receiver

Radically:根本上地

RSA

- A message is a bit pattern.
- A bit pattern can be uniquely represented by an integer number.
- Thus encrypting a message is equivalent to encrypting a number.

Example

- m= 10010001. This message is uniquely represented by the decimal number 145.
- To encrypt m, we encrypt the corresponding number, which gives a new number (the ciphertext).

RSA:

1. 选择两个大素数， p 和 q (越大越难破解)（e.g., 1024 bits each）
2. 计算 n=p*q 和 z = (p-1)*(q-1)
3. 选择小于 n 的数 e，且使 e 和 z 没有（非 1 的）公因数。使用字母 e 表示是因为这个值将被用于加密(e.g., 4 和 9 没有公因数，6 和 9 有公因数（3）)
4. 求一个数 d， 使 ed-1 可以被 z 整除，使用字母 d 是因为这个值将被用于解密，换句话说就是：ed mod z = 1
5. Bob 的公钥就是（n， e），私钥就是（n, d）

加密:

Alice 发送一个由整数 m 表示的比特组合给 Bob, 且 m<n, Alice 的明文报文 m 加密后就是:

$c = m^e \bmod n$ (i.e., remainder when m^e is divided by n)

c 就是被发送的密文

解密:

Bob 将 c 进行如下运算:

$m = c^d \bmod n$ (i.e., remainder when c^d is divided by n)

这就要求用他的私钥（n, d）， m 就是 Alice 所发送的消息

Example:

Bob chooses $p=5$, $q=7$. Then $n=35$, $z=24$.

$e=5$ (so e, z relatively prime).

$d=29$ (so $ed-1$ exactly divisible by z).

	letter	m	m^e	$c = m^e \bmod n$
encrypt:	l	12	1524832	17

	c	c^d	$m = c^d \bmod n$	letter
decrypt:	17	481968572106750915091411825223071697	12	l

上图中得me不是1524832， 而是12**5 = 248832，上图是一个计算错误，但后面得计算结果都是正确得。**

如下的公式符合公钥私钥加密体系：

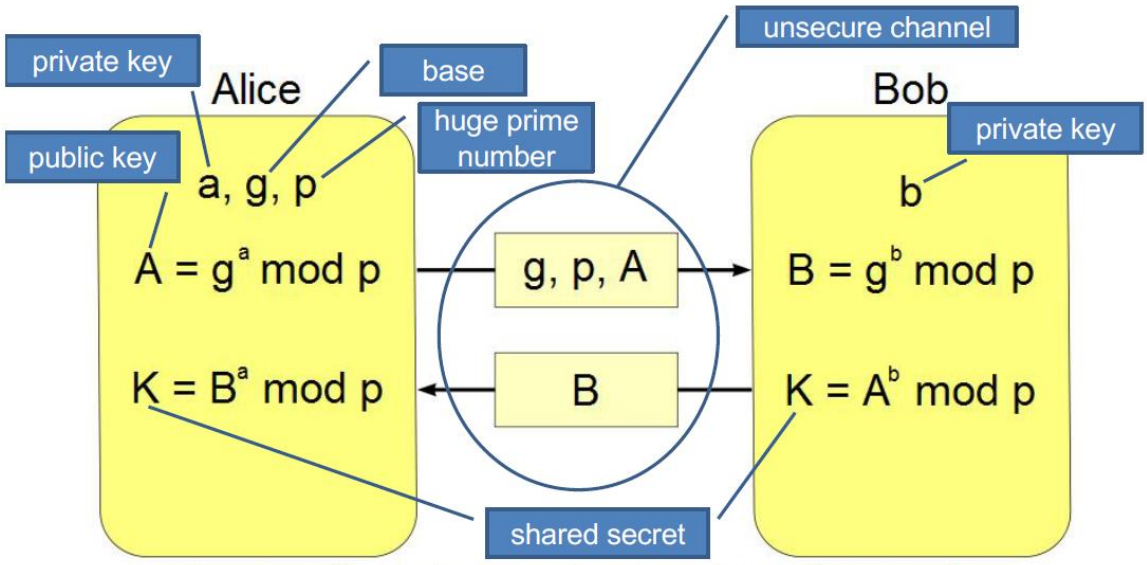
Magic happens! $m = (\underbrace{m^e \bmod n}_{c})^d \bmod n$

即是：

$$\underbrace{K_B^-(K_B^+(m))}_{\text{use public key first, followed by private key}} = m = \underbrace{K_B^+(K_B^-(m))}_{\text{use private key first, followed by public key}}$$

Result is the same!

Diffie-Hellman key exchange

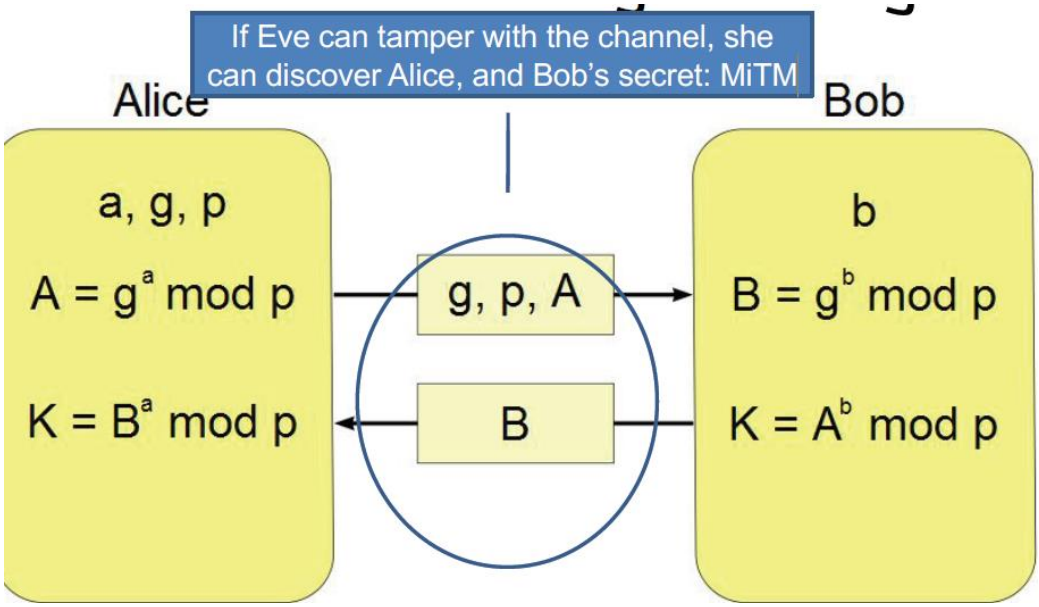


$K = A^b \bmod p = (g^a \bmod p)^b \bmod p = g^{ab} \bmod p = (g^b \bmod p)^a \bmod p = B^a \bmod p$

Alice's private key = 5, Bob's private key = 4, g=3, p=7

Alice's public key = $3^5 \bmod 7 = 5$, Bob's public key = $3^4 \bmod 7 = 4$

Alice's shared key = $4^5 \bmod 7 = 2$, Bob's shared key = $5^4 \bmod 7 = 2$

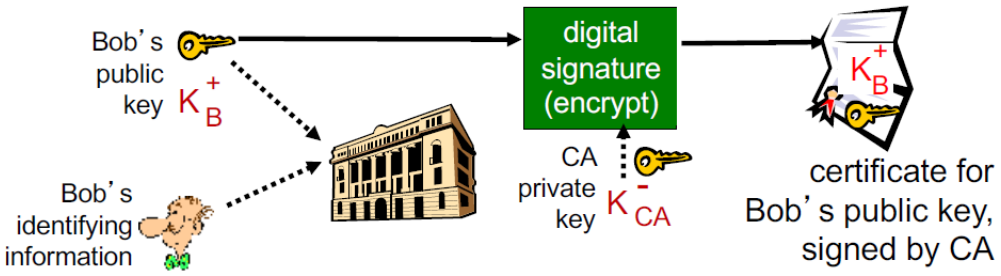


然而，该技术也存在许多不足： 没有提供双方身份的任何信息. 它是计算密集性的，因此容易遭受阻塞性攻击，即对手请求大量的密钥.受攻击者花费了相对多的计算资源来求解无用的幂系数而不是在做真正的工作. 没办法防止重演攻击. 容易遭受中间人的攻击.第三方C在和A通信时扮演B；和B通信时扮演A.A和B都与C协商了一个密钥，然后C就可以监听和传递通信量.中间人的攻击按如下进行： B在给A的报文中发送他的公开密钥. C截获并解析该报文.C将B的公开密钥保存下来并给A发送报文，该报文具有B的用户ID但使用C的公开密钥YC，仍按照好像是来自B的样子被发送出去.A收到C的报文后，将YC和B的用户ID存储在一块.类似地，C使用YC向B发送好像来自A的报文. B基于私有密钥XB和YC计算秘密密钥K1.A基于私有密钥XA和YC计算秘密密钥K2.C使用私有密钥XC和YB计算K1，并使用XC和YA计算K2. 从现在开始，C就可以转发A发给B的报文或转发B发给A的报文，在途中根据需要修改它们的密文.使得A和B都不知道他们在和C共享通信.

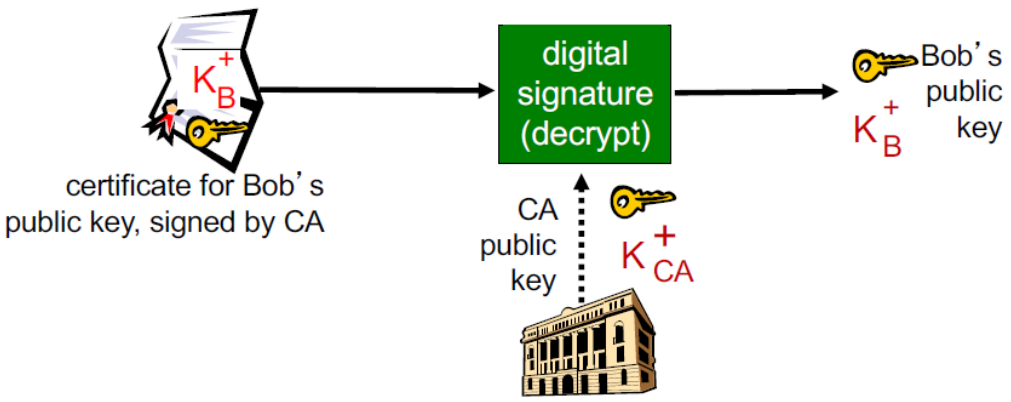
Public-key certification: Certification authorities

certification authority (CA): binds public key to particular entity, E.

- E (person, router) registers its public key with CA.
  - E provides "proof of identity" to CA.
  - CA creates certificate binding E to its public key.
  - certificate containing E's public key digitally signed by CA – CA says "this is E's public key"



- when Alice wants Bob's public key:
  - gets Bob's certificate (Bob or elsewhere).
  - apply CA's public key to Bob's certificate, get Bob's public key



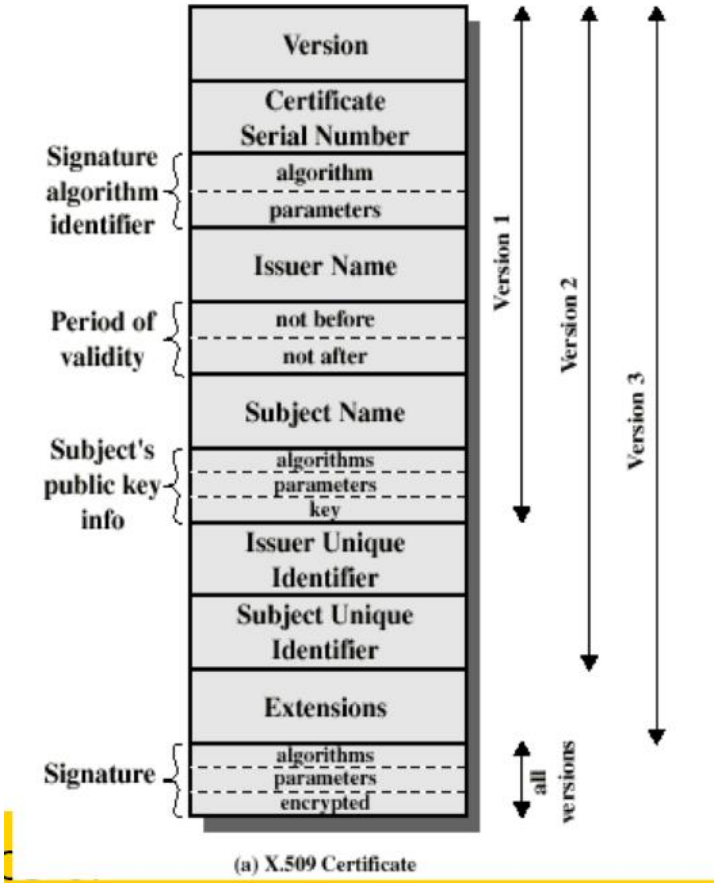
Public Key Distribution of Secret Key

- Prepare a message
- Encrypt that message using conventional encryption using one time session key
- Encrypt the session key using public-key encryption with Alice's **public key**
- Attach the encrypted session key to the message and send it to Alice
- Only Alice can decrypt the session key
- Bob has obtained Alice's public key by means of Alice's **public-key certificate**, must be a valid key

