COMP3020J Information security of IoT

1.Cryptography:

Cipher/cryptosystem是用来加密Plaintext的，加密完变成ciphertext，解密的过程叫decrypt. 用到的key相当于是这个cryptosystem的configuration



我们认为整个系统对于attacker是已知的，只有key是未知的，即加密算法(Kerckhoffs’ Principle). 因此一个cryptosystem必须在整个系统已知的情况下保持安全.

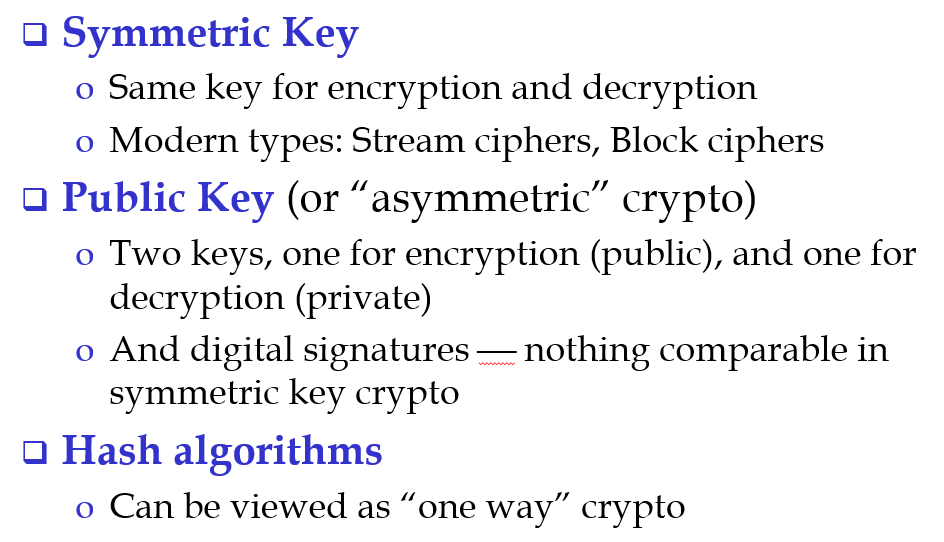
Cryptosystem is secure if best known attack is to try all keys

Cryptosystem is insecure if any shortcut attack is known

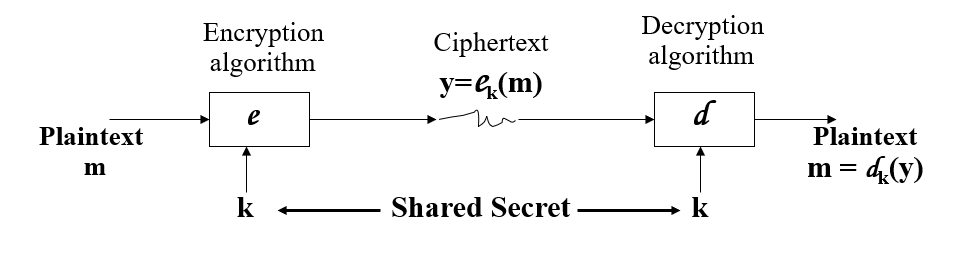
但是有时候破解secure的cryptosystem比insecure的还要快(特殊情况)

最初的加密方法：shift几位：Caesar’s Cipher(上移3位)---过于简单，可以穷举

现在的加密方法：



Symmetric Encryption:



在对称加密中发送方和接收方使用同一个pre-shared key

分为两种:Stream Cipher & Block Cipher

Stream Cipher: 相当于generalized one-time-pad

Key相对较短，stretched into keystream，把这个keystream当成one-time-pad使用

没有使用confusion和diffusion，因此对statistical analysis的抗性会较差

Block Cipher: 相当于generalized codebook

Block Cipher key决定了codebook，每个key对应不同的codebook，并且使用到了confusion和diffusion

使用到了Confusion和Diffusion（为了阻碍statistical analysis）

* **Confusion** means that the key does not relate in a simple way to the ciphertext. In particular, each character of the ciphertext should depend on several parts of the key. ---每个ciphertext的character应该由key中的好几部分决定
* **Diffusion** means that if we change a character of the plaintext, then several characters of the ciphertext should change, and similarly, if we change a character of the ciphertext, then several characters of the plaintext should change. i.e. frequency statistics of letters, [digraphs], etc. in the plaintext are diffused over several characters in the ciphertext, which means that much more ciphertext is needed to do a meaningful statistical attack.---当ciphertext或者plaintext中的一位改变时，对应的另一种text中应该有很多位改变。

Stream Cipher：

每次对一个bit或者一个Byte operate，作为stream

Stream Cipher就是plaintext和keystream的简单combine。一般用XOR来进行combine

用途：用于大量未知长度的plaintext，比如secure wireless connection

缺点：需要和message一样多的bits，difficult in practice

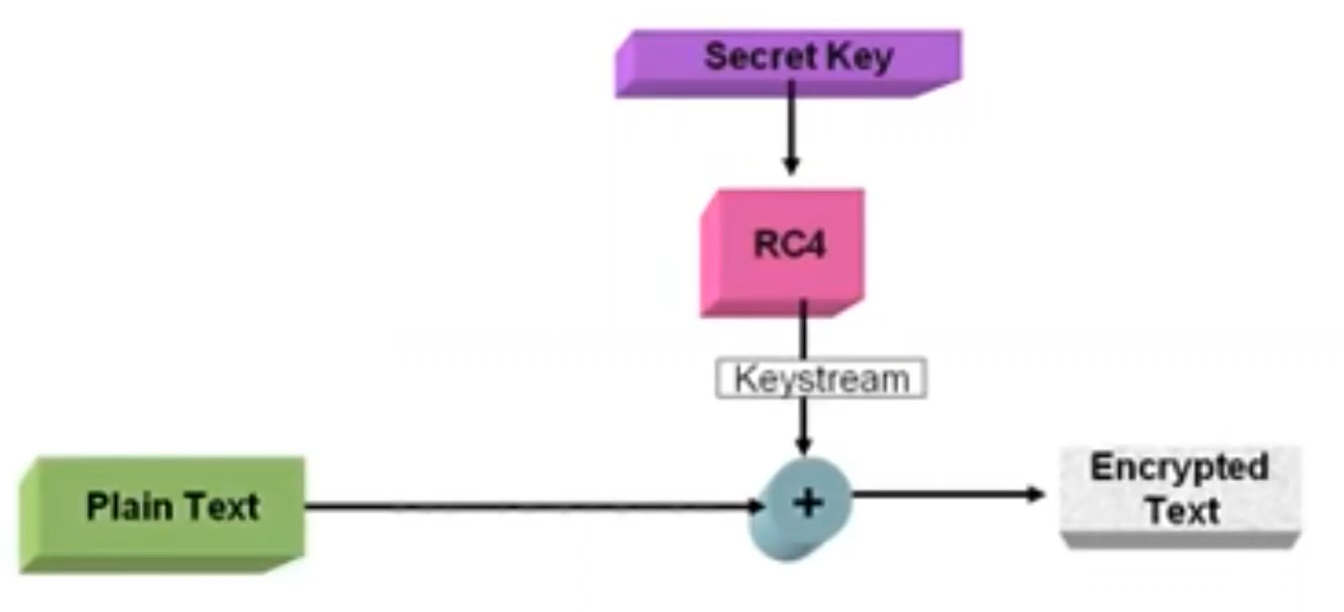
如果key是完全随机的，则是secure的

几种实现：

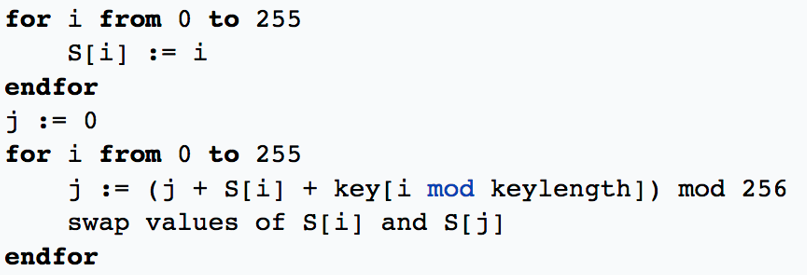
RC4：在Wired Equivalent Privacy(WEP) for 802.11和SSL中使用较多

强度很高且容易实现

包含两部分:KSA(Key Scheduling Algorithm)& PRGA(Pseudo Random Generation Algorithm)