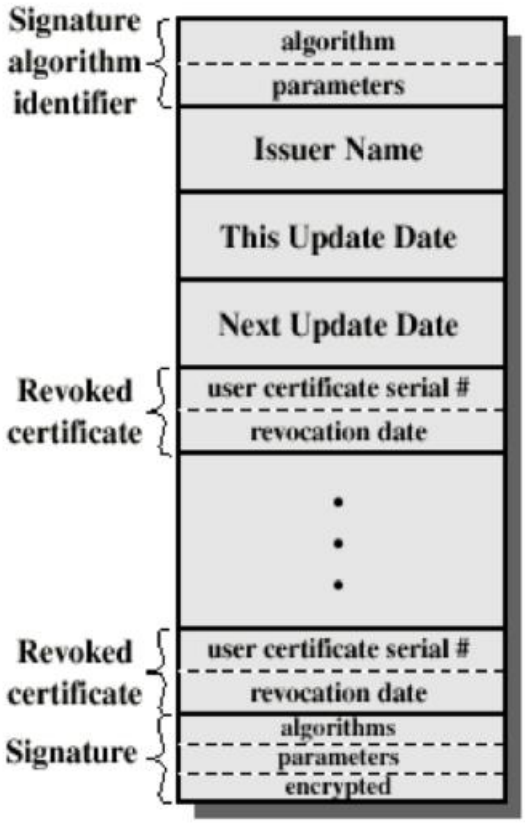
这里我们注意到，RSA 所要求的指数运算是相当耗费时间的过程。形成对比的是，DES 用软件实现要比 RSA 快 100 倍，用硬件实现则要快 1000 ~ 10 000 倍［RSA Fast 2012］。所以，在实际应用中，RSA 通常与对称密钥密码结合起来使用。例如，如果 Alice 要向 Bob 发送大量的加密数据，她可以用下述方式来做。首先，Alice 选择一个用于加密数据本身的密钥；这个密钥有时称为**会话密钥**（session key），该会话密钥表示为 K_S。Alice 必须把这个会话密钥告知 Bob，因为这是他们在对称密钥密码（如 DES 或 AES）中所使用的共享对称密钥。Alice 可以使用 Bob 的 RSA 公钥来加密该会话密钥，即计算 $c = (K_S)^e \bmod n$。Bob 收到了该 RSA 加密的会话密钥 c 后，解密得到会话密钥 K_S。Bob 此时已经知道 Alice 将要用于她的加密数据传输的会话密钥了。

X.509 Authentication Service

- Distributed set of servers that maintains a database about users.
- Each certificate contains the public key of a user and is signed with the private key of a CA.
- Is used in S/MIME, IP Security, SSL/TLS and SET.
- RSA is recommended to use but not mandatory.
- Digital Signature is assumed to use Hash algorithm
- Digital Certificate: user's id, public-key and CA information as input to hash function. Hash is then encrypted with CA's private key to produce **Digital Certificate**



(a) X.509 Certificate



(b) Certificate Revocation List

X.509证书身份验证的最常见用途是在使用SSL时验证服务器的身份，最常见的是在从浏览器使用HTTPS时。浏览器将自动检查服务器提供的证书是否已由其维护的可信证书颁发机构列表之一发出（即数字签名）。
您还可以使用SSL与“ 相互身份验证 ”；然后，服务器将从客户端请求有效证书，作为SSL握手的一部分。服务器将通过检查其证书是否由可接受的权限签名来验证客户端。如果提供了有效证书，则可以通过应用程序中的servlet API获取该证书。Spring Security X.509模块使用过滤器提取证书。它将证书映射到应用程序用户，并加载该用户的授权权限集，以便与标准Spring Security基础结构一起使用。

Obtaining a User's Certificate

Characteristics of certificates generated by CA:

- Any user with access to the public key of the CA can recover the user public key that was certified.
  - User can independently calculate hash, decrypt digital certificate using CA's public key, extract hash and compare if hashes match.
- No part other than the CA can modify the certificate without this being detected.

Certificates stored in a Directory server – not part of standard.

CA生成的证书的特征：

- 任何访问CA公钥的用户都可以恢复已认证的用户公钥。

用户可以独立计算哈希，使用CA的公钥解密数字证书，提取哈希并比较哈希是否匹配。

- 除了CA之外，任何其他部件都不能在未检测到该情况的情况下修改证书。

•存储在目录服务器中的证书-不是标准的一部分。

Distributed Directory

Users can be registered with a CA and would know its public Key
Now if A got its certificate from CA X1 and B got it from CA X2.
If A doesn't know CA X2's public key, it can't trust B's certificate issued by CA X2.
However, if the two CA's have securely exchanged their public keys, then it can work.

- A obtains the certificate of X2 signed by X1
- A knows securely X1's public key
- A obtains X2's public key from certificate and can verify using X1's signature on certificate
- A can now get B's certificate from CA X2.
- Since it has trusted public key for CA X2, things work as usual.

用户可以在CA中注册并知道其公钥：

•现在，如果a从ca x1获得证书，b从ca x2获得证书。

•如果a不知道ca x2的公钥，则不能信任ca x2颁发的b的证书。

•但是，如果两个CA安全地交换了他们的公钥，那么它就可以工作。

– a获得由x1签署的x2证书

– A安全地知道X1的公钥

– a从证书中获取x2的公钥，并可以使用x1在证书上的签名进行验证。

– A现在可以从CA X2获得B的证书。

– 因为它信任CA x2的公钥，所以一切照常工作。

Distributed Directory: Certificate Chain

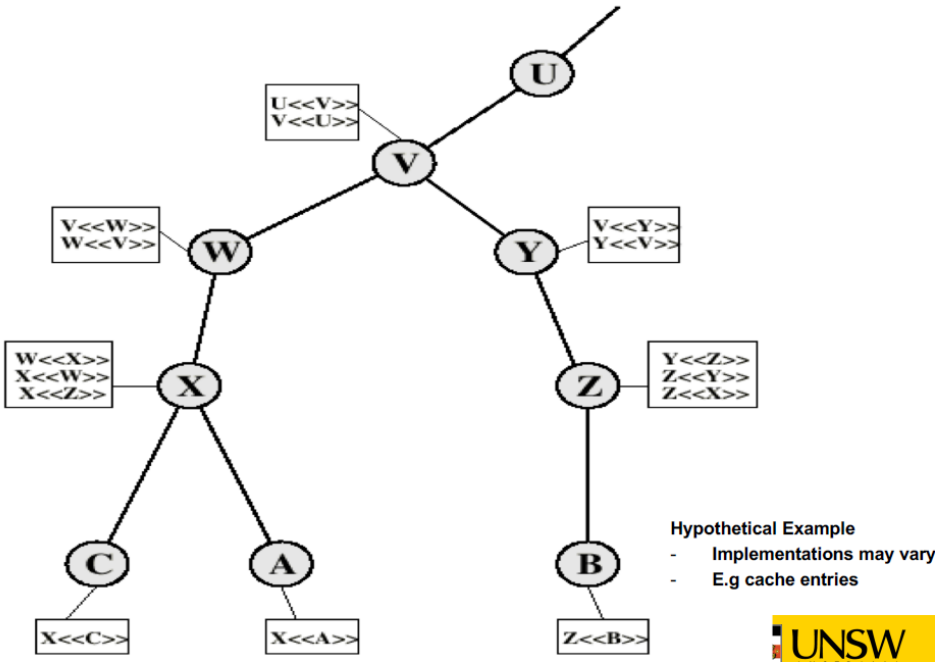
- Notation Y <<X>> Certificate of user X issued by authority Y
- A obtains B's public key using the following X.509 notation

  X_1 << X_2>> X_2 << B>>
- B obtains A's public key using the following X.509 notation

  X_2<< X_1>> X_1 << A>>
  - *Arbitrary chain is possible as long as consecutive pair* (X_n, X_{n+1}) *of CAs have exchanged certificates securely*

Notation:符号, arbitrary:任意的, consecutive:连续的

X.509 CA Hierarchy



Previous figure: Connected Circles hierarchical relationship, boxes shows certificates maintained in each CA's directory

– Forward Certs: Certs of X generated by other CAs (e.g at circle X, W<<X>>) – PARENT

– Reverse Certs: Certs generated by X for others. (e.g. at circle X, X<<C>> X<<A>>) - CHILD

A can acquire the following Certs from the directory to establish as certification to B

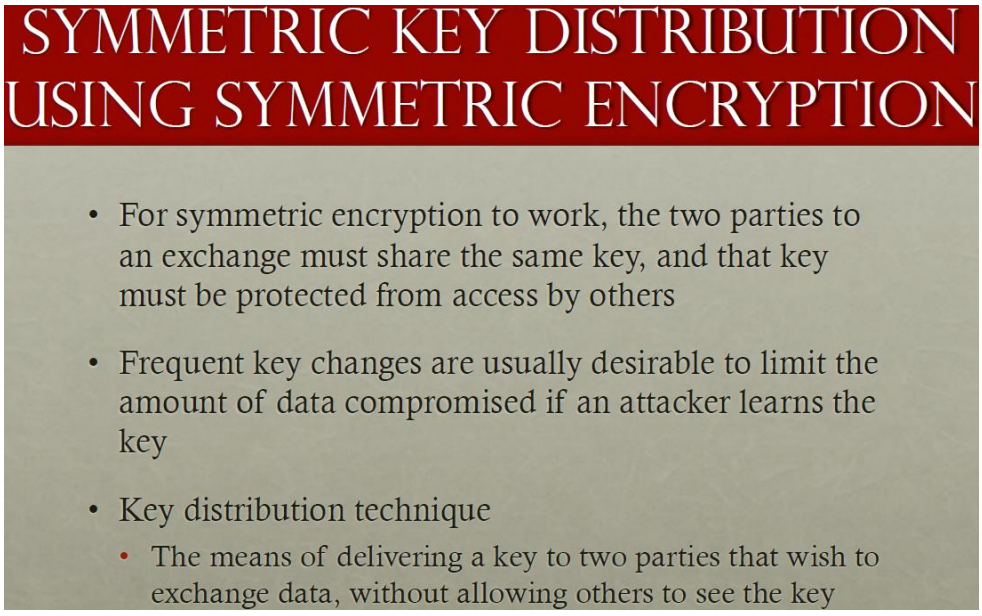
X <<W>> W <<V>> V <<Y>> Y<<Z>>Z<<B>>

(Try to get A's certificate)

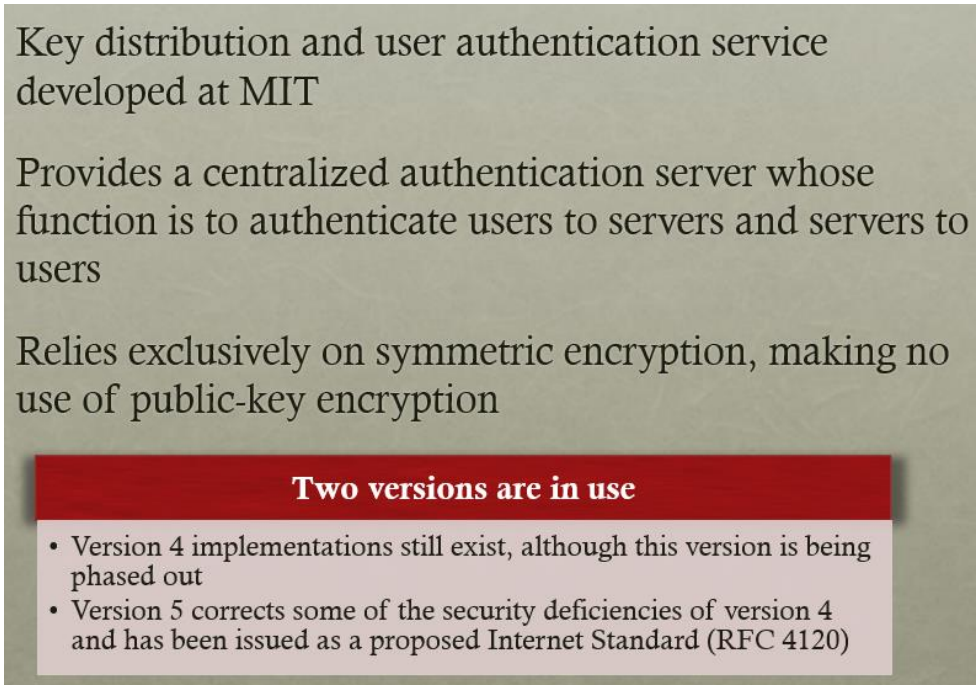
Revocation of Certificates（移除认证）

- Reasons for revocation:
  - The users secret key is assumed to be compromised.
  - The user is no longer certified by this CA.
  - The CA's certificate is assumed to be compromised.