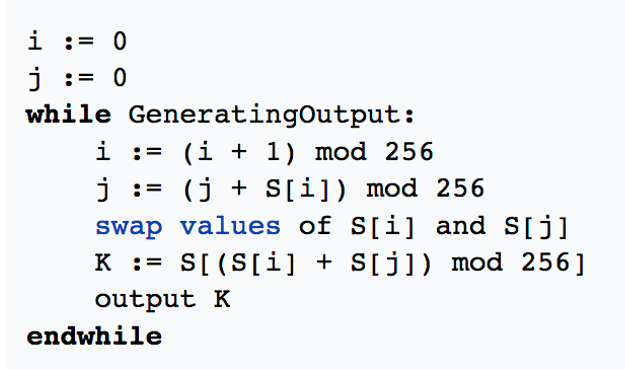


KSA: 假设一个Array S：



PRGA：

每次输出一个Byte的keystream，可以根据需求一直输出

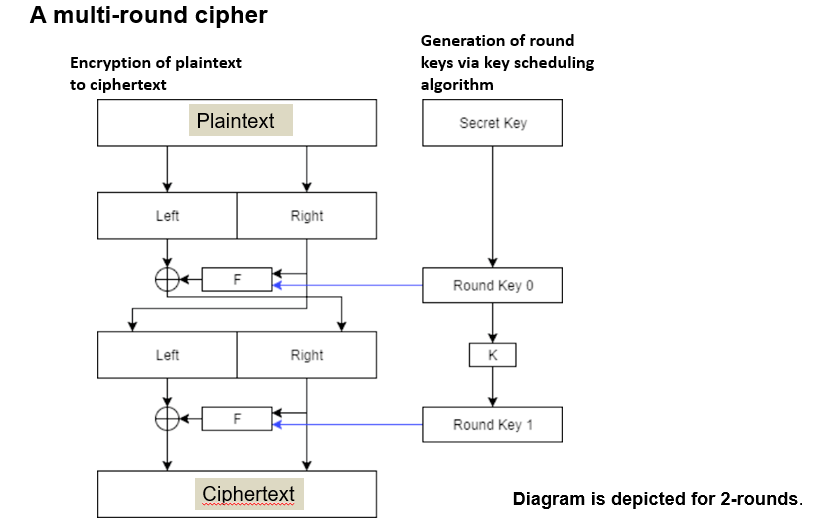


Block Cipher：plaintext被分成多个block，对每个block进行加密

Output ciphertext看起来是完全随机的，由plaintext iterating a round function

Round function的input是key和上一个round的output，一般由软件实现

Feistel Ciphers：用来构造block cipher的一种structure



每一轮的left就是上一轮的right，每一轮的right就是上一轮的left和round function的XOR。F就是round function

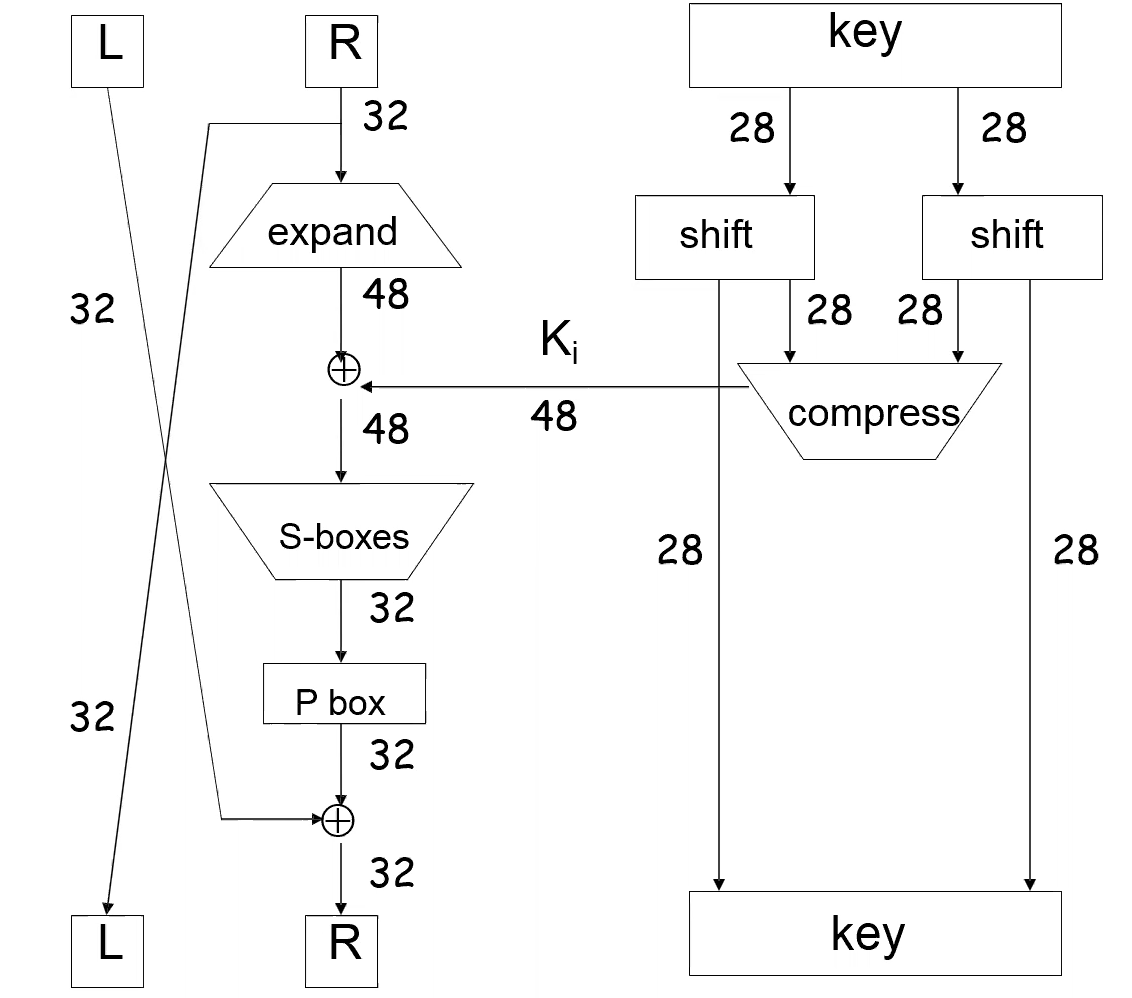
在Decrypt的时候只需要把round key反向使用就可以得到plaintext

DES(Data Encryption Standard)：64bits的输入和输出块，56bits的key(其实是64位，但是有8位只作为check bits)，16 rounds，48位的subkey

Round function相对简单，DES的security完全取决于S-boxes，其他部分完全是线性的

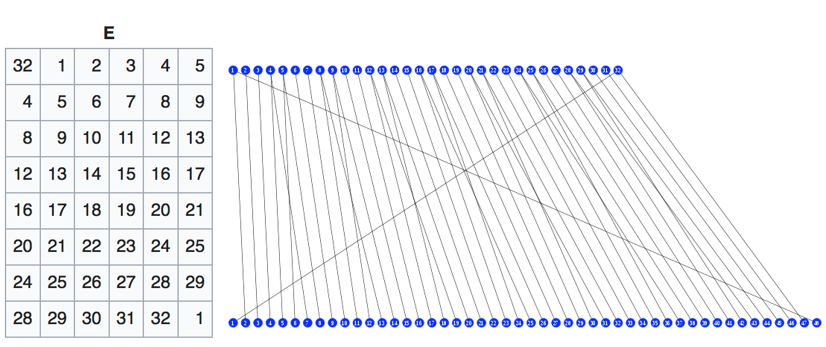
在DES的第一个round之前会将左右互换，在完成最后一个(round 16)round后互换回来

DES的一个round：

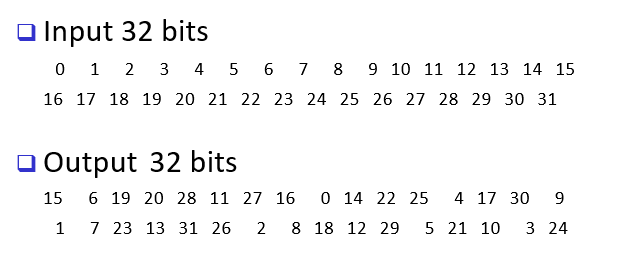


Expand通过下图的对应实现：

有的数会对应两个数，因此可以实现扩展

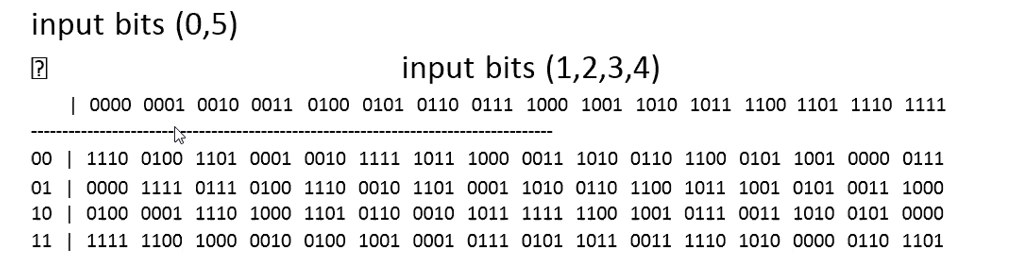


P box并不具有加密作用，只是简单的替换(逆矩阵)：



S-box的作用是将6位的数变成4位的数，比如：下图是S-boxes中的第一个box

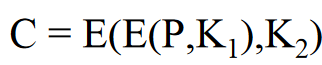
最左和最右的两位作为row index使用，中间四位与row中的column对应



由于现在DES的强度虽然没有back door，但是还是会被穷举破解，因此采用了triple DES



先用K1加密再用K2解码

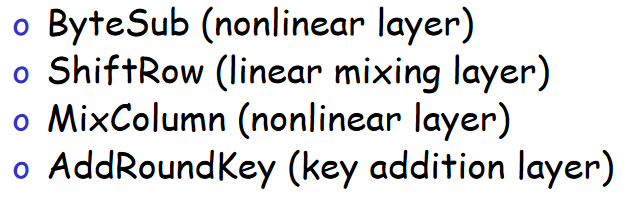
不用的原因是可以使用一种semi-practical plaintext attack,即使用预先计算好的有2^56个entries的table对K2的table进行破解，可以找到对应的

后来在1990s研发了AES(Advanced Encryption Standard):

仍然是block cipher，但是不再使用Feistel Cipher的structure了

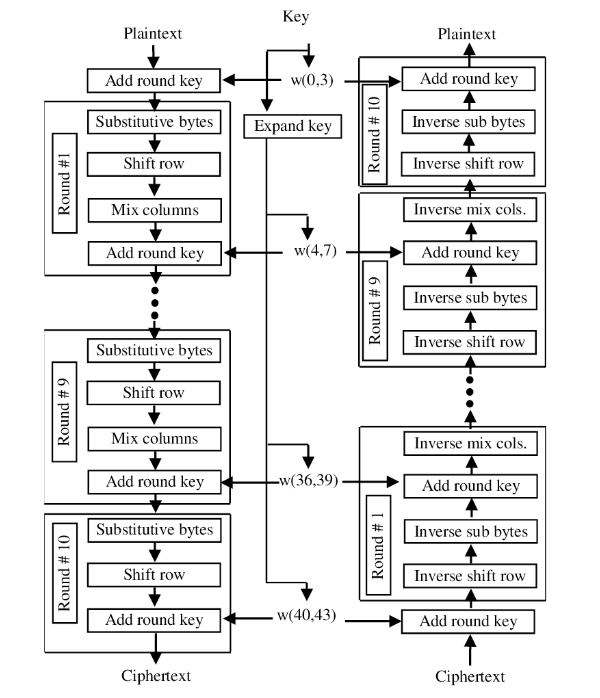
128bits block size, 128bits key length(也有别的长度)