WK03: Kerberos



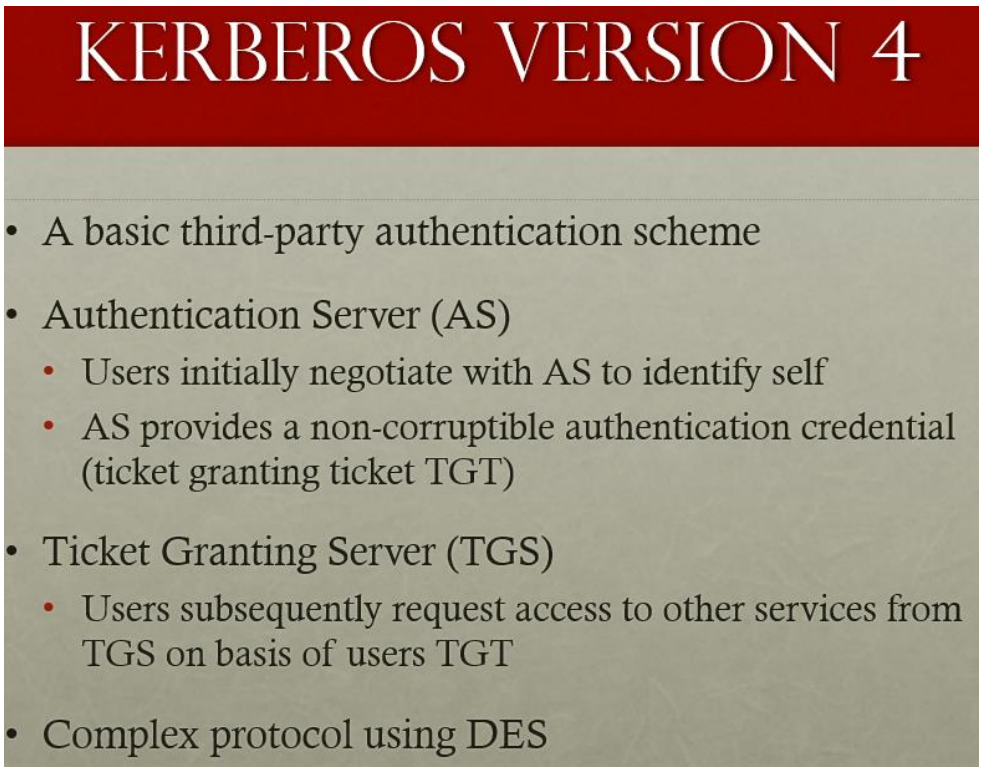
如果攻击者知道密钥，通常需要频繁的密钥更改来限制被破坏的数据量。



KERBEROS是在MIT开发的密钥分发和用户身份验证服务

•提供一个集中式身份验证服务器，其功能是将用户验证到服务器，将服务器验证到用户。

•完全依赖对称加密，不使用公钥加密



1.基本第三方认证方案 2.身份验证服务器（AS）•用户最初与AS协商确定自己 •AS提供不可损坏的身份验证凭证（Ticket Granting Ticket TGT）3.票据授予服务器（TGS） •用户随后根据用户TGT从TGS请求访问其他服务 4.使用DES的复杂协议

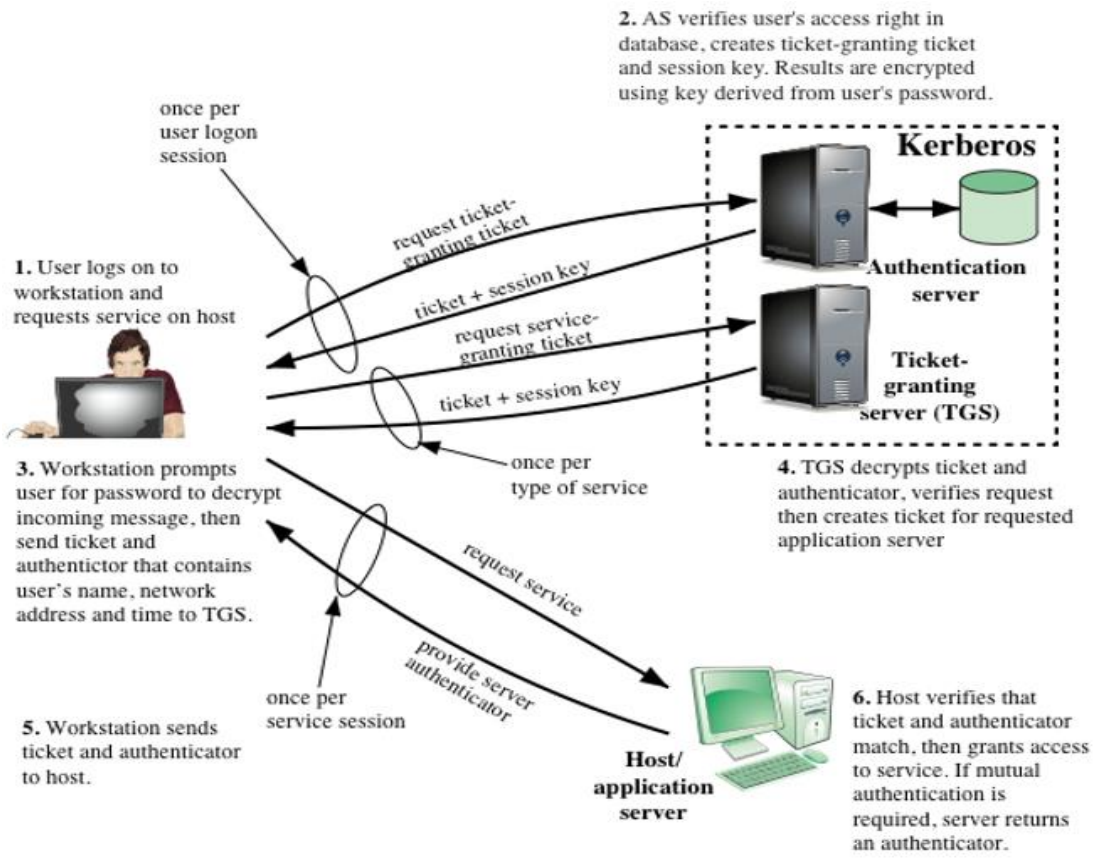


Figure 4.1 Overview of Kerberos

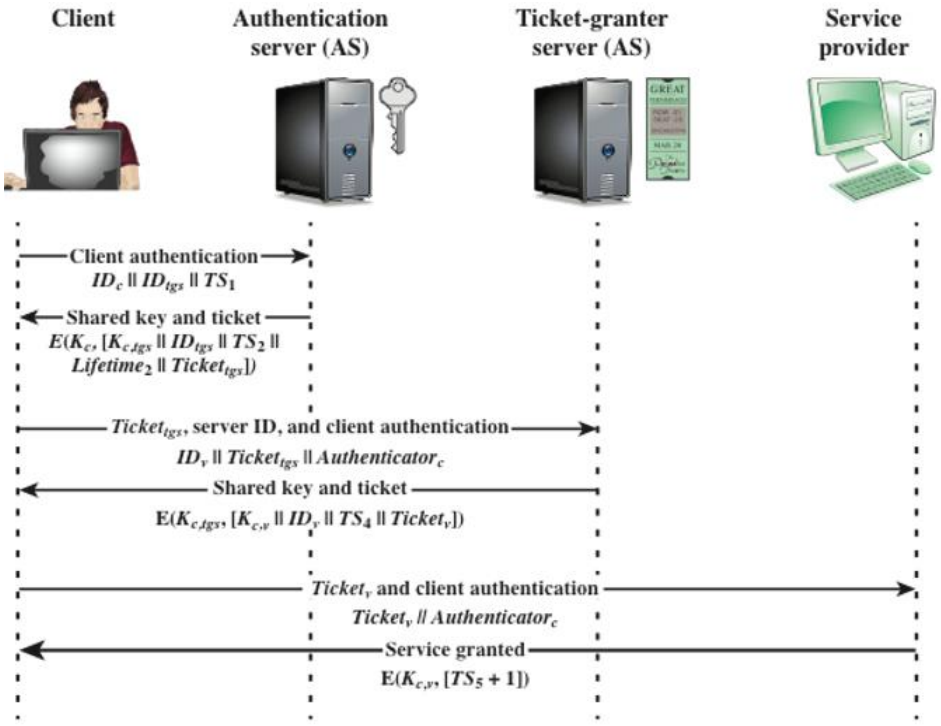


Figure 4.2 Kerberos Exchanges

(1) C → AS $ID_c \| ID_{tgs} \| TS_1$

(2) AS → C $E(K_c, [K_{c,tgs} \| ID_{tgs} \| TS_2 \| Lifetime_2 \| Ticket_{tgs}])$

$Ticket_{tgs} = E(K_{tgs}, [K_{c,tgs} \| ID_C \| AD_C \| ID_{tgs} \| TS_2 \| Lifetime_2])$

(a) Authentication Service Exchange to obtain ticket-granting ticket

(3) C → TGS $ID_v \| Ticket_{tgs} \| Authenticator_c$

(4) TGS → C $E(K_{c,tgs}, [K_{c,v} \| ID_v \| TS_4 \| Ticket_v])$

$Ticket_{tgs} = E(K_{tgs}, [K_{c,tgs} \| ID_C \| AD_C \| ID_{tgs} \| TS_2 \| Lifetime_2])$

$Ticket_v = E(K_v, [K_{c,v} \| ID_C \| AD_C \| ID_v \| TS_4 \| Lifetime_4])$

$Authenticator_c = E(K_{c,tgs}, [ID_C \| AD_C \| TS_3])$

(b) Ticket-Granting Service Exchange to obtain service-granting ticket

(5) C → V $Ticket_v \| Authenticator_c$

(6) V → C $E(K_{c,v}, [TS_5 + 1])$ (for mutual authentication)

$Ticket_v = E(K_v, [K_{c,v} \| ID_C \| AD_C \| ID_v \| TS_4 \| Lifetime_4])$

$Authenticator_c = E(K_{c,v}, [ID_C \| AD_C \| TS_5])$

(c) Client/Server Authentication Exchange to obtain service

Table 4.2 Rationale for the Elements of the Kerberos Version 4 Protocol
(page 1 of 3)

Message (1)	Client requests ticket-granting ticket.
ID_C	Tells AS identity of user from this client.
ID_{tgs}	Tells AS that user requests access to TGS.
TS_1	Allows AS to verify that client's clock is synchronized with that of AS.
Message (2)	AS returns ticket-granting ticket.
K_c	Encryption is based on user's password, enabling AS and client to verify password, and protecting contents of message (2).
$K_{c,tgs}$	Copy of session key accessible to client created by AS to permit secure exchange between client and TGS without requiring them to share a permanent key.
ID_{tgs}	Confirms that this ticket is for the TGS.
TS_2	Informs client of time this ticket was issued.
$Lifetime_2$	Informs client of the lifetime of this ticket.
$Ticket_{tgs}$	Ticket to be used by client to access TGS.

(a) Authentication Service Exchange

Table 4.2 Rationale for the Elements of the Kerberos Version 4 Protocol
(page 2 of 3)

Message (3)	Client requests service-granting ticket.
ID_V	Tells TGS that user requests access to server V.
$Ticket_{tgs}$	Assures TGS that this user has been authenticated by AS.
$Authenticator_c$	Generated by client to validate ticket .
Message (4)	TGS returns service-granting ticket.
$K_{c,tgs}$	Key shared only by C and TGS protects contents of message (4).
$K_{c,v}$	Copy of session key accessible to client created by TGS to permit secure exchange between client and server without requiring them to share a permanent key.
ID_V	Confirms that this ticket is for server V.
TS_4	Informs client of time this ticket was issued.
$Ticket_V$	Ticket to be used by client to access server V.
$Ticket_{tgs}$	Reusable so that user does not have to reenter password.
K_{tgs}	Ticket is encrypted with key known only to AS and TGS, to prevent Tampering.
$K_{c,tgs}$	Copy of session key accessible to TGS used to decrypt authenticator, thereby authenticating ticket.
ID_C	Indicates the rightful owner of this ticket.
AD_C	Prevents use of ticket from workstation other than one that initially requested the ticket.
ID_{tgs}	Assures server that it has decrypted ticket properly.
TS_2	Informs TGS of time this ticket was issued.
$Lifetime_2$	Prevents replay after ticket has expired.
$Authenticator_c$	Assures TGS that the ticket presenter is the same as the client for whom the ticket was issued has very short lifetime to prevent replay.
$K_{c,tgs}$	Authenticator is encrypted with key known only to client and TGS, to prevent tampering.
ID_C	Must match ID in ticket to authenticate ticket.
AD_C	Must match address in ticket to authenticate ticket.
TS_3	Informs TGS of time this authenticator was generated.

(b) Ticket-Granting Service Exchange

Table 4.2 Rationale for the Elements of the Kerberos Version 4 Protocol
(page 3 of 3)

Message (5)	Client requests service.
$Ticket_V$	Assures server that this user has been authenticated by AS.
$Authenticator_c$	Generated by client to validate ticket.
Message (6)	Optional authentication of server to client.
$K_{c,v}$	Assures C that this message is from V.
$TS_5 + 1$	Assures C that this is not a replay of an old reply.
$Ticket_v$	Reusable so that client does not need to request a new ticket from TGS for each access to the same server.
K_v	Ticket is encrypted with key known only to TGS and server, to prevent Tampering.
$K_{c,v}$	Copy of session key accessible to client; used to decrypt authenticator, thereby authenticating ticket.
ID_C	Indicates the rightful owner of this ticket.
AD_C	Prevents use of ticket from workstation other than one that initially requested the ticket.
ID_V	Assures server that it has decrypted ticket properly.
TS_4	Informs server of time this ticket was issued.
$Lifetime_4$	Prevents replay after ticket has expired.
$Authenticator_c$	Assures server that the ticket presenter is the same as the client for whom the ticket was issued; has very short lifetime to prevent replay.
$K_{c,v}$	Authenticator is encrypted with key known only to client and server, to prevent tampering.
ID_C	Must match ID in ticket to authenticate ticket.
AD_c	Must match address in ticket to authenticate ticket.
TS_5	Informs server of time this authenticator was generated.