有10-14round(取决于key length)，每个round使用下面的几个函数：

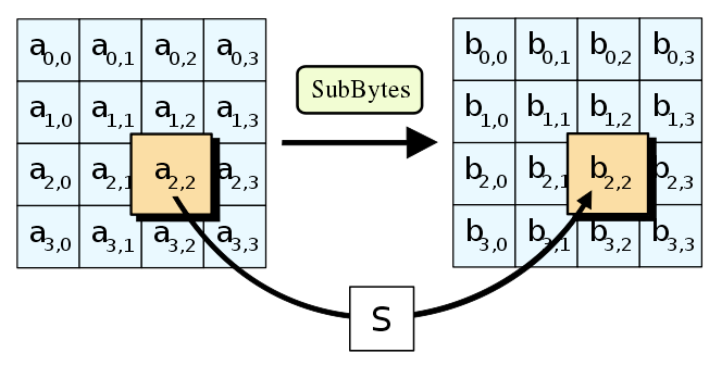


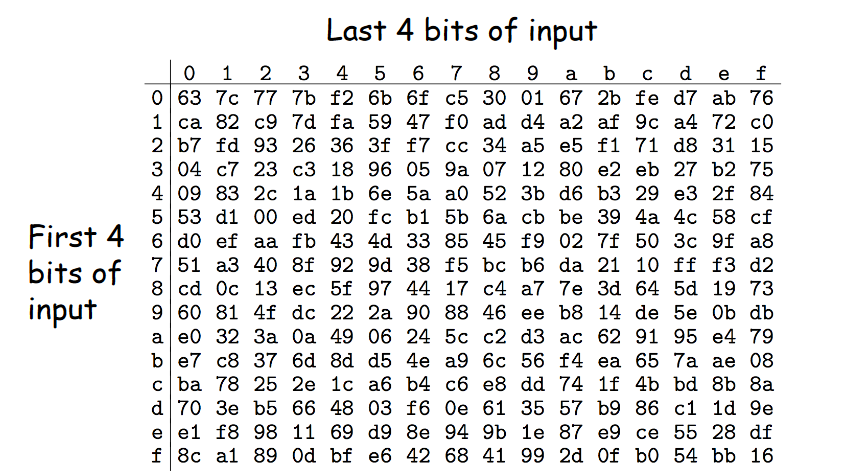
其中分为三个layer：non-linear，linear mixing layer，key addition layer

Block diagram：w代表一个word，32 bit(4 byte)为一个word

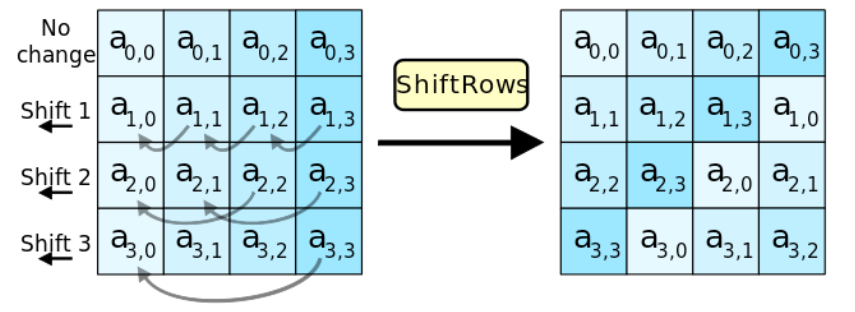


Byte substitution: 非线性，相当于AES的S-box，但是是可逆的invertible

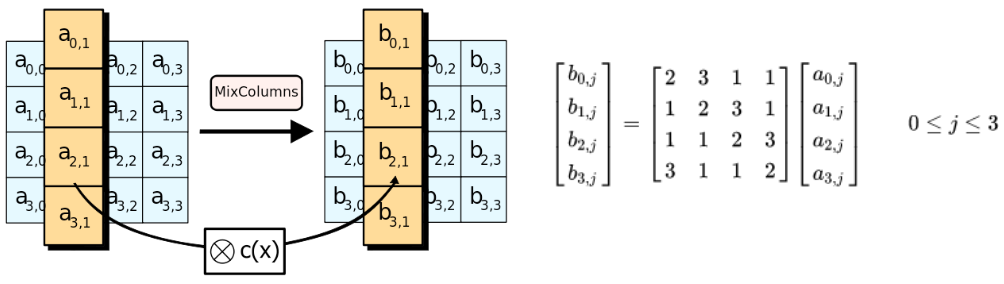




Cyclic Shift rows：根据行数左移



Mix Column：和一个matrix进行XOR运算，线性，可逆的过程。相当于look-up table



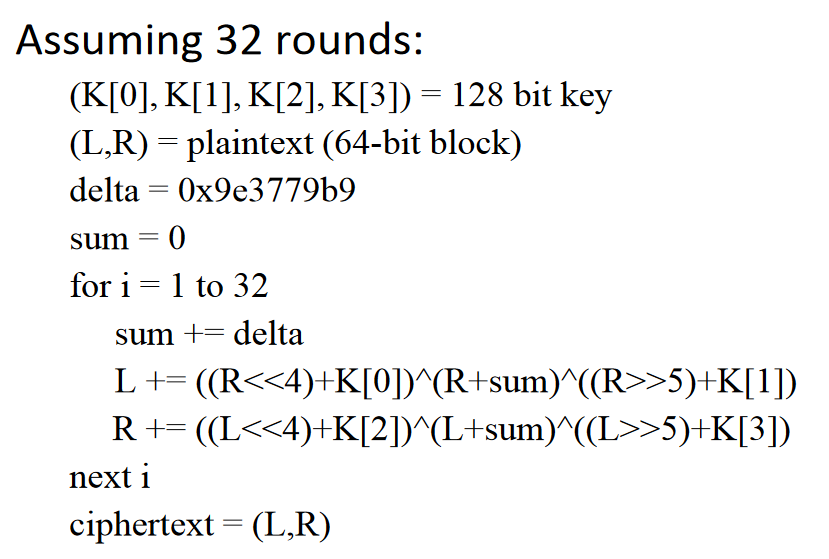
Add Round key：round key和block进行XOR运算得到结果

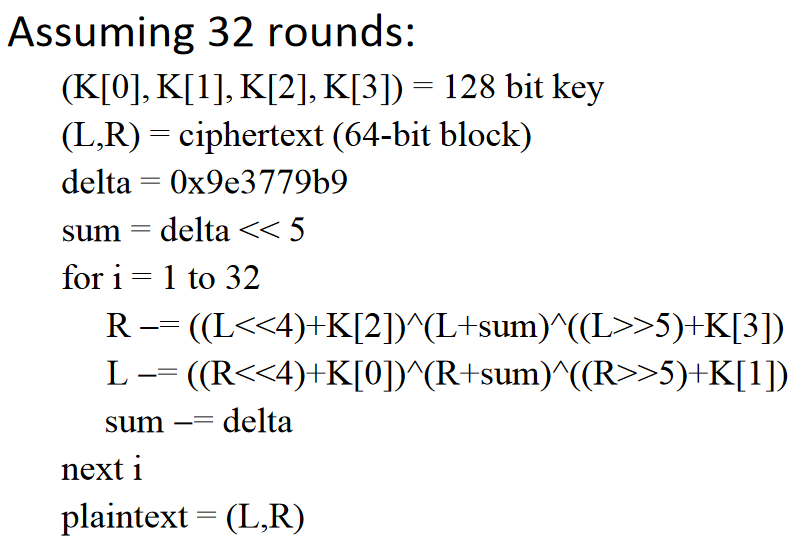
TEA: Tiny Encryption Algorithm

64 bit block, 128 bit key, round数目可变，一般认为32以上是secured

使用weak round function，但是round很多次：

Encryption and decryption：>>和<<表示shift





TEA非常简单,快且容易实现，对于memory的需求也小，但是可能会有related key attack，因此有了别的版本的TEA---XTEA(Extended TEA) & STEA(Simplified TEA)

Block cipher modes:

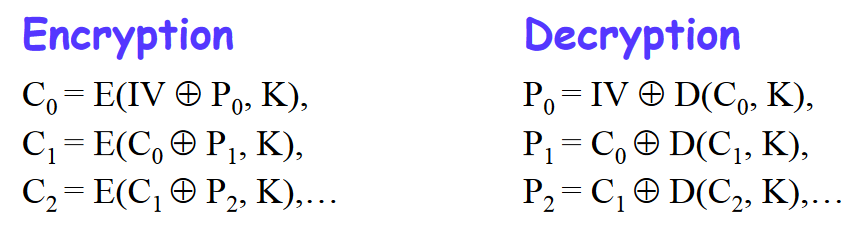
Multiple Blocks brings problem: 加密多个block如果分别独立加密或者使用完全不同的key并不现实，因此propose了几种block cipher modes：

Electronic Codebook(ECB): 用同一个Key分别加密各个Block，这里的同一个Key相当于是同一个codebook，而不是说完全一样的同一个key

Cipher Block Chaining(CBC): 将各个Block Chain在一起，容易实现并且比ECB更加安全

CBC需要一个随机的向量来启动，记为IV---IV是随机的，但是并不是secret的

实际上和运用了classic codebook with additive的算法是类似的

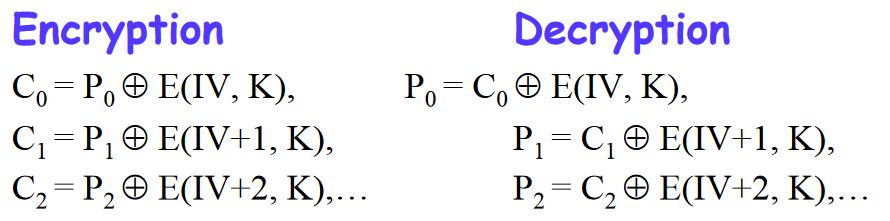


CBC的优点是就算是完全相同的plaintext block也会产生不一样的ciphertext

并且可以恢复传输中产生的error，因为后续的Cn和前面产生错误的Cm Block在接受时并没有联系，因此后续的Plaintext并不会受到影响

Counter Mode(CTR): 让Block Cipher像Stream Cipher一样工作，多用于Random Access

同样需要先生成一个随机的向量IV



Integrity: to detect unauthorized writing and modification

Encryption提供了confidentiality：prevent unauthorized disclosure of data，但是并不能提供integrity

一种提供Integrity的方法是使用MAC(Message Authentication Code):

MAC是通过CBC residue计算的，也就是CBC中的最后一个ciphertext block，相当于是cryptographic checksum(因为最后一个需要前面的全都正确)

一般使用的话会用一个key作为encryption使用，另一个key作为MAC使用

Attacking Symmetric Encryption：

两种：Cryptanalysis & Brute Force attack(暴力破解)

Cryptanalysis基于的是算法的本质，根据给定的一些plaintext的可能特点破译特定信息，从而deduce整个key

Public Key Cryptography(PKC): 也被称为asymmetric cryptography

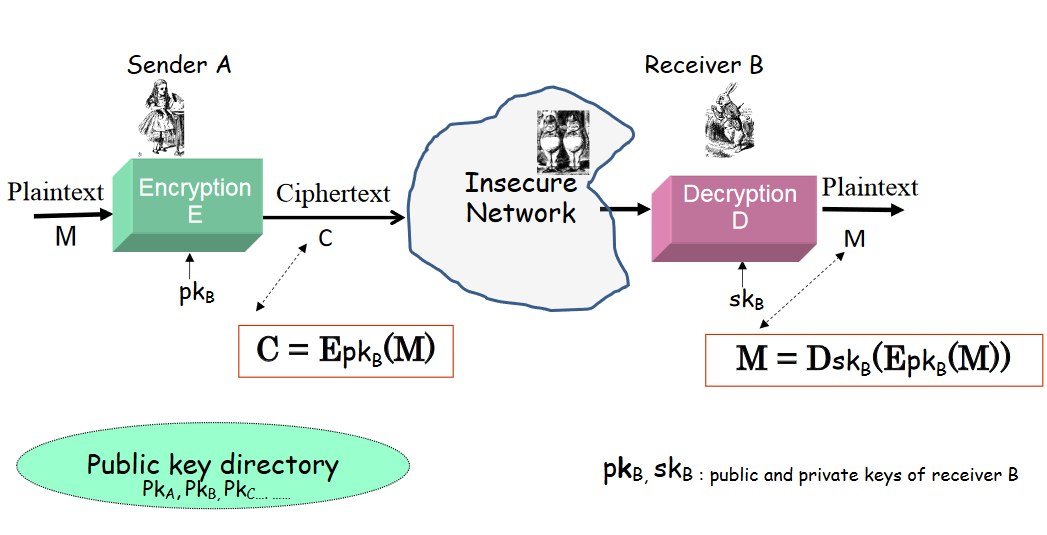
公钥加密主要被用来解决两个问题：key distribution---如何在不需要一个被信任的key distribution center的情况下实现secure communication；digital signatures---如何验证自称从某人发送来的信息是完整的

在PKC中有一组密钥：public key(pk) & private key(sk)

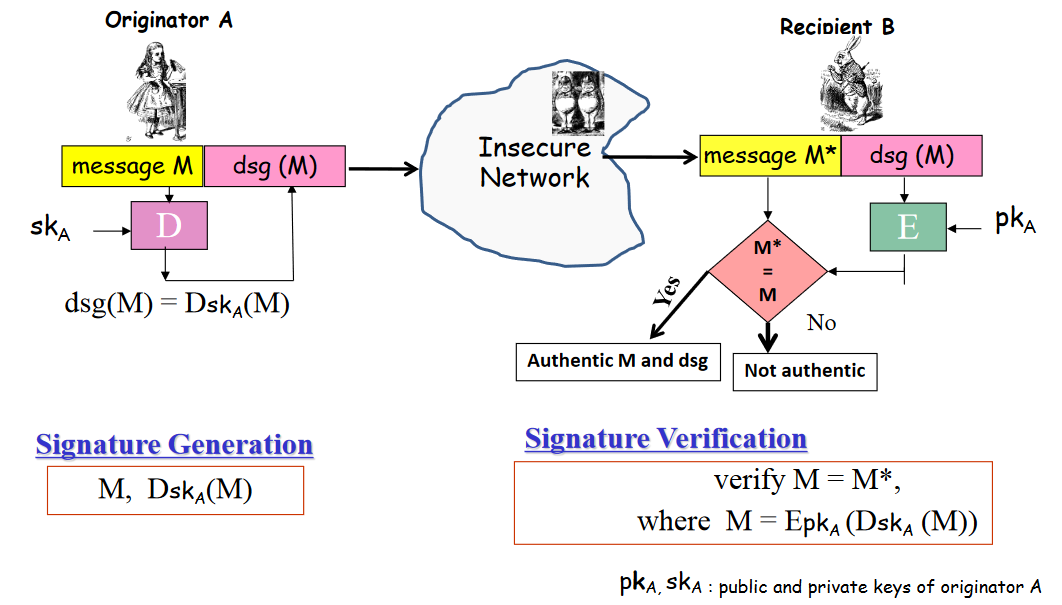
公钥用来加密以及验证signature，私钥用来解密并且生成signature。这种方式被称为trap door，one-way function：从一个方向计算很容易，但是从相反的方向就很难(在没有给定的信息的情况下)。

Trapdoor function：用来生成key pairs。给定的能让计算过程变得简单的信息被称为trapdoor。

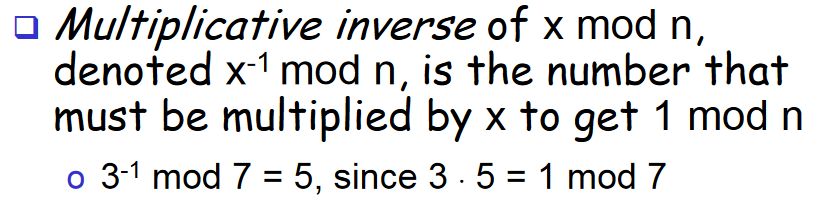
加密传输过程:

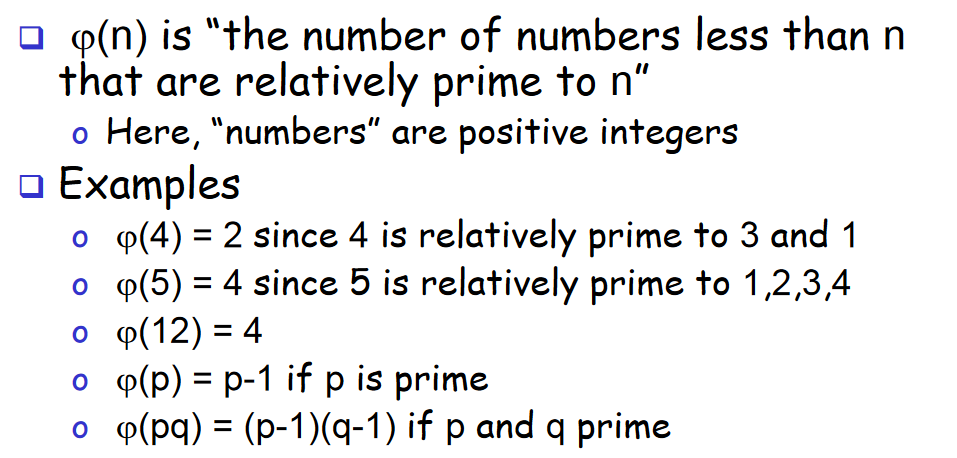


签名的验证过程：



Modulus计算中的一些表示：





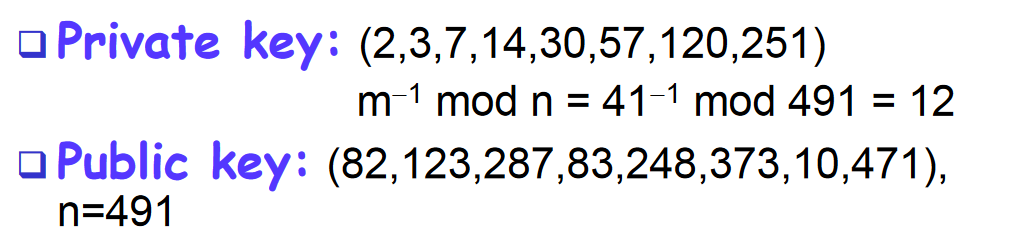
具体的非对称加密算法:

Knapsack: Knapsack是一个经典问题，给定一个容量固定的书包，如何在不超重的情况下装进价值最高的物品。General Knapsack是NP-Complete问题，也就是非常复杂的，但是另一种SIK(Super increasing Knapsack)是很容易解决的。SIK就是指后面的物品的重量要比前面所有的加起来还要大。

因此在Knapsack cryptosystem中，public key就是GK，private key就是SIK和conversion factor。首先有一串SIK(记为PK)，然后设定两个数m和n，其中n需要比SIK所有的数的和更大。通过对SIK中的每一项和m的乘积对n求模得到GK序列(i.e. GKi = [PKi \* m] % n)

私钥中的conversion factor通过m­-1 mod n求得(即最小的和m相乘后模n余1的数)

对于一串需要被加密的数来说：比如10010110，假设PK和SK如下：



则加密完的Ciphertext就是83+0+0+83+0+373+10+0=548

解密的时候将Ciphertext和conversion factor相乘并对n求模：548\*12 % 491 = 193

然后解sum=193的SIK就可以得到原来的10010110

在这个算法中conversion factor的作用相当于就是将问题从GK转换到SIK

但是Knapsack后来被Apple使用lattice reduction破解掉了（原因是因为得到GK的modulus算法还是不够general，因此可以破解），虽然现在有secured Knapsack，但是此后几乎就没有选择这个算法了

RSA:

假设p和q是两个很大的质数，用N=p\*q作为模的底，然后选择与p-1和q-1相质的e, 然后在选择d使得e\*d mod (p-1)(q-1) = 1。(N, e)就是PK，d就是SK

在需要加密的时候将要加密的信息M当成一个数来处理: C = Me mod N

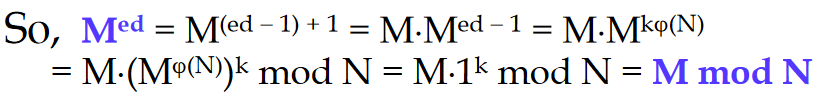
Decrypt：M = Cd mod N

证明需要涉及到Euler’s Theorem:

If x is relatively prime to n then xφ(n) = 1 mod n 其中φ(n)是欧拉函数，指的是小于n的正整数中和n互质的数的个数。

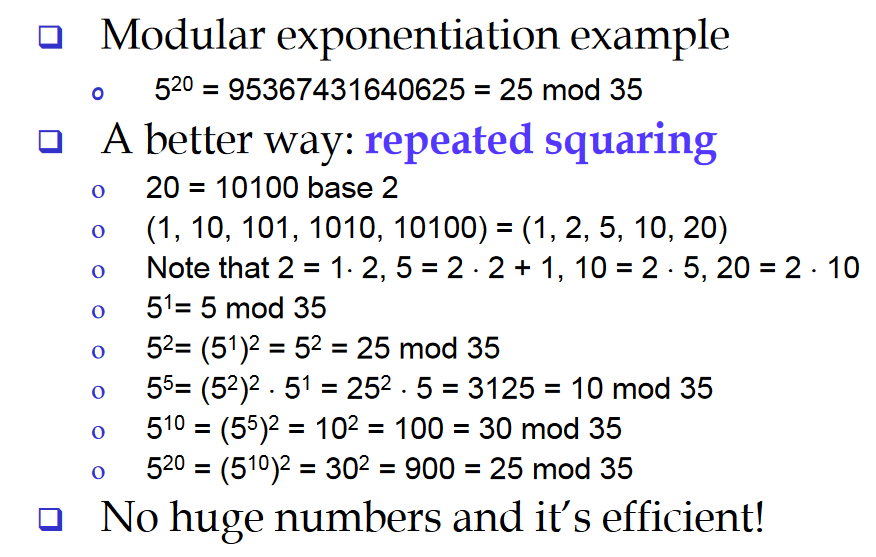
根据性质可以知道对于N来说φ(N)=(q-1)(p-1)，且有ed=1 mod (q-1)(p-1)