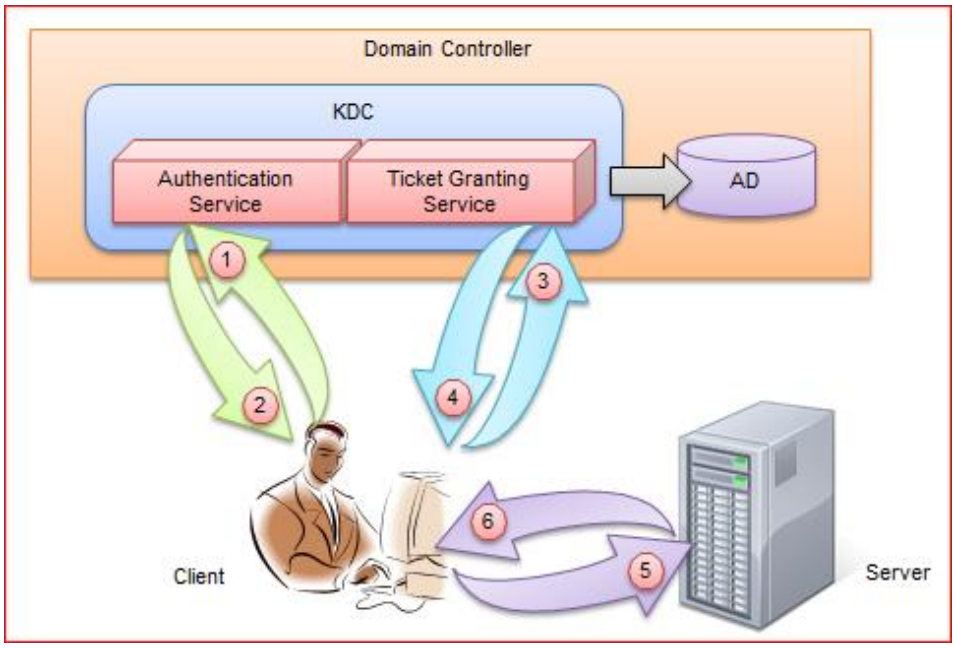
(c) Client/Server Authentication Exchange

Kerberos 简介：

1. 一个安全认证协议
2. 用tickets验证
3. 避免本地保存密码和在互联网上传输密码
4. 包含一个可信任的第三方
5. 使用对称加密
6. 客户端与服务器（非KDC）之间能够相互验证

Kerberos只提供一种功能——在网络上安全的完成用户的身份验证。它并不提供授权功能或者审计功能。

首次请求，三次通信方：
the Authentication Server
the Ticket Granting Server
the Service or host machine that you’re wanting access to.
每次通信，消息包含两部分，一部分可解码，一部分不可解码
服务端不会直接有KDC通信
KDC保存所有机器的账户名和密码
KDC本身具有一个密码



这里以HTTP服务作为例子，如果想要获取HTTP服务，你首先要向KDC表明你自己的身份。这个过程可以在你的程序启动时进行。Kerberos可以通过kinit获取。介绍自己通过未加密的信息发送至KDC获取Ticket Granting Ticket (TGT)。

（1）信息包含：你的用户名/ID,你的IP地址,TGT的有效时间

Authentication Server收到你的请求后，会去数据库中验证，你是否存在。注意，仅仅是验证是否存在，不会验证对错。如果存在，Authentication Server会产生一个随机的Session key(可以是一个64位的字符串)。这个key用于你和Ticket Granting Server (TGS)之间通信。

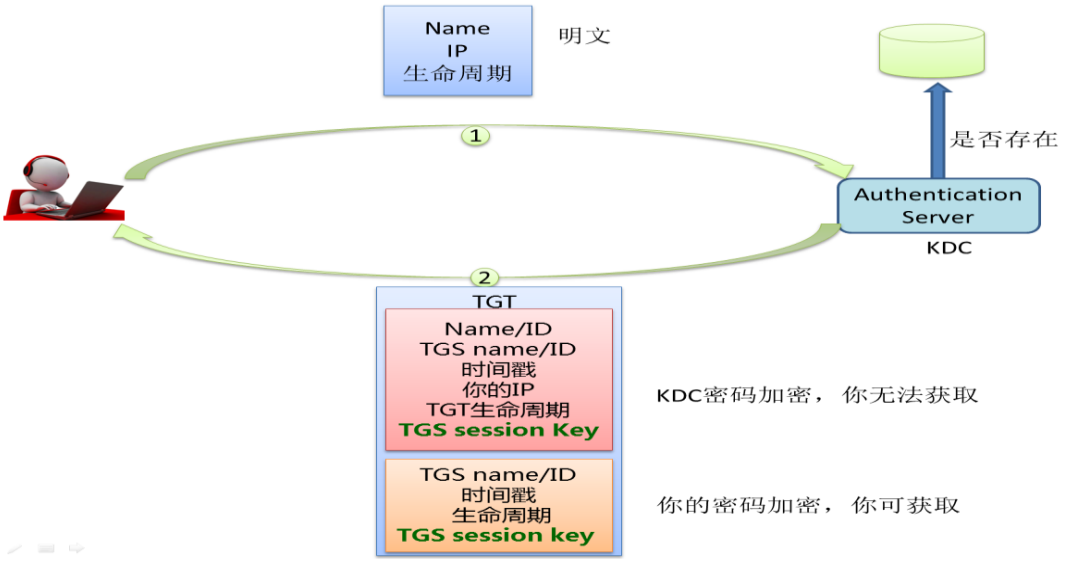
（2）回送信息，Authentication Server同样会发送两部分信息给你，一部分信息为TGT，通过KDC自己的密码进行加密(只能TGS解密)，包含：你的name/ID,TGS的name/ID,时间戳,你的IP地址,TGT的生命周期,TGS session key.

$$Ticket_{tgs} = E(K_{tgs}, [K_{c,tgs} \| ID_C \| AD_C \| ID_{tgs} \| TS_2 \| Lifetime_2])$$

另外一部分通过你的密码(所以只有你能解密)进行加密，包含的信息有:TGS的name/ID,时间戳,生命周期,TGS session key

$$E(K_c, [K_{c,tgs} \| ID_{tgs} \| TS_2 \| Lifetime_2 \| Ticket_{tgs}])$$

如果你的密码是正确的，你就能解密第二部分信息，获取到TGS session key。如果，密码不正确，无法解密，则认证失败。第一部分信息TGT，你是无法解密的，但需要展示缓存起来。在Message1中，你发送了时间戳所以AS知道这是具有时效性的，在Message2中，包含了Client的信息，所以你也知道这是从AS发过来的。



如果第一部分你已经成功，你已经拥有无法解密的TGT和一个TGS Session Key。

（1） 请求信息

a) 通过TGS Session Key加密的认证器部分：你的name/ID,时间戳

$Authenticator_c$ Generated by client to validate ticket .

这部分不会再reuse，只有很短的生存时间，它用来证明了用户的身份（因为它存在很短所以不用担心它被人重用。）

b) 明文传输部分：请求的Http服务名（就是请求信息）,HTTP Service的Ticket生命周期

ID_V Tells TGS that user requests access to server V.

c) TGT部分

$Ticket_{tgs}$ Assures TGS that this user has been authenticated by AS.

这部分并没有证明用户的身份而只是用来分发密钥

Ticket Granting Server收到信息后，首先检查数据库中是否包含有你请求的Http服务名。如果无，直接返回错误信息。如果存在，则通过KDC的密码解密TGT，这个时候。我们就能获取到TGS Session key。然后，通过TGS Session key去解密你传输的第一部分认证器，获取到你的用户名和时间戳。

TGS再进行验证：

1.对比TGT中的用户名与认证器中的用户名

2.比较时间戳，不能超过一定范围

3. 检查是否过期

4. 检查IP地址是否一致

5. 检查认证器是否已在TGS缓存中（避免应答攻击）

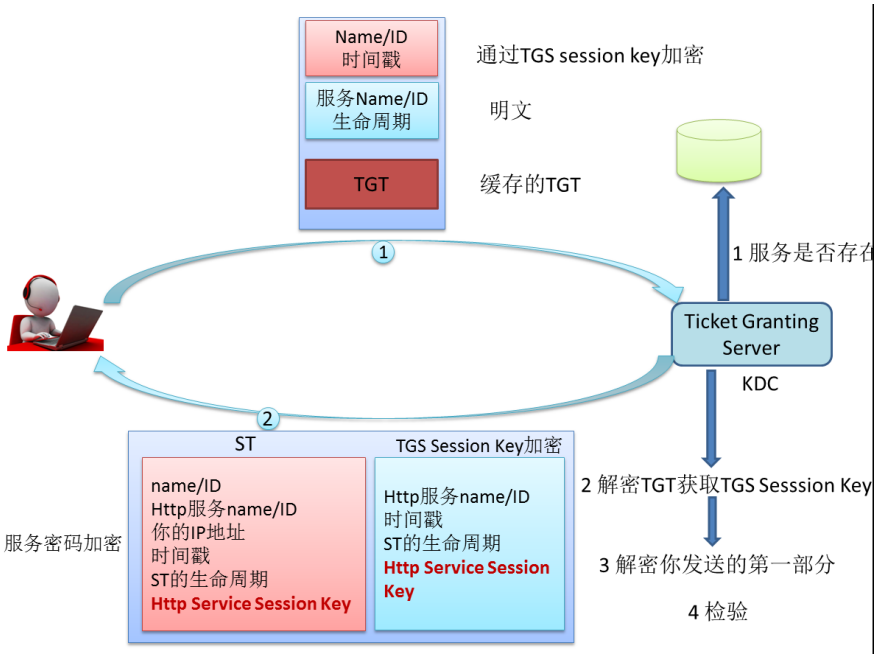
6. 可以在这部分添加权限认证服务：TGS随机产生一个Http Service Session Key，同时准备Http Service Ticket(ST)。

（2） 回答信息

a) 通过Http服务的密码进行加密的信息（ST）：你的name/ID，Http服务name/ID,你的IP地址,时间戳,ST的生命周期，Http Service Session Key

b) 通过TGS Session Key加密的信息:Http服务name/ID,时间戳,ST的生命周期,Http Service Session Key

你收到信息后，通过TGS Session Key解密，获取到了Http Service Session Key，但是你无法解密ST。



在前面两步成功后，以后每次获取Http服务，在Ticket没有过期，或者无更新的情况下，都可直接进行这一步。省略前面两个步骤。

（1） 请求信息

a) 通过Http Service Session Key，加密部分：你的name/ID，时间戳

b) ST

Http服务端通过自己的密码解压ST（KDC是用Http服务的密码加密的），这样就能够获取到Http Service Session Key，解密第一部分。

服务端解密好ST后，进行检查：

1. 对比ST中的用户名（KDC给的）与认证器中的用户名

2. 比较时间戳，不能超过一定范围

3. 检查是否过期

4. 检查IP地址是否一致

5. 检查认证器是否已在HTTP服务端的缓存中（避免应答攻击）

（2） 应答信息

a) 通过Http Service Session Key加密的信息：Http服务name/ID，时间戳

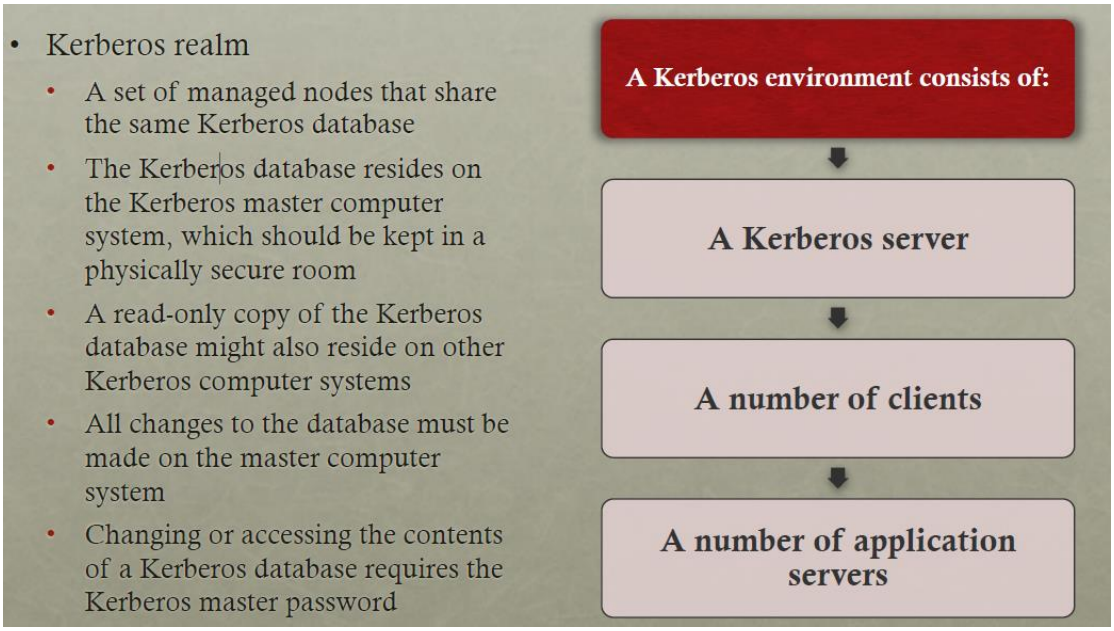
你在通过缓存的Http Service Session Key解密这部分信息，(因此你也可以确定这是服务器发送的消息)然后验证是否是你想要的服务器发送给你的信息。完成你的服务器的验证。

Finally, at the conclusion of this process, the client and server share a secret key. This key can be used to encrypt future messages between the two or to exchange a new random session key for that purpose.

Kerberos Realm

Kerberos Realms and Multiple Kerberi A full-service Kerberos environment consisting of a Kerberos server, a number of clients, and a number of application servers requires the following:

1. The Kerberos server must have the user ID and hashed passwords of all participating users in its database. All users are registered with the Kerberos server.
2. The Kerberos server must share a secret key with each server. All servers are registered with the Kerberos server.



Kerberos领域

•共享同一个kerberos数据库的一组托管节点

•kerberos数据库位于kerberos主计算机系统上，该系统应保存在物理安全的房间中。

•kerberos数据库的只读副本也可能驻留在其他kerberos计算机系统上。

•必须在主计算机系统上对数据库进行所有更改。

•更改或访问kerberos数据库的内容需要kerberos主密码

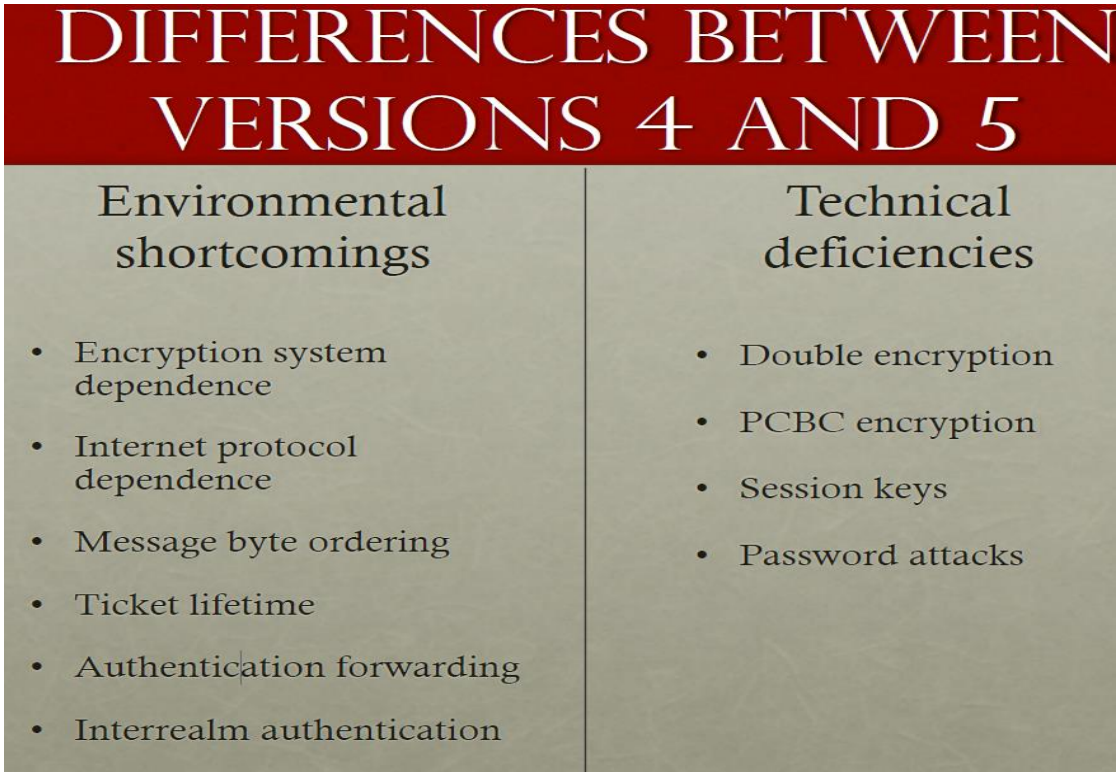


Table 4.3 Summary of Kerberos Version 5 Message Exchanges

(1) C → AS $Options \| IDc \| Realmc \| IDtgs \| Times \| Nonce1$

(2) AS → C $Realmc \| IDC \| Tickettgs \| E(Kc, [Kc,tgs \| Times \| Nonce1 \| Realmtgs \| IDtgs])$

$Tickettgs = E(Ktgs, [Flags \| Kc,tgs \| Realmc \| IDC \| ADC \| Times])$

(a) Authentication Service Exchange to obtain ticket-granting ticket

(3) C → TGS $Options \| IDv \| Times \| \| Nonce2 \| Tickettgs \| Authenticatorc$

(4) TGS → C $Realmc \| IDC \| Ticketv \| E(Kc,tgs, [Kc,v \| Times \| Nonce2 \| Realmv \| IDv])$

$Tickettgs = E(Ktgs, [Flags \| Kc,tgs \| Realmc \| IDC \| ADC \| Times])$

$Ticketv = E(Kv, [Flags \| Kc,v \| Realmc \| IDC \| ADC \| Times])$

$Authenticatorc = E(Kc,tgs, [IDC \| Realmc \| TS1])$

(b) Ticket-Granting Service Exchange to obtain service-granting ticket

(5) C → V $Options \| Ticket_v \| Authenticator_c$

(6) V → C $E_{Kc,v} [TS_2 \| Subkey \| Seq\#]$

$Ticketv = E(Kv, [Flags \| Kc,v \| Realmc \| IDC \| ADC \| Times])$

$Authenticatorc = E(Kc,v, [IDC \| Realmc \| TS2 \| Subkey \| Seq\#])$

(c) Client/Server Authentication Exchange to obtain service

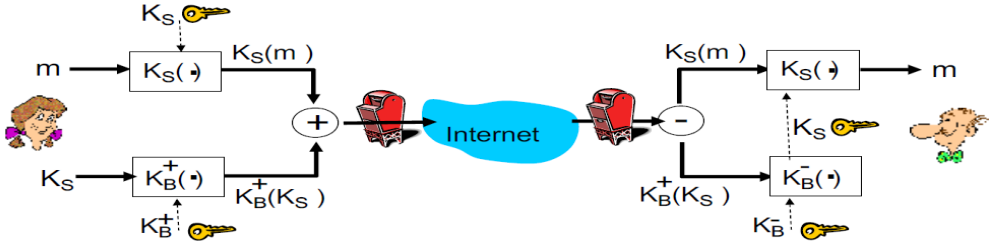
WK03: PGP Secure Socket Layer (SSL) and TLS

Security at Internet Protocol Layers

- Security can be implemented at various layers
- Application Layer Security
  - *A quick glimpse at secure email (PGP)*
- Secure Socket Layer (SSL) / Transport Layer Security (TLS)
  - this week
- Network Layer Security
  - IPSec next week
- Link Layer Security
  - WLAN covered earlier, more on Enterprise later

Secure e-mail: Alice and Bob

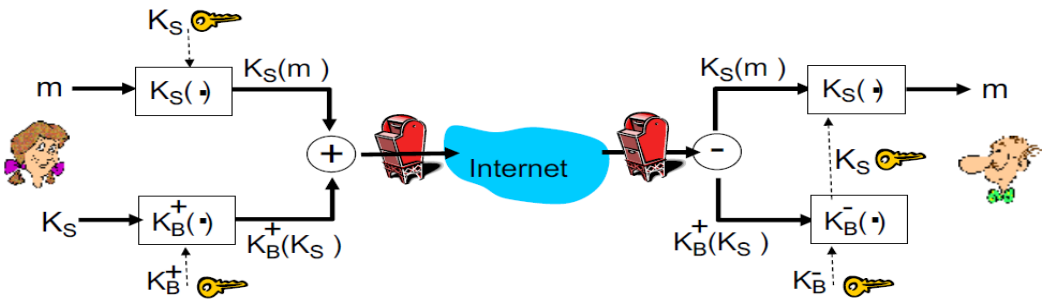
❖ Alice wants to send confidential e-mail, m, to Bob.



Alice:

❖ generates random *symmetric* private key, K_S
❖ encrypts message with K_S (for efficiency)
❖ also encrypts K_S with Bob's public key
❖ sends both $K_S(m)$ and $K_B(K_S)$ to Bob

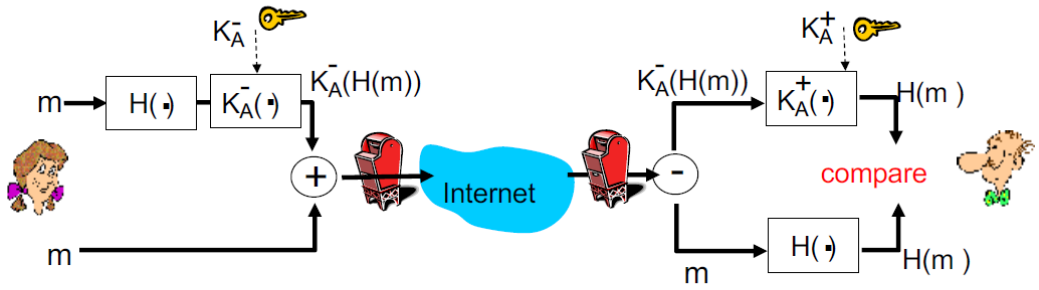
❖ Alice wants to send confidential e-mail, m, to Bob.



Bob:

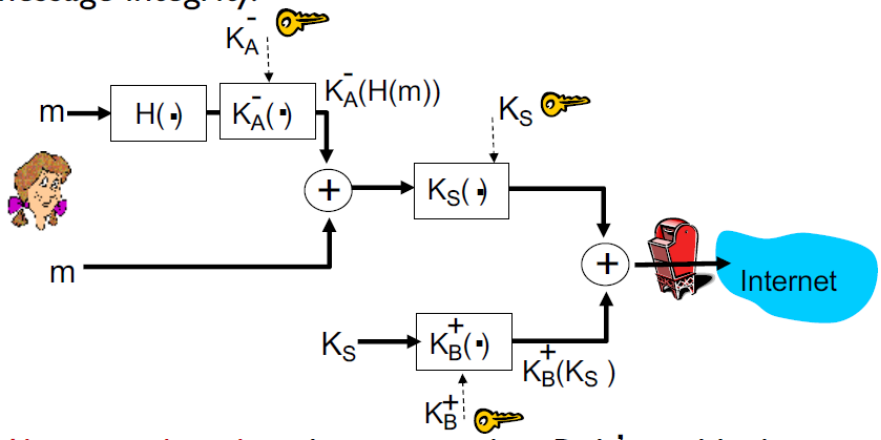
❖ uses his private key to decrypt and recover K_S
❖ uses K_S to decrypt $K_S(m)$ to recover m

- Alice wants to provide sender authentication message integrity



- Alice digitally signs message
- sends both message (in the clear) and digital signature

- Alice wants to provide secrecy, sender authentication, message integrity.



Alice uses three keys: her private key, Bob's public key, newly created symmetric key (Exercise: see how Bob's side works)

DNS Attacks

- Domain Name Service: translation between a host name and the corresponding IP address, maintained by a hierarchy of name servers.
- DoS Attack: A number of Denial of Service (DOS) attacks in recent years
- MiTM: attacker can impersonate a DNS server
  - return a bogus address , divert traffic to a malicious server, allows into collect user passwords or other credentials.
- Cache poisoning attack aka DNS spoofing:
  - attackers to plant bogus addresses, thus diverting a user request to malicious servers.

Impersonate:扮演， divert:转移

想到的第一种针对 DNS 服务的攻击是分布式拒绝服务（DDoS）带宽洪泛攻击（参见 1.6 节）。例如，某攻击者能够试图向每个 DNS 根服务器发送大量的分组，使得大多数合法 DNS 请求得不到回答。这种对 DNS 根服务器的 DDoS 大规模攻击实际发生在 2002 年 10 月 21 日。在这次攻击中，该攻击者利用了一个僵尸网络向 13 个 DNS 根服务器中的每个都发送了大批的 ICMP ping 报文。（第 4 章中讨论了 ICMP 报文。此时，知道 ICMP 分组是特殊类型的 IP 数据报就可以了。）幸运的是，这种大规模攻击所带来的损害很小，对用户的因特网体验几乎没有或根本没有影响。攻击者的确成功地将大量的分组指向了根服务器，但许多 DNS 根服务器受到了分组过滤器的保护，配置的分组过滤器阻挡了所有指向根服务器的 ICMP ping 报文。这些被保护的服务器因此未受伤害并且与平常一样发挥着作用。此外，大多数本地 DNS 服务器缓存了顶级域名服务器的 IP 地址，使得这些请求过程通常绕过了 DNS 根服务器。

对 DNS 的潜在更为有效的 DDoS 攻击将是向顶级域名服务器（例如向所有处理.com 域的顶级域名服务器）发送大量的 DNS 请求。过滤指向 DNS 服务器的 DNS 请求将更为困难，并且顶级域名服务器不像根服务器那样容易绕过。但是这种攻击的严重性通过本地 DNS 服务器中的缓存技术可将部分地被缓解。

DNS 能够潜在地以其他方式被攻击。在中间人攻击中，攻击者截获来自主机的请求并返回伪造的回答。在 DNS 毒害攻击中，攻击者向一台 DNS 服务器发送伪造的回答，诱使服务器在它的缓存中接收伪造的记录。这些攻击中的任一种，都能够将满怀信任的 Web 用户重定向到攻击者的 Web 站点。然而，这些攻击难以实现，因为它们要求截获分组或扼制住服务器［Skoudis 2006］。

另一种重要的 DNS 攻击本质上并不是一种对 DNS 服务的攻击，而是充分利用 DNS 基础设施来对目标主机发起 DDoS 攻击（例如，你所在大学的邮件服务器）。在这种攻击中，攻击者向许多权威 DNS 服务器发送 DNS 请求，每个请求具有目标主机的假冒源地址。这些 DNS 服务器则直接向目标主机发送它们的回答。如果这些请求能够精心制作成下述方式的话，即响应比请求（字节数）大得多（所谓放大），则攻击者不必自行产生大量的流量就有可能淹没目标主机。这种利用 DNS 的反射攻击至今为止只取得了有限的成功［Mirkovic 2005］。

总而言之，DNS 自身已经显示了对抗攻击的令人惊讶的健壮性。至今为止，还没有一个攻击已经成功地妨碍了 DNS 服务。已经有了成功的反射攻击；然而，通过适当地配置 DNS 服务器，能够处理（和正在处理）这些攻击。