所以ed=φ(N)\*k + 1，通过这个关系可以得到：



在RSA中求次方是非常复杂和昂贵的，因此有一些算法可以更简单且有效地实现：

第一种是repeated squaring---利用上一步的结果进行平方往下求余数



第二种方法是对于所有用户使用同一个e：这样public key操作只需要两次乘法，但是private key的操作仍然很昂贵，因为对于不同的user来说解密所使用的p, q, N仍然是不同的

但是如果M比N的三次方还要小的话，modulus操作就失去了意义，这种情况被称为cube root attack。解决方法就是让M的位数足够多(i.e. 数足够大)

Diffie Hellman:

这是一个key exchange algorithm，目的是获取一个symmetric key，而不是用来加密或者验证签名的

这个算法基于discrete log problem---已知g, p, gk mod p，求k

对于Diffie Hellman algorithm来说，令p为质数，g称为generator

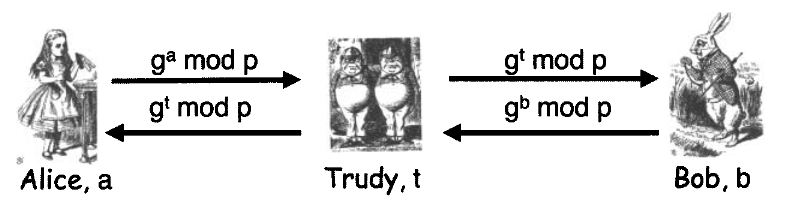
对于任意一个比p小的整数x来说，一定存在n使得x = gn mod p

假设A和B之间互相通信，A选择a并发送ga mod p给B，B选择b并发送gb mod p给A，则gab可以作为两人通信时的symmetric key使用

在这个情境下，g和p就是public key，a和b就是private key

如果中间的信息被人获取，则需要解决discrete log problem才能够得到a和b(discrete log problem虽然没有达到NP-Complete的程度但是还是非常难解的)

但是这个算法会收到Man-in-the-Middle Attack：



则T可以在A和B都无法察觉的情况下获取对称密钥，从而监听对话的所有的内容。

可以通过加密exchange的过程(对称密钥或者公钥) / 通过私钥签名

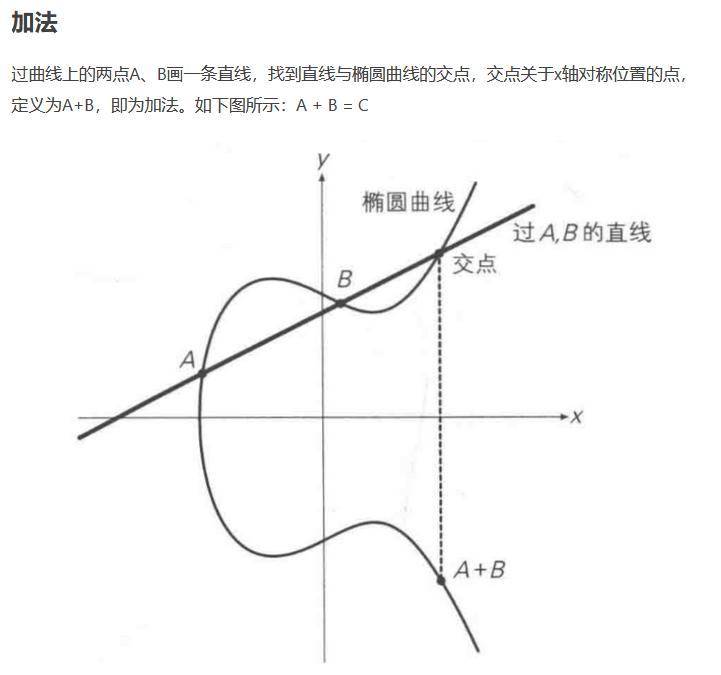
Elliptic Curve based Cryptography(ECC):

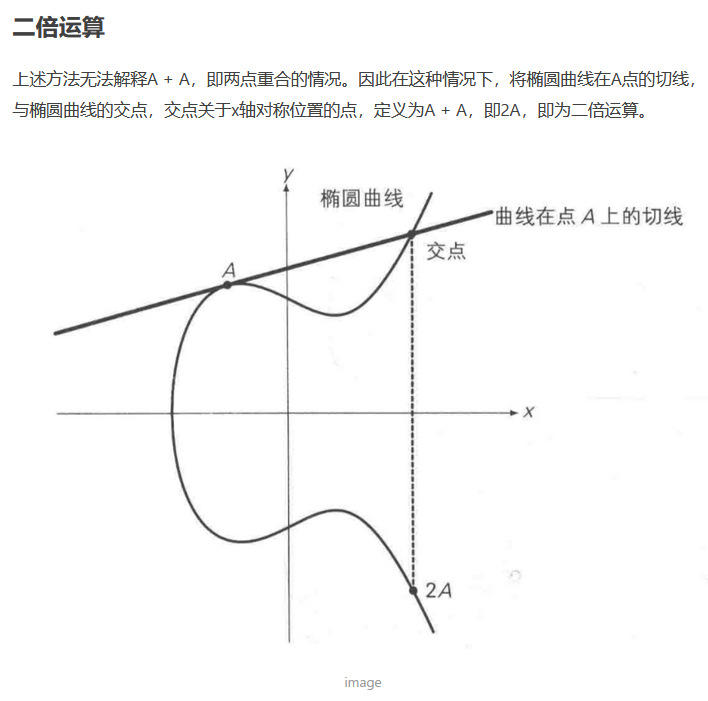
ECC并不是一个加密算法，只是通过椭圆曲线来进行public key system中的数学计算，并且可以通过更少的bits实现相同强度的加密。

标准的椭圆曲线方程：E: y2=ax3+ bx2+cx+d，且必须在零点处有切线(即非奇异的，光滑的)