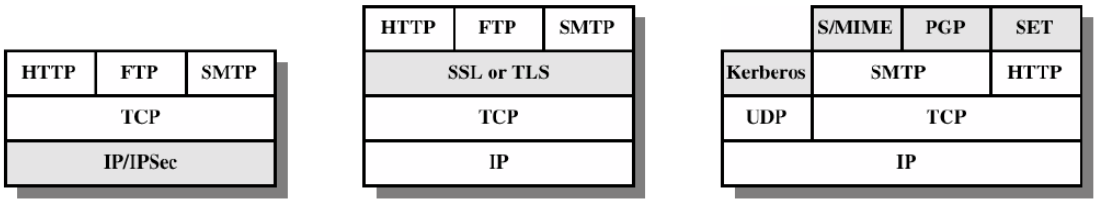
DNS Security

- IETF DNS Security Extensions (DNSSEC)
- Approach similar to PGP
  - response signed by the private key of a DNS server.
  - a requester can verify using the corresponding public key.
  - a digital signature also provides the integrity of the response data, Confidentiality not an issue for DNS records
- Recent study : only 1% of domains use the DNSSEC
- Very few registrars support DNSSEC .
- Mechanism for communicating DNSSEC information has several security vulnerabilities.
- DDoS defence in not part of DNSSEC
  - Intrusion Detection/Prevention System for DOS (later week)

SSL: Secure Sockets Layer

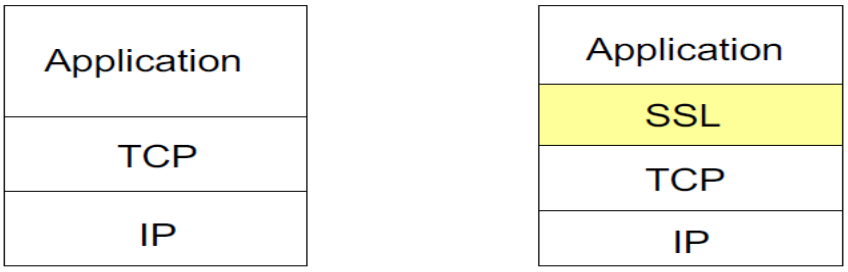
- widely deployed security protocol
  - supported by almost all browsers, web servers
  - https
  - billions $/year over SSL
- mechanisms: [Woo 1994], implementation: Netscape
- variation -TLS: transport layer security, RFC 2246
- provides
  - *Confidentiality, integrity, authentication*
- original goals:
  - Web e-commerce transactions
  - encryption (especially credit-card numbers
  - Web-server authentication
  - optional client authentication
  - minimum hassle in doing business with new merchant
- available to all TCP applications
  - secure socket interface

安全协议的分层应用：



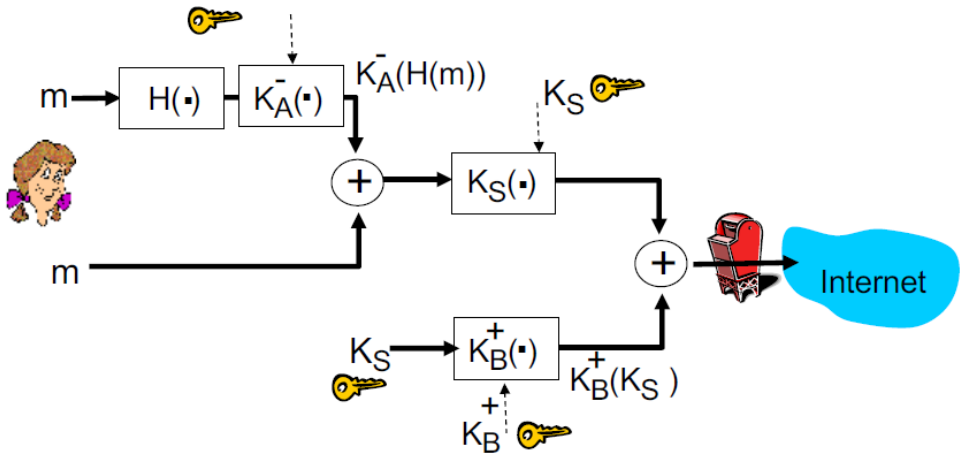
(a) Network Level (b) Transport Level (c) Application Level

SSL and TCP/IP



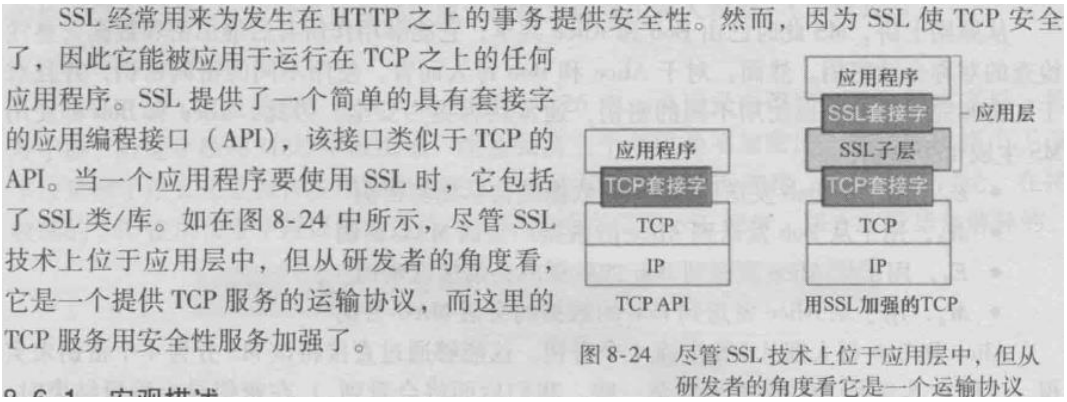
normal application *application with SSL*

SSL provides application programming interface (API) to applications
C and Java SSL libraries/classes readily available



- but want to send byte streams & interactive data
- want set of secret keys for entire connection
- want certificate exchange as part of protocol: handshake phase

SSL: a simple secure channel



SSL 经常用来为发生在 HTTP 之上的事务提供安全性。然而，因为 SSL 使 TCP 安全了，因此它能被应用于运行在 TCP 之上的任何应用程序。SSL 提供了一个简单的具有套接字的应用编程接口（API），该接口类似于 TCP 的 API。当一个应用程序要使用 SSL 时，它包括了 SSL 类/库。如在图 8-24 中所示，尽管 SSL 技术上位于应用层中，但从研发者的角度看，它是一个提供 TCP 服务的运输协议，而这里的 TCP 服务用安全性服务加强了。

图 8-24 尽管 SSL 技术上位于应用层中，但从研发者的角度看它是一个运输协议

handshake: Alice and Bob use their certificates, private keys to authenticate each other and exchange shared secret

key derivation: Alice and Bob use shared secret to derive set of keys

data transfer: data to be transferred is broken up into series of records

connection closure: special messages to securely close connection

A simple handshake

这是一个宏观的 SSL 连接过程，所以称为类 SSL。具体的连接步骤在后面 procedure 部分

在握手阶段，Bob 需要：①与 Alice 创建一条 TCP 连接；②验证 Alice 是真实的 Alice；③发送给 Alice 一个主密钥，Bob 和 Alice 持用该主密钥生成 SSL 会话所需的所有对称密钥。这三个步骤显示在图 8-25 中。注意到一旦创建了 TCP 连接，Bob 就向 Alice 发送一个 hello 报文。Alice 则用她的证书进行响应，证书中包含了她的公钥。如在 8.3 节所讨论，因为该证书已被某 CA 证实过，Bob 明白无误地知道该公钥属于 Alice。然后，Bob 产生一个主密钥（MS）（该 MS 将仅用于这个 SSL 会话），用 Alice 的公钥加密该 MS 以生成加密的主密钥（EMS），并将该 EMS 发送给 Alice。Alice 用她的私钥解密该 EMS 从而得到该 MS。在这个阶段后，Bob 和 Alice（而无别的人）均知道了用于这次 SSL 会话的主密钥。

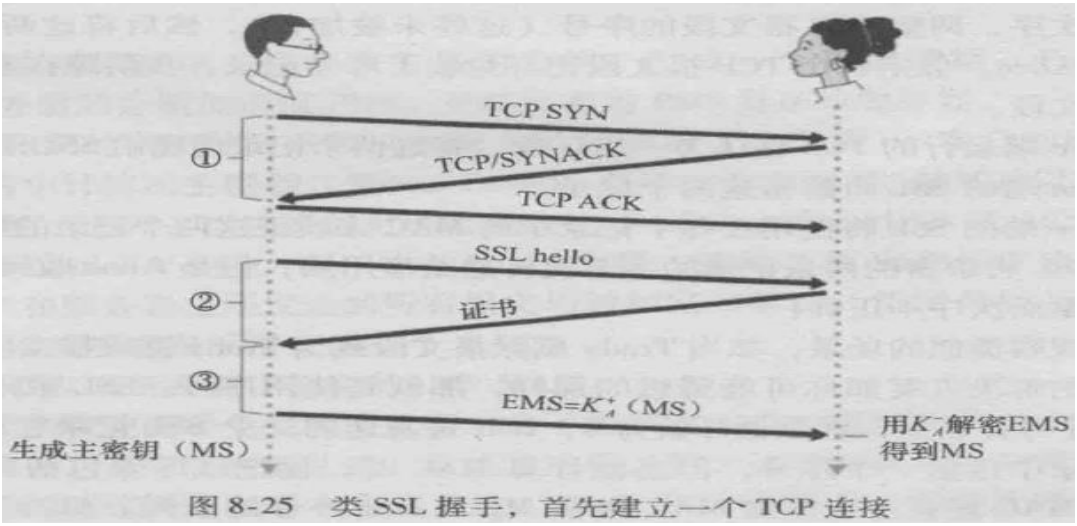
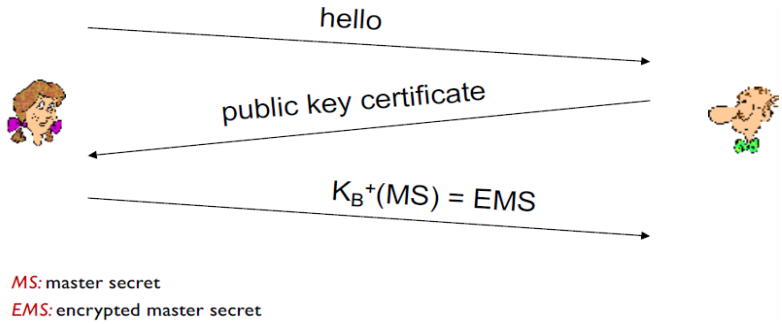


图 8-25　类 SSL 握手，首先建立一个 TCP 连接



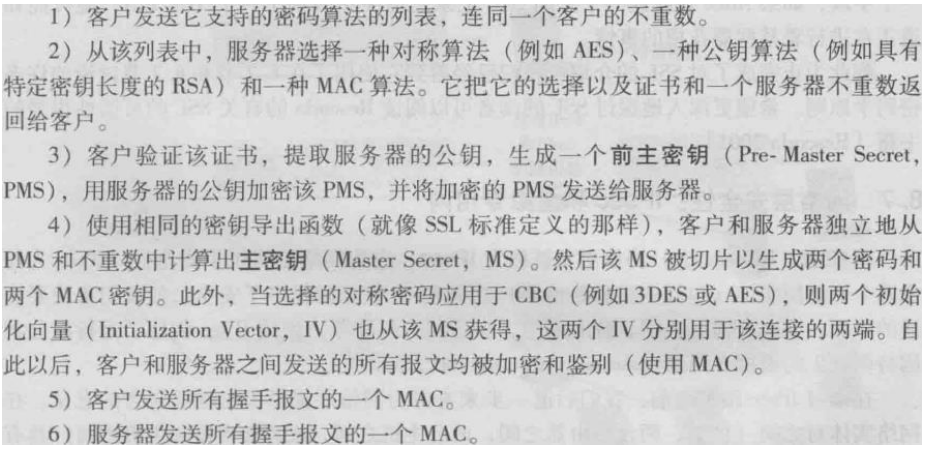
MS: master secret
EMS: encrypted master secret

Purpose

1. server authentication
2. negotiation(协商): agree on crypto algorithms
3. establish keys
4. client authentication (optional)

Procedure:

1. client sends list of algorithms it supports, along with client nonce
2. server chooses algorithms from list; sends back: choice + certificate + server nonce
3. client verifies certificate, extracts server's public key, generates pre_master_secret, encrypts with server's public key, sends to server
4. client and server independently compute encryption and MAC keys from pre_master_secret and nonces
5. client sends a MAC of all the handshake messages
6. server sends a MAC of all the handshake messages



1）客户发送它支持的密码算法的列表，连同一个客户的不重数。

2）从该列表中，服务器选择一种对称算法（例如 AES）、一种公钥算法（例如具有特定密钥长度的 RSA）和一种 MAC 算法。它把它的选择以及证书和一个服务器不重数返回给客户。

3）客户验证该证书，提取服务器的公钥，生成一个**前主密钥**（Pre- Master Secret，PMS），用服务器的公钥加密该 PMS，并将加密的 PMS 发送给服务器。

4）使用相同的密钥导出函数（就像 SSL 标准定义的那样），客户和服务器独立地从 PMS 和不重数中计算出**主密钥**（Master Secret，MS）。然后该 MS 被切片以生成两个密码和两个 MAC 密钥。此外，当选择的对称密码应用于 CBC（例如 3DES 或 AES），则两个初始化向量（Initialization Vector，IV）也从该 MS 获得，这两个 IV 分别用于该连接的两端。自此以后，客户和服务器之间发送的所有报文均被加密和鉴别（使用 MAC）。

5）客户发送所有握手报文的一个 MAC。

6）服务器发送所有握手报文的一个 MAC。

Why last two messages with MAC exchanged?