但是在这里用到的是简化版的Weierstrass方程：E:y2=x3+ax+b

首先定义椭圆曲线加密中的运算：

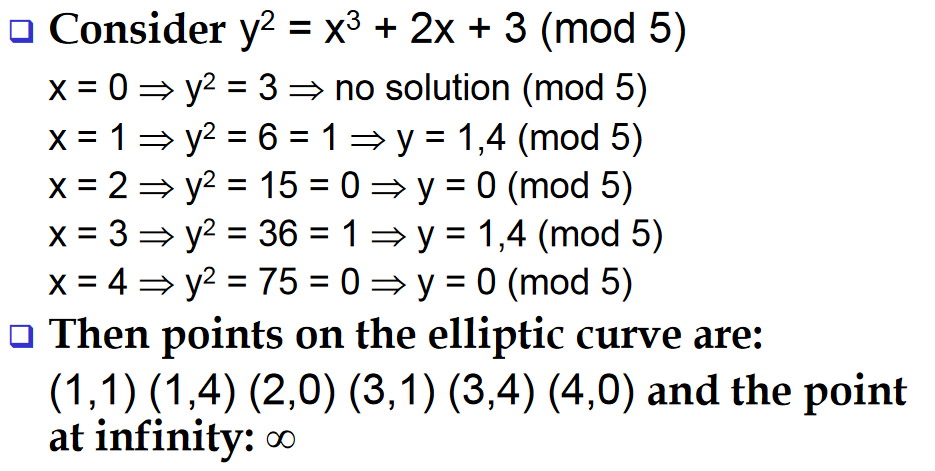




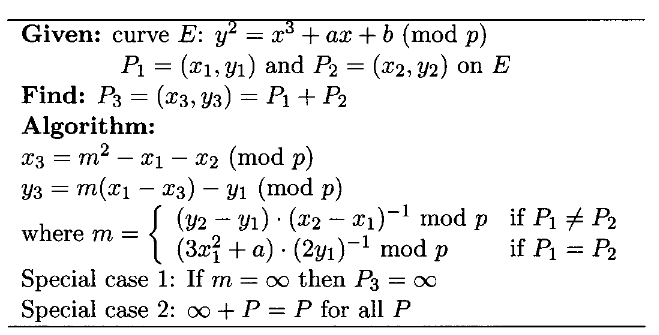
负号为关于X轴对称

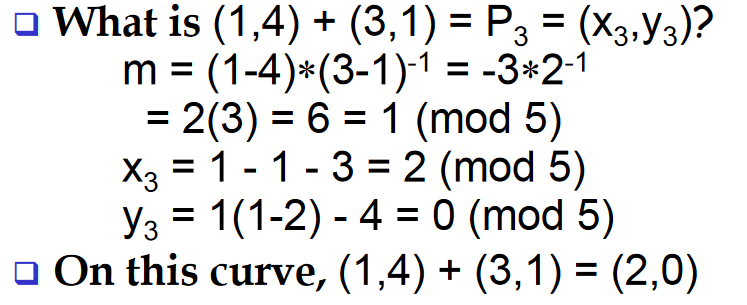
椭圆的无穷远点记为O∞，有计算：O∞+ O∞ = O∞，O∞ + P = P

但是由于椭圆曲线是连续的，并不适合加密，所以通过modulus运算将其变成离散的点：



用算法可以表示上面的几何加法：

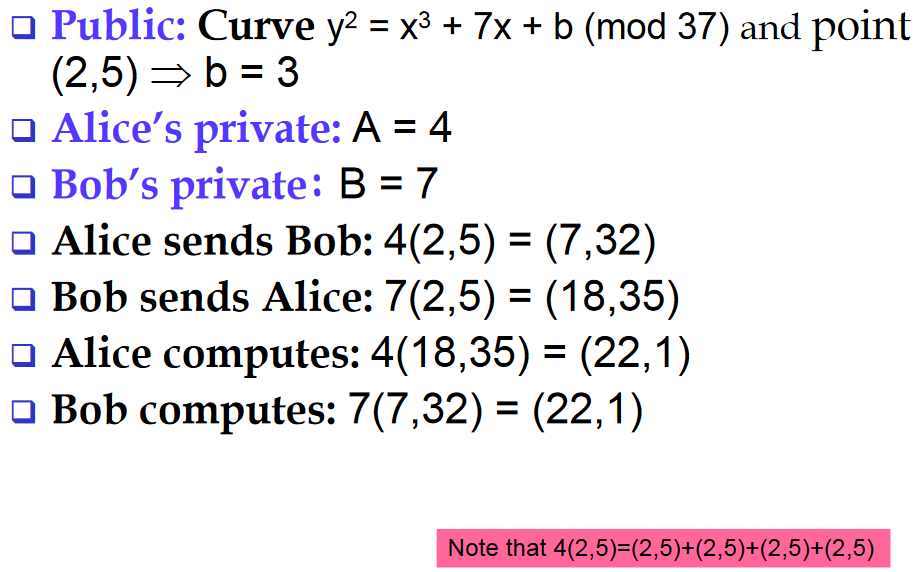




ECC Diffie-Hellman key exchange：

在对称加密中可以用曲线和一个点作为公钥，两个人的倍数作为私钥：

公钥的点记为(x,y)，因为AB\*P=BA\*P，所以得到的结果是相同的



就可以得到一个shared secret

详细的可以参考<https://blog.csdn.net/sitebus/article/details/82835492>

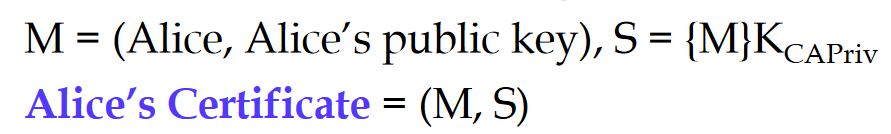
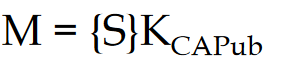
Public key crypto的用途：confidentiality，Authentication protocols，Digital Signature---provide non-repudiation

Non-repudiation：假设一个客户下了一个订单，但是不小心将货物弄丢了并且声称没有下过订单，由于使用的对称加密因此商家也可以伪造相同的MAC，所以没法证明客户是否下过订单。但是public key crypto避免了这个问题，因为客户的私钥对别人是未知的，因此没法伪造

Sign and Encrypt & Encrypt and Sign

两者都是安全的，因为中间人没办法破解

Digital certificate是由Certificate Authority(CA)提供的，可以通过CA提供的public key进行验证



Public Key Infrastructure(PKI)：the stuff needed to securely use public key crypto

-key generation and management

-CA

-Certificate Revocation Lists(CRLs)

-……

并没有统一的标准，但是通常来讲有三种模型：

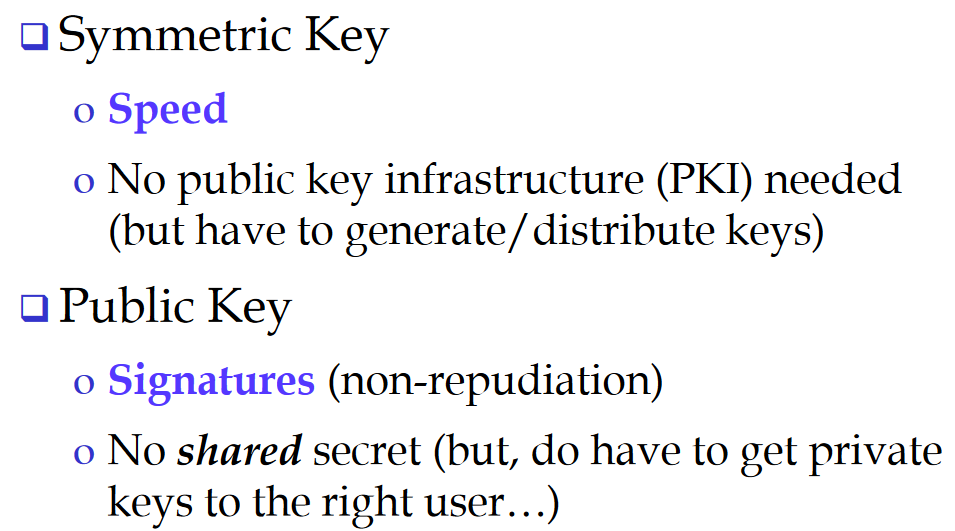
Monopoly model: 全球都通过一个CA来进行认证。问题在于谁来当这个CA？并且如果这个CA一旦出现问题就是很大的问题

Oligarchy model：有少数几个trusted CA---这种方式现在多用于浏览器中

用户可以选择信任任意一个CA

Anarchy model：任何人都可以是CA，用户必须选择谁来信任---在PGP中使用

对称和非对称加密的对比：

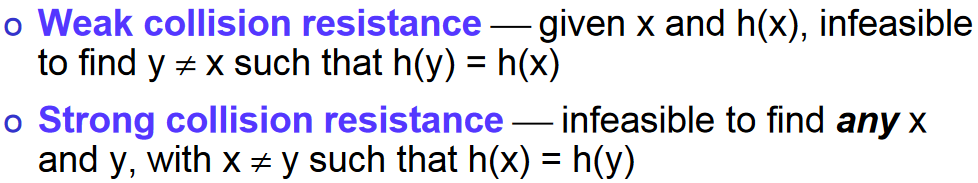


Hash function：

Motivation：假如M很大的话，那么签名S算起来和发送都是非常costly的

因此如果能用一个函数简化一下就显得非常合适

因此有了哈希函数：h(x)---提供了compression，efficiency，one-way，weak collision resistance & strong collision resistance



对于一个哈希函数来说，如果想暴力破解则需要2^(N/2)次操作

那么如何设计hash function就很关键：预期的properties有很多：

---avalanche effect，改变一个bit的input，则output应该改变一半左右的bits

---需要有一些rounds

---既需要security也需要speed

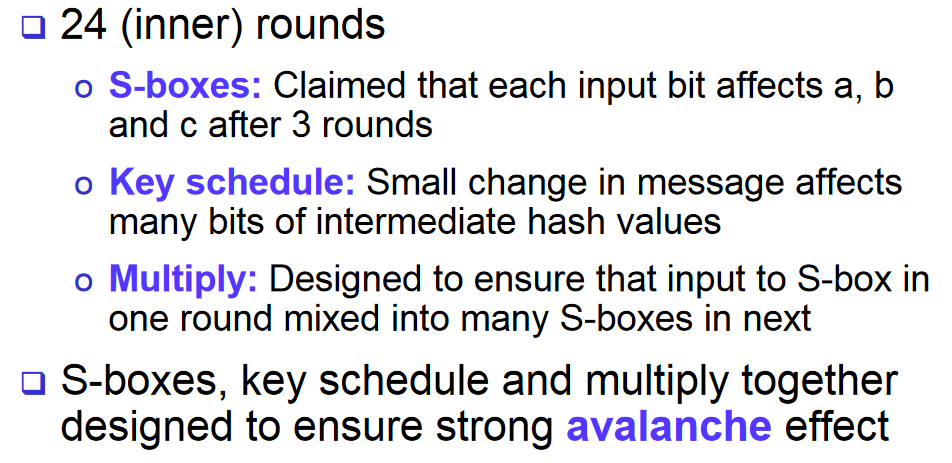
Tiger Hash: optimized for 64 bits hash function, and secure

Input被分成512bits的block，最后的output是192bits(3\*64bits)

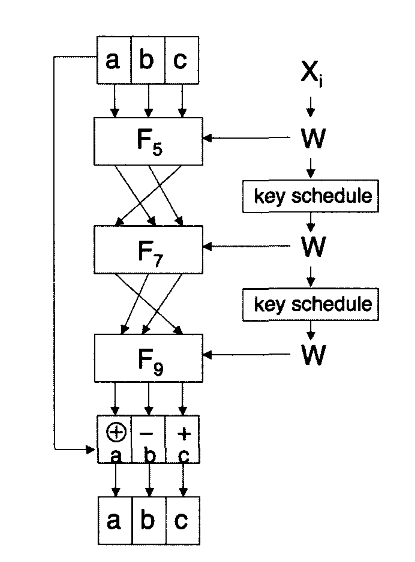
中间的轮次的block都是192bits的

4 S-boxes, each maps 8 bits to 64 bits

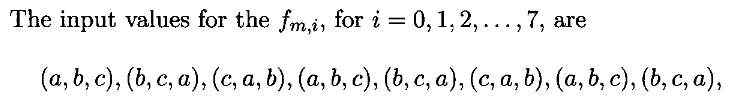
Key-schedule used



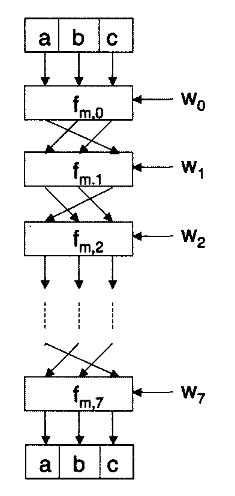
Tiger hash的其中一个outer round，重复三次



A，b，c的序列和Inner round的序列有关：



每个F有8个inner round：



HMAC：在计算hash值的时候，把Key也混进去---即验证哈希值的时候还需要提供正确的口令才能得到正确的结果

但是H(M,K)和H(K,M)都不是很合适：

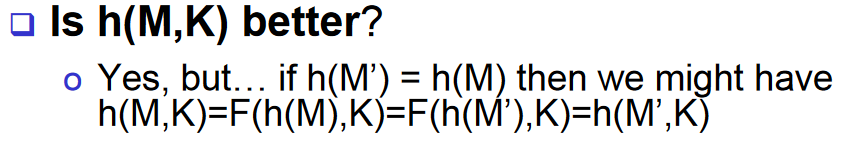
如果使用的是H(K,M)，则有:



假设Trudy尝试破译，并令M’ = (M,X)，由于F是已知的，所以可能在完全不知道K的情况下推导出M’的HMAC：

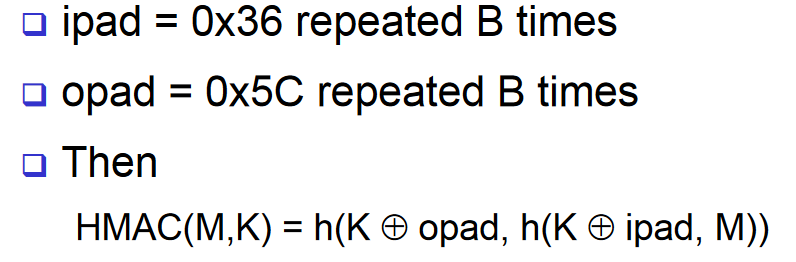


那么H(M,K)呢？



由于这两种方法都不可行，因此在RFC2014中规定了正确的HMAC方法：

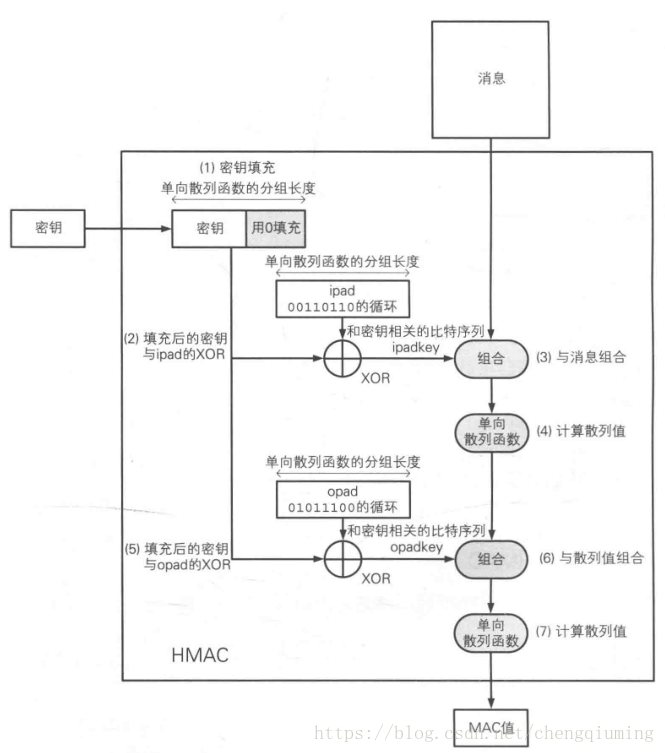
假设B为hash方法中每个block的长度in bytes(比如tiger，MD5和SHA-1中为64-bit)



ipad和opad就是和block等长的0x36和0x5C的循环

这样可以完全地把key混进输出的hash中

具体的图解：



HMAC的用处：

在线竞标online bids---可以用于保持bids secret，并且发送竞标后不能改变(因为会发生collision)。但是有个问题：会被forward search attack，但是可以通过一些小trick解决

Spam reduction---减少垃圾邮件：并不能消除垃圾邮件，只能让发送垃圾邮件的成本变得很高

假设现在的时间为T，R是需要发送方求的值，M是邮件信息

Sender需要求得R使得h(M,R,T)=(0,0,0……0,X) 开始的N位都为0

发送端把(M,R,T)同时发送过去，接收端计算h(M,R,T)是否前N位都为0

工作量对于发送端来讲大约是2N次hash，对于接收端永远是一次hash

所以要合理地选择N使得对于发送端来说工作量不会过于庞大并且对于垃圾邮件发送者来说过于庞大

2.Access Control: 所谓access control，其实就是对系统资源访问的权限的控制，也就是安全问题

(1) Authentication---一个用户是否应该被允许access a system

Identification和authentication的区别：identification是一个一对多的关系，从一组数据集中找到所对应的一个，而authentication是一个一对一的关系，即你是否是你声称你是的这个人

authentication必须要cooperative subjects，而identification不需要，就算subject不配合也应当能够检测出subject的身份

如何实现authentication：可以基于你所知道的，比如密码；可以基于你本身，比如各种biometrics(fingerprint……)；也可以基于你有的，比如IC卡等

Password：安全性其实很差 但是由于没有成本且对于administrator来说更容易修改，所以使用的还是很多。破解password不像破解crypto中的key，要容易很多，而且由于密码不是随机的，可以使用dictionary attack

由于比较好破解，所以一般在尝试几次之后就会锁住系统

存密码的时候需要用到hash function，不可以直接存密码

但是破解方法可以结合dictionary和forward search推测h(x)

为了防范这种attack，可以引入salt：对于每个password随机生成一个盐salt，salt不是secret的，类似于CBC和CTR中的IV。对于attacker来说，进行dictionary attack时要重新计算一遍hash，会增加很多的工作量

Biometrics：

对于一种biometrics来说，我们希望它有以下特质：

Universal---基本上所有人都有的

Distinguishing---能够区分开不同个体的

Permanent---被测定后不会轻易改变的物理特性

Collectable---容易收集的

Safe & user-friendly

步骤：enrollment & recognition

Enrollment：put subject’s biometric info into database

可以比较缓慢或者多次重复，但是必须非常准确

Recognition：Biometric detection，需要快，simple且相对准确

Biometrics的fraud & insult：

Fraud指的是A被辨识成B，insult指的是A无法被辨识成A

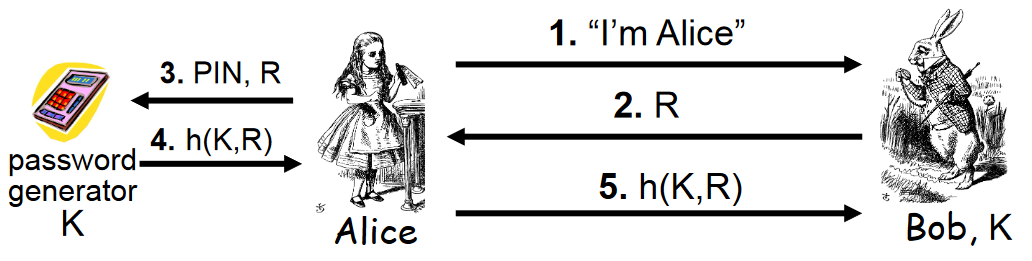
Fraud rate和insult rate之间是一个tradeoff的关系，一个降低则另一个就会升高

常用的一些biometrics：fingerprint（过程大概是记录整个图像，增强并选取其中的一些点，在recognition时将点分离出来并与数据库中的点的数据进行对比），hand(Palm print)，Iris(虹膜，基于hamming distance算法，容易使用眼睛的图片进行破解，因此需要加上活体检测)，

一般来说会考虑一个系统的EER(Equal Error Rate, 即fraud rate=insult rate)：令这两者相等时一般会被用来比较不同的biometric system

Something you have：典型的比如IC卡，汽车钥匙等

密码生成器：password generator：密码生成器通过和B使用相同的key进行hash运算



2-factor authentication：使用三种方式中的两种，比如password + biometrics

Single sign-on：一次认证之后以后就不用重复认证了，通过Kerberos protocol结合cookies等技术实现

(2) Authorization---限制authenticated user能够访问的资源，也就是权限

首先是system certification，过去向政府出售product需要政府认证security level，相当于是authorization的历史

Orange book：Trusted Computing System Evaluation Criteria (TCSEC)，用来检验security products

后继者：Common Criteria---international government standard

EAL1-7---Evaluation Assurance Level

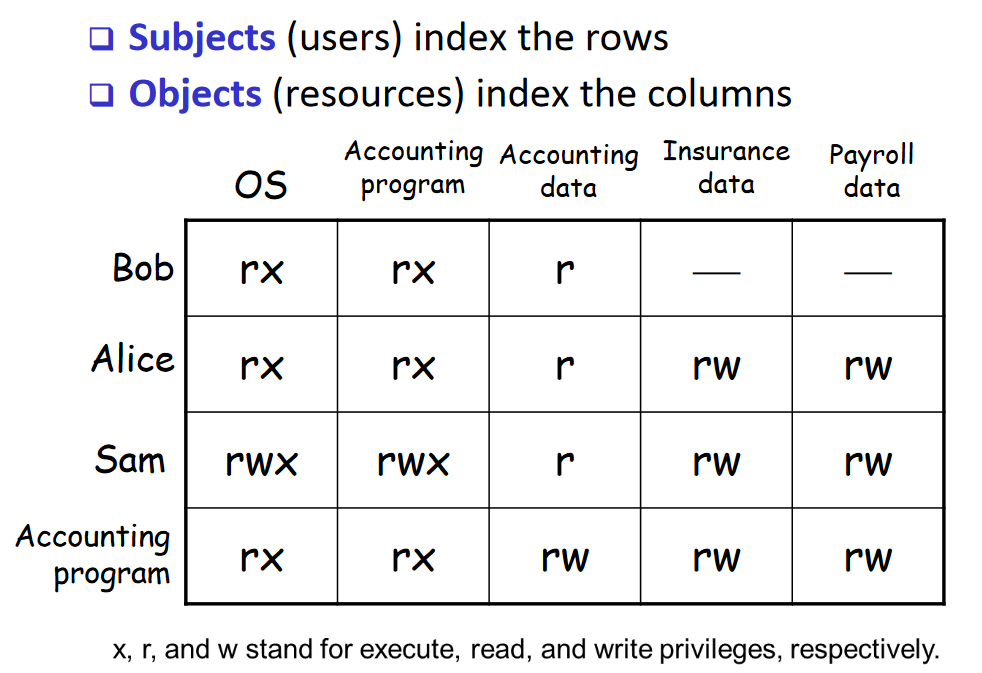
EAL4 is most commonly sough---Minimum needed to sell to government

EAL7 requires formal proof so not many products existing (about 2 EAL7 products)

Classic View of Authorization：

Access Control Lists(ACLs) & Capabilities(C-Lists)

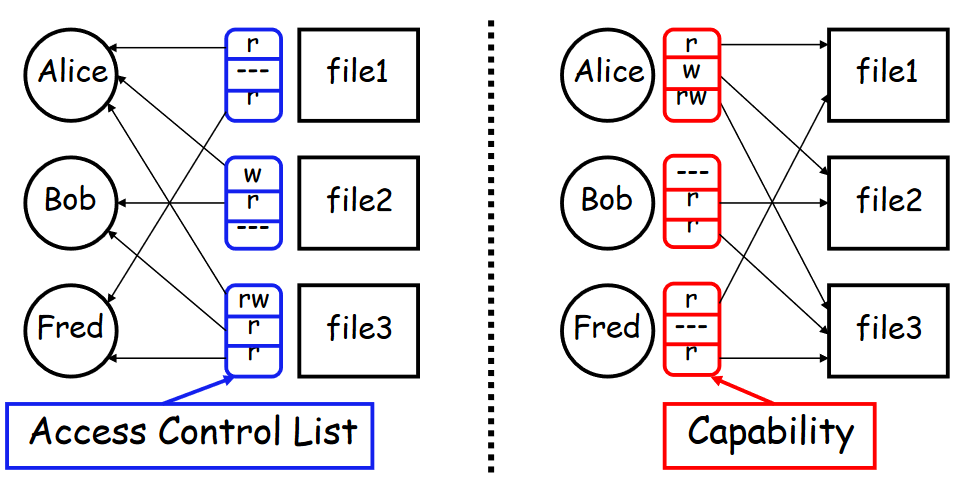
Lampson’s Access Control Matrix:



ACL和C-Lists就在这个矩阵中可以体现出来：

列 控制的是ACL，也就是Access control list，对于一个资源来说访问权限的控制

行 控制的是Capabilities，也就是对于一个用户来说权限所能及的范围

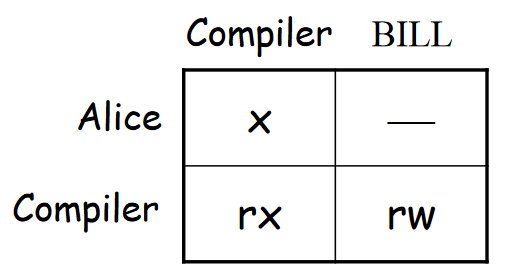


相比较说，C-Lists是built-in-system的，而ACLs需要单独的方法来进行关联，所以Capabilities相比较ACLs来说有天然的优势，但是更难实现。ACLs更容易实现对资源的管控，更改对某个资源的访问；而C-Lists更容易实现对用户权限的管理，增删用户等，而且可以避免confused deputy问题：

Confused Deputy---困惑的助手：权限问题

比如有一个由server提供的收费编译服务，收费的记账BILL只有compiler有读写权限

Alice即client可以调用compiler来进行编译服务。假如Alice把编译输出的文件命名为BILL，那么compiler就会把编译输出的结果覆写到BILL中，尽管Alice并没有读写BILL的权限，这个问题就叫做Confused Deputy



Classification & Clearance：

Classification是对object而言的，Clearance是对subject而言的

US Department of Defense(DoD)用四个等级来定义：

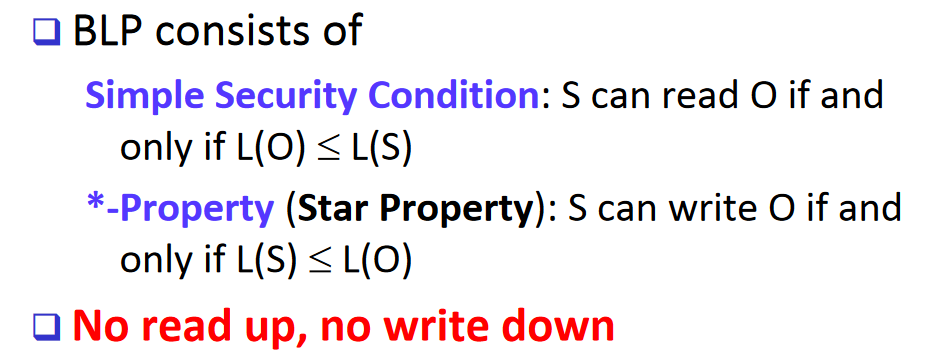
TOP SECRET，SECRET，CONFIDENTIAL，UNCLASSIFIED

利用L(S)和L(O)来代表Object和Subject的等级，也就是上面四个等级

当L(S)和L(O)不同的Object和Subject访问相同的system resource时，就需要用到Multi-Level Security(MLS): 比如身份信息中有的对于ALL Public可见，有的只对公司内部人员可见，有的只有senior manager才可见，这就是MLS的一种应用

MLS model：这里只会提及比较简单的，其余的模型比较难以分析

Bell-LaPadula(BLP)：用来实现confidentiality---prevent unauthorized reading

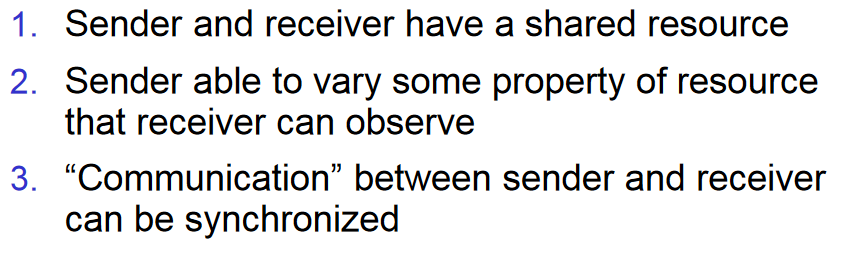


Biba’s Model：用来实现integrity---prevent unauthorized writing

假设一个Object中有多个Object，则其integrity level由所有object中最低的一个决定

Covert Channel：并不是设计者所希望设计出来的，可能跨等级泄露一些信息：比如文件系统中，A和B共享一个File Space，A的等级>B，A可以通过写入一个文件向B传输信息，B可以通过是否有文件存在获取这个信息，比如有文件则记为1，没有文件则记为0。通过不断地检查文件，B可以读取一串信息，即可以获取A的Clearance level的信息

Covert Channel存在的条件：



但是Covert Channel的存在几乎是不可避免的，所以只能尽量减小convert channel capacity，比如DoD尝试将其限制在每秒一个bit的速率

Inference Control：推断控制

有一些特殊的信息可以普通信息的回复中推断出来，需要避免这种情况

比如说medical records，有很大的研究价值，如何在不泄露个人信息的情况下进行研究？

只是去掉名字是不够的，比如可以添加很多限定条件使得这个群体很小，就可以得到一个人的详细信息

因此首先要避免从人数/数量推断，所以有query size control：少于k%时不会release info

Randomization：向数据集中加入一些随机的noise

CAPTCHA：验证码

Completely Automated Public Turing test to tell Computers and Humans Apart：

理论上讲现在还没有能够通过图灵测试的AI

CAPTCHA中的Public指的是测试所用的program和data是public的

CAPTCHA的目的是为了只让人而不是机器可以获取权限，所以其实是一种access control

CAPTCHA的规则：让人很容易就可以通过，而机器很难通过（即使知道CAPTCHA软件是怎么写的）

最好有多种验证码的方式，比如盲人无法使用视觉验证；现存的验证码方式主要是audio和visual，text-based不太现实，因为很容易识别

破解CAPTCHA的方式就是解决一些AI中的经典问题，比如OCR中的segmentation problem，distorted text/audio，对于人来说很容易解决，但是电脑并不擅长处理这些

Firewalls：防火墙---其实就是对于network的access control，功能就是filter

防火墙的种类：

Packet filter：在network layer工作---可以通过source IP/Port，dest IP/Port和Flag等来进行筛选。优点是快，缺点是没有state的概念(因为对于网络层来说TCP的链接是不可见的)，并且无法与application data关联

会受到ACK SCAN攻击：attacker通过不断地尝试发送ACK通过防火墙以获取可以通过防火墙的Port Number，在ACK包通过之后接收到这种ACK时会返回一个RST被置1的包(这里会收到一个RST为1的包是因为TCP的工作原理，如果发送时这个TCP连接不存在的话就会返回一个RST为1的包让另一端关闭这个连接)，因此就可以判断可用的端口号。ACK SCAN是很多攻击的第一步

Stateful Packet filter：在transport layer工作---在packet filter中加入了state的概念，通过connection id等识别TCP链接，甚至可以试别DNS过程中的UDP包

优点是在保留了所有Packet filter的功能的同时避免了ACK SCAN

Application proxy：代理，在application layer工作

在允许进入之前检查application data

优点是可以全面地检查各种信息，包括application data中的病毒，字宏等等

缺点是慢

Deep Packet Inspection: 基于application data的特征值，行为模式和应用层协议进行检测

Intrusion Detection System(IDS)：Detection而不是prevention

IDS是从日志分析演化而来的，用于检测正在发生的intrusion，而不是像Intrusion Prevention一样防止Intrusion发生

可以从Approach和architecture来进行分类：

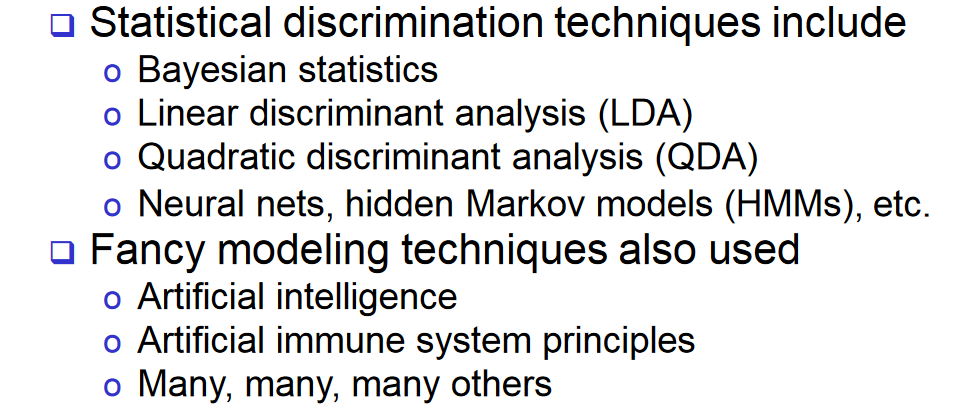
Approaches：

Signature-based IDS：根据已知的特定signature或者pattern检测attack

比如说最经典的signature，就是M秒内输入错误N次密码就会发出警告---但是如何定义次数和时间，需要基于一个normal behavior来定义；以及如果intruder每M秒尝试N-1次以正好避开这个检测机制，就需要用到一些别的策略，比如启发式算法等等

Signature detection的优点：简单，效率高(在数量还reasonable的情况下)，在检测到时就知道attack的类型；缺点：signature file需要及时更新，只能检测到已知的attack，如果signature的数量过多的话系统效率就会极其低下

Anomaly-based IDS：定义一个behavior baseline，当行为偏离这个baseline过远时系统会发出警告：这个baseline通过mean定义，距离这个baseline的最大的正常距离(也就是正常和不正常的距离)通过variance定义。Mean通过具有代表性的behavior进行测量，一定不能在attack时测量；测量abnormal，也就是variance的方法则很多：



但是对于Anomaly Detection来说，IDS并不是静态的，而是需要一直更新的，因此攻击者可以通过一点点地改变IDS中的mean值和variance让系统认为attack是正常的。

Anomaly