- client typically offers range of algorithms, some strong, some weak
- man-in-the middle could delete stronger algorithms from list
- last 2 steps prevent this
  - last two messages are encrypted

Why two random nonces?

- suppose Trudy sniffs all messages between Alice & Bob
- next day, Trudy sets up TCP connection with Bob, sends exact same sequence of records
  - Bob (Amazon) thinks Alice made two separate orders for the same thing
  - solution: Bob sends different random nonce for each connection. This causes encryption keys to be different on the two days
  - Trudy's messages will fail Bob's integrity check

你可能想知道在步骤 1 和步骤 2 中存在不重数的原因。序号不足以防止报文段重放攻击吗？答案是肯定的，但它们并不只是防止“连接重放攻击”。考虑下列连接重放攻击。假设 Trudy 嗅探了 Alice 和 Bob 之间的所有报文。第二天，Trudy 冒充 Bob 并向 Alice 发送正好是前一天 Bob 向 Alice 发送的相同的报文序列。如果 Alice 没有使用不重数，她将以前一天发送的完全相同的序列报文进行响应。Alice 将不怀疑任何不规矩的事，因为她接收到的每个报文将通过完整性检查。如果 Alice 是一个电子商务服务器，她将认为 Bob 正在进行第二次订购（正好订购相同的东西）。在另一方面，在协议中包括了一个不重数，Alice 将对每个 TCP 会话发送不同的不重数，使得这两天的加密密钥不同。因此，当 Alice 接收到来自 Trudy 重放的 SSL 记录时，该记录将无法通过完整性检查，并且假冒的电子商务事务将不会成功。总而言之，在 SSL 中，不重数用于防御“连接重放”，而序号用于防御在一个进行中的会话中重放个别分组。

Key derivation

considered bad to use same key for more than one cryptographic operation

- use different keys for message authentication code (MAC) and encryption

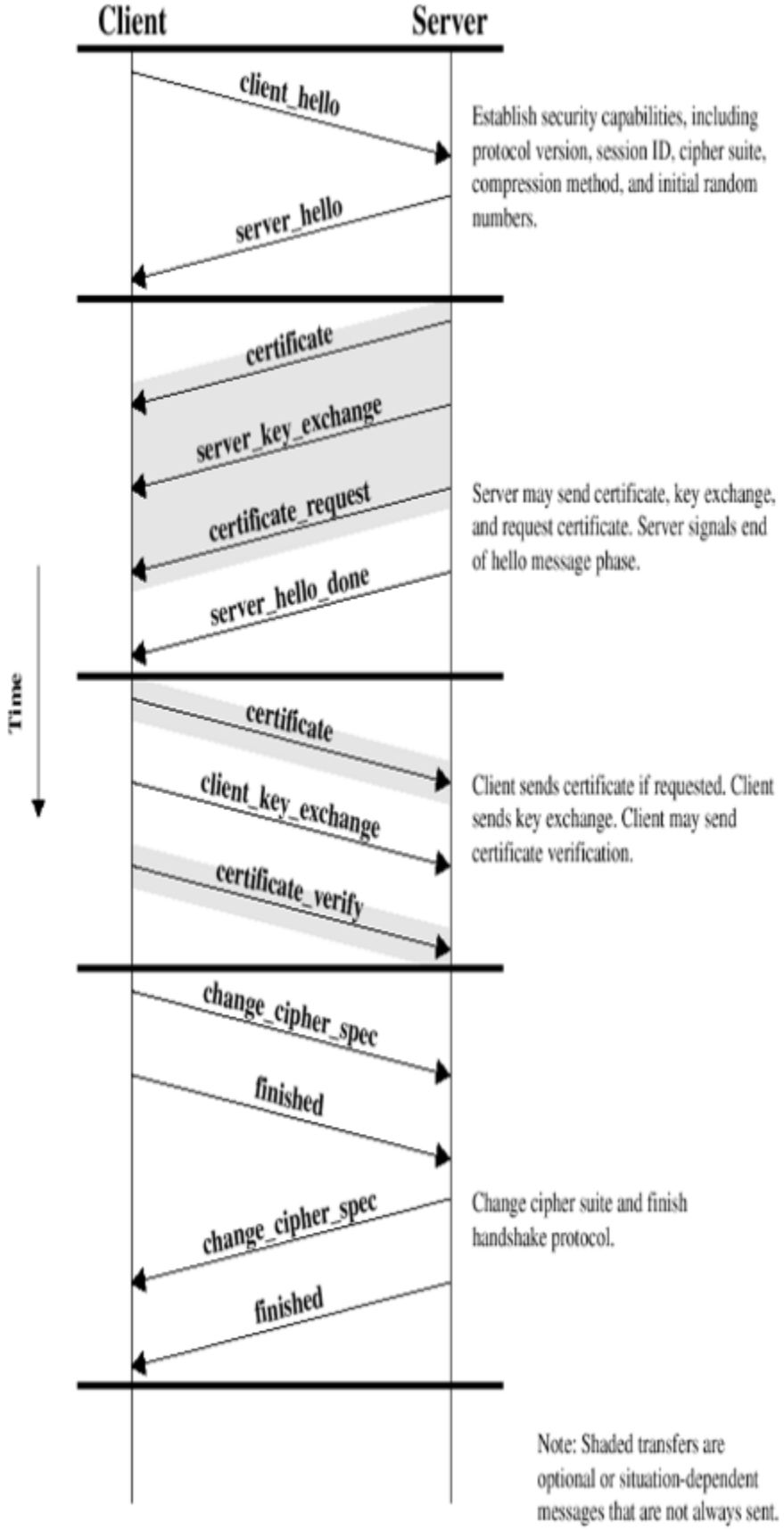
four keys:

- K_c = encryption key for data sent from client to server
- M_c = MAC key for data sent from client to server
- K_s = encryption key for data sent from server to client
- M_s = MAC key for data sent from server to client

keys derived from key derivation function (KDF)

- takes master secret and (possibly) some additional random data and creates the keys

更细节的握手过程如下：


Client
Server
client_hello
Establish security capabilities, including protocol version, session ID, cipher suite, compression method, and initial random numbers.
server_hello
certificate
server_key_exchange
certificate_request
Server may send certificate, key exchange, and request certificate. Server signals end of hello message phase.
server_hello_done
Time
certificate
client_key_exchange
Client sends certificate if requested. Client sends key exchange. Client may send certificate verification.
certificate_verify
change_cipher_spec
finished
change_cipher_spec
Change cipher suite and finish handshake protocol.
finished
Note: Shaded transfers are optional or situation-dependent messages that are not always sent.

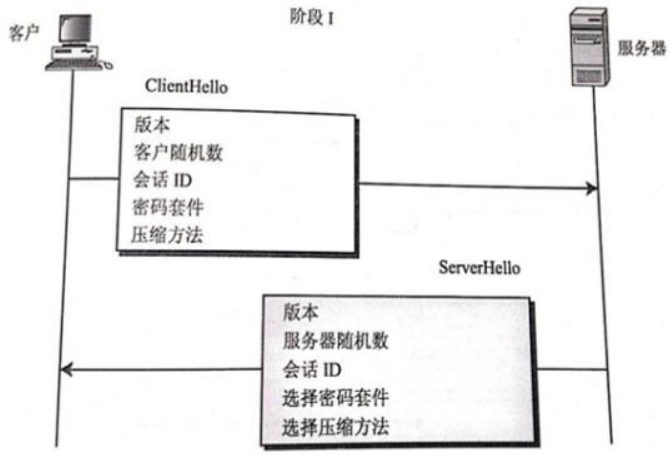


图 17-14 握手协议的阶段 I

ClientHello 包含的内容（ServerHello 是对应的差不多的内容）：

（1）客户端可以支持的 SSL 最高版本号

（2）一个用于生成主秘密的 32 字节的随机数。

（3）一个确定会话的会话 ID。

（4）一个客户端可以支持的密码套件列表。

密码套件格式：每个套件都以“SSL”开头，紧跟着的是密钥交换算法。用“With”这个词把密钥交换算法、加密算法、散列算法分开，例如：SSL_DHE_RSA_WITH_DES_CBC_SHA,表示把 DHE_RSA(带有 RSA 数字签名的暂时 Diffie-HellMan)定义为密钥交换算法；把 DES_CBC 定义为加密算法；把 SHA 定义为散列算法。

（5）一个客户端可以支持的压缩算法列表。

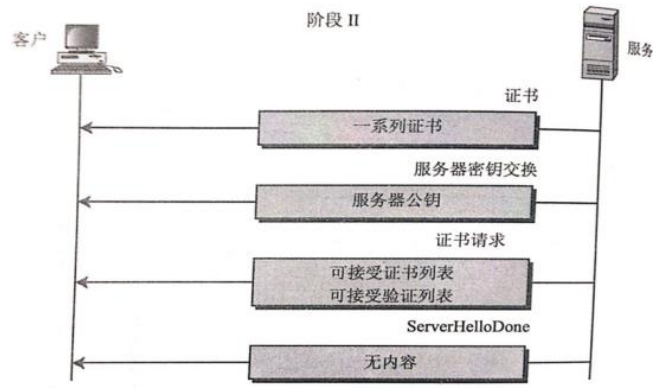


图 17-15 握手协议的阶段 II

服务器启动 SSL 握手第 2 阶段，是本阶段所有消息的唯一发送方，客户机是所有消息的唯一接收方。该阶段分为 4 步：

（a）证书：服务器将数字证书和到根 CA 整个链发给客户端，使客户端能用服务器证书中的服务器公钥认证服务器。

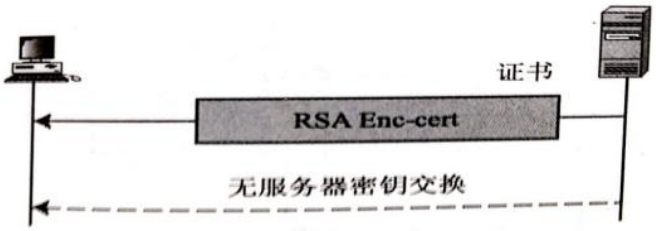
（b）服务器密钥交换（可选）：这里视密钥交换算法而定

（c）证书请求：服务端可能会要求客户自身进行验证。

（d）服务器握手完成：第二阶段的结束，第三阶段开始的信号

这里重点介绍一下服务端的验证和密钥交换。这个阶段的前面的（a）证书 和（b）服务器密钥交换是基于密钥交换方法的。而在 SSL 中密钥交换算法有 6 种：无效（没有密钥交换）、RSA、匿名 DiffiHellman、

暂时 Diffie-Hellman、固定 Diffie-Hellman、Fortezza。在阶段 1 过程客户端与服务端协商的过程中已经确定使哪种密钥交换算法。如果协商过程中确定使用 RSA 交换密钥，那么过程如下图：



a. RSA 证书

这个方法中，服务器在它的第一个信息中，发送了 RSA 加密/解密公钥证书。不过，因为预备主秘密是由客户端在下一个阶段生成并发送的，所以第二个信息是空的。注意，公钥证书会进行从服务器到客户端的验证。当服务器收到预备主秘密时，它使用私钥进行解密。服务端拥有私钥是一个证据，可以证明服务器是一个它在第一个信息发送的公钥证书中要求的实体。

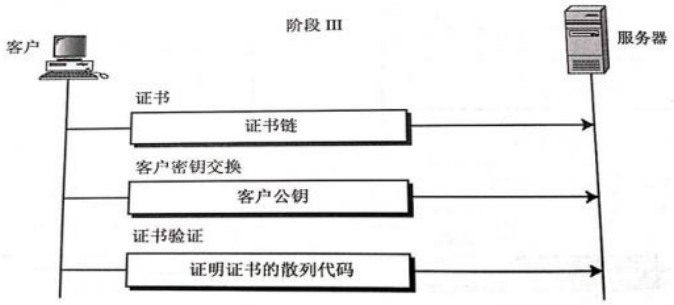
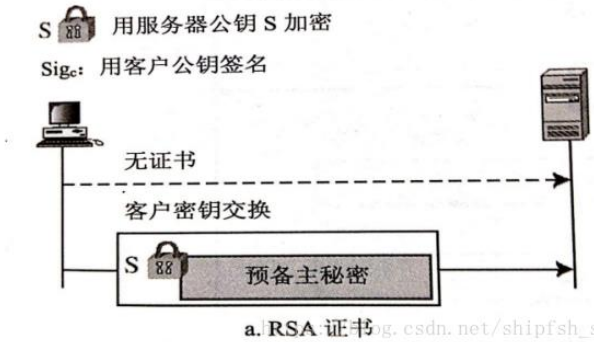


图 17-17　握手协议的阶段 III

客户机启动 SSL 握手第 3 阶段，是本阶段所有消息的唯一发送方，服务器是所有消息的唯一接收方。该阶段分为 3 步：（a）证书（可选）：为了对服务器证明自身，客户要发送一个证书信息，这是可选的，在 IIS 中可以配置强制客户端证书认证。（b）客户机密钥交换（Pre-master-secret）：这里客户端将预备主密钥发送给服务端，注意这里会使用服务端的公钥进行加密。（c）证书验证（可选），对预备秘密和随机数进行签名，证明拥有（a）证书的公钥

下面也重点介绍一下 RSA 方式的客户端验证和密钥交换。



a. RSA 证书

这种情况，除非服务器在阶段 II 明确请求，否则没有证书信息。客户端密钥交换方法包括阶段 II 收到的由 RSA 公钥加密的预备主密钥。阶段 III 之后，客户要有服务器进行验证，客户和服务器都知道预备主密钥。

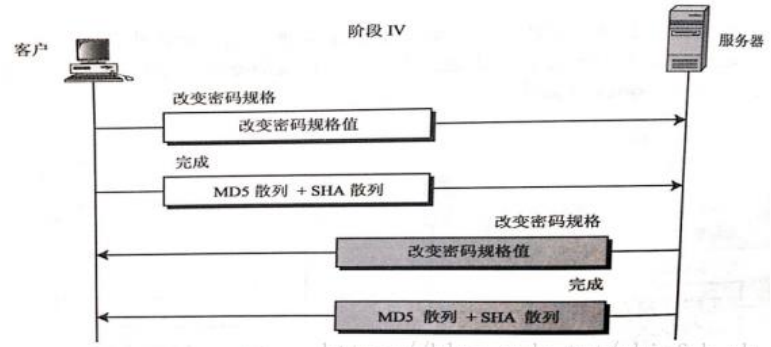


图 17-19 握手协议的阶段 IV

这样握手协议完成，下面看下什么是预备主密钥，主密钥是怎么生成的。为了保证信息的完整性和机密性，SSL 需要有六个加密秘密：四个密钥和两个 IV。为了信息的可信性，客户端需要一个密钥（HMAC），为了加密要有一个密钥，为了分组加密要一个 IV，服务也是如此。SSL 需要的密钥是单向的，不同于那些在其他方向的密钥。如果在一个方向上有攻击，这种攻击在其他方向是没影响的。生成过程如下：

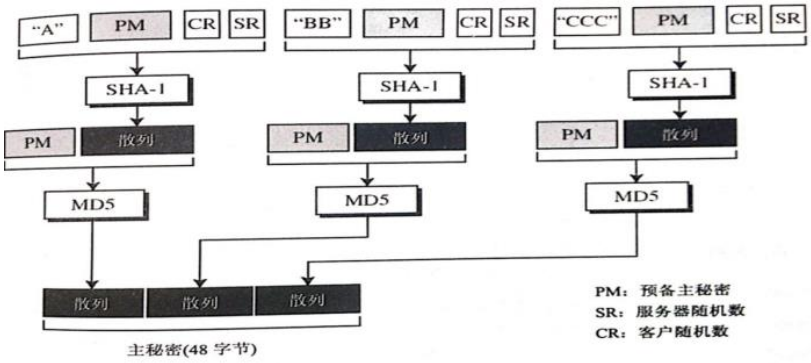


图 17-8 从预备主秘密计算主秘密

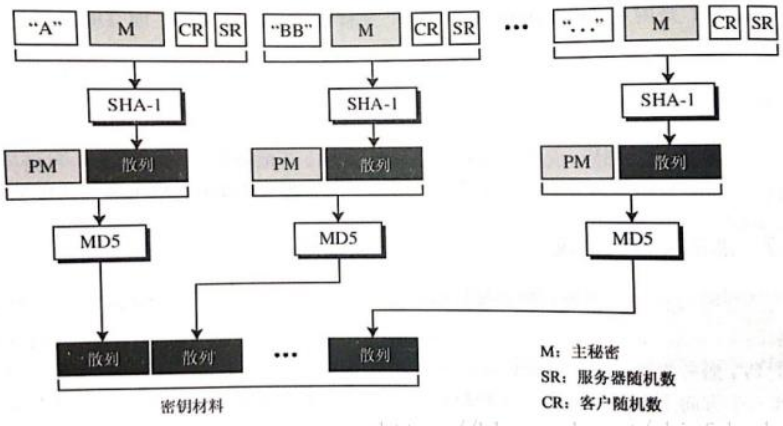


图 17-9 从主秘密计算密钥材料

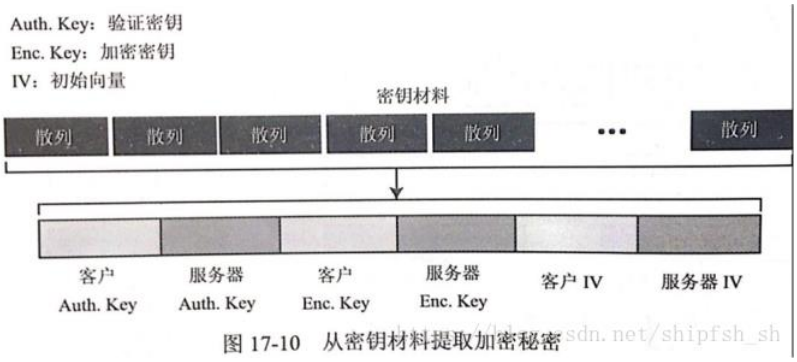


图 17-10 从密钥材料提取加密秘密

Data records

why not encrypt data in constant stream as we write it to TCP?

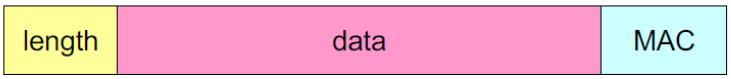
- where would we put the MAC? If at end of TCP connection, no message integrity until all data processed.
- e.g., with instant messaging, how can we do integrity check over all bytes sent before displaying?

instead, break stream in series of records

- each record carries a MAC
- receiver can act on each record as it arrives

issue: in record, receiver needs to distinguish MAC from data

- want to use variable-length records



为什么不在将数据写入TCP时加密常量流中的数据？

–Mac放在哪里？如果在TCP连接结束时，在处理所有数据之前没有消息完整性。

–例如，对于即时消息，我们如何在显示前对发送的所有字节进行完整性检查？

•相反，在一系列记录中中断流

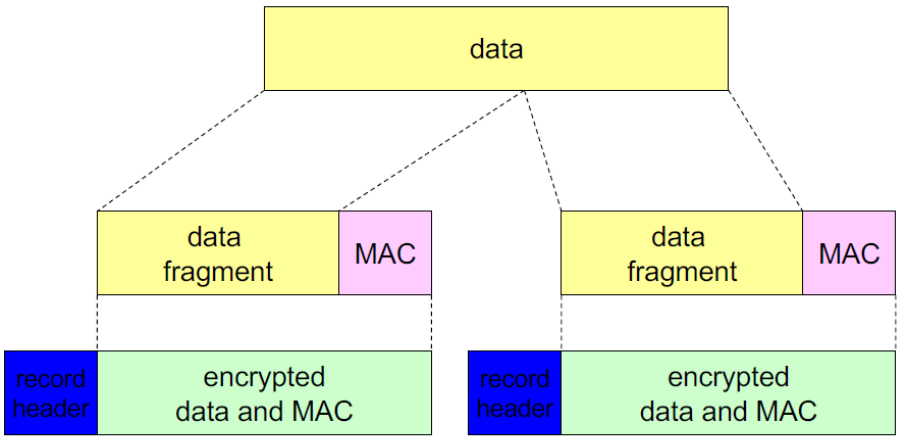
–每条记录都有一个Mac

–接收者可以在每条记录到达时对其进行操作。

•问题：在记录中，接收器需要区分MAC和数据

–要使用可变长度记录

SSL record protocol

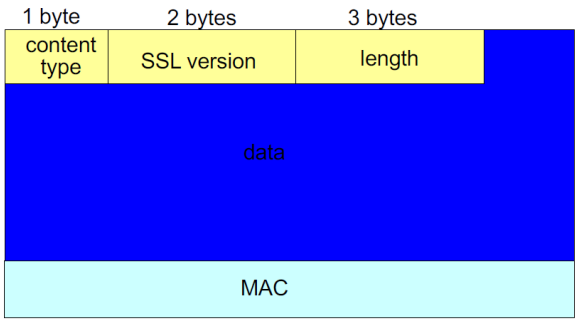


record header: content type; version; length

MAC: includes sequence number, MAC key M_x

fragment: each SSL fragment 2^{14} bytes (~16 Kbytes)

SSL record format



data and MAC encrypted (symmetric algorithm)

Sequence numbers

- *problem:* attacker can capture and replay record or re-order records
- *solution:* put sequence number into MAC:
  - MAC = MAC(M_x, sequence||data)
  - *note: no sequence number field*

- *problem:* attacker could replay all records in future
- *solution:* use nonce

对该问题的解决方案如你可能猜想的那样，那就是使用序号。SSL 采用如下的方式。Bob 维护一个序号计数器，计数器开始为 0，Bob 每发送的一个 SSL 记录它都增加 1。Bob 并不实际在记录中包括一个序号，但当他计算 MAC 时，他把该序号包括在 MAC 的计算中。所以，该 MAC 现在是数据加 MAC 密钥 M_B 加当前序号的散列。Alice 跟踪 Bob 的序号，通过在 MAC 的计算中包括适当的序号，使她验证一条记录的数据完整性。SSL 序号的使用阻止了 Trudy 执行诸如重排序或重放报文段等中间人攻击。（为什么？）

Control information

problem: truncation attack:

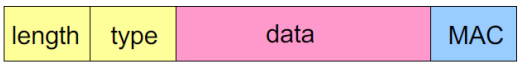
– attacker forges TCP connection close segment

– one or both sides thinks there is less data than there actually is.

solution: record types, with one type for closure

– type 0 for data; type 1 for closure

MAC = MAC(M_x, sequence||type||data)



Truncation:截断； forge:伪造

SSL Architecture

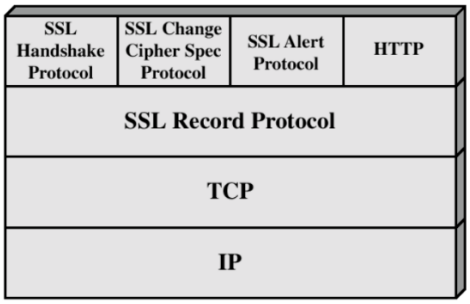
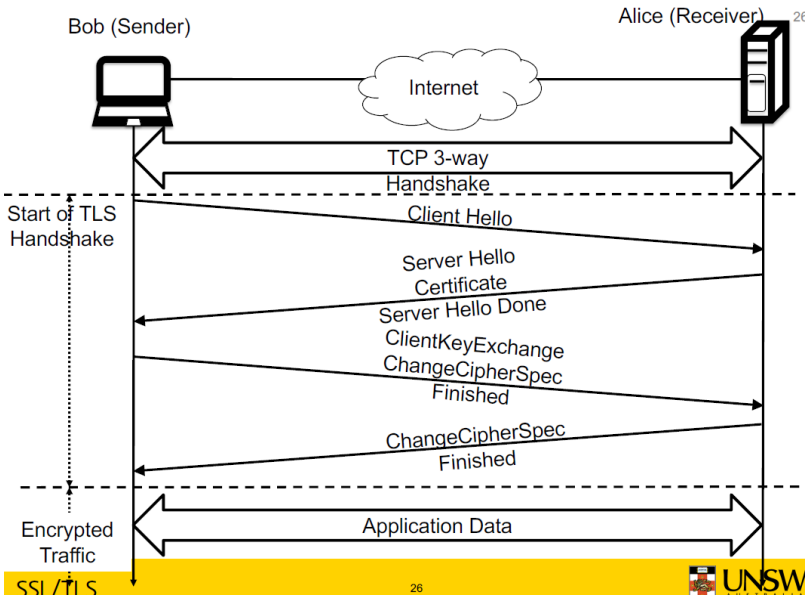


Figure 7.2 SSL Protocol Stack

TLS (Transport Layer Security)

The same record format as the SSL record format.
Defined in RFC 2246.
Similar to SSLv3.
Differences in the:

- version number
- message authentication code
- pseudorandom function
- alert codes
- cipher suites
- client certificate types
- certificate_verify and finished message
- cryptographic computations
- padding



TLS的说明：IETF将SSL标准化，即RFC 2246，并将其称为TLS（Transport Layer Security）。从技术上讲，TLS 1.0与SSL 3.0的差异非常微小。但正如RFC所述"the differences between this protocol and SSL 3.0 are not dramatic, but they are significant enough to preclude interoperability between TLS 1.0 and SSL 3.0"（本协议和SSL 3.0之间的差异并不是显著，却足以排除TLS 1.0和SSL 3.0之间的互操作性）。TLS 1.0包括可以降级到SSL 3.0的实现，这削弱了连接的安全性。

Handshake

- First Bob and Alice exchange the three-way TCP SYN, SYNACK and ACK messages.
- Bob then sends a ClientHello message to Alice along with the cipher suites (ciphers and thehash functions it supports) and a nonce, a large, random number, chosen specifically for this run of the protocol.
- Alice responds with a ServerHello message along with her choice from the cipher suites (e.g.,AES for confidentiality, RSA for the public key, SHA2 for the Message Authentication Code (MAC)), a certificate containing her public key and a nonce. Additionally, she could also request the client's certificate and parameters for other TLS extensions.
- Bob checks validity of the certificate and is assured that it belongs to Alice. He initiates theClientKeyExchange message. This can use a range of key exchange methods, e.g., RSA orthe Diffie-Hellman (and variants) to establish a symmetric key for the ensuing session.
  - For example, when using RSA, Bob could generate a 48-bit Pre-master secret (PMS) and encryptit with Alice's public key obtained using the steps as described above and send it to Alice.
- Bob sends a ClientCiptherSpec and a Finished Message suggesting that the key generation and authentication are complete.
- Alice also has the shared key at this point. She responds with a ChangeCipherSpec and a Finished Message back to Bob.
- Bob decrypts the message with the negotiated symmetric key and performs a message integrity check

Key derivation

- client nonce, server nonce, and pre-master secret input into pseudo random-number generator.
  - produces master secret
- master secret and new nonces input into another random-number generator: "key block"

- key block sliced and diced:
  - client MAC key
  - server MAC key
  - client encryption key
  - server encryption key
  - client initialization vector (IV)
  - server initialization vector (IV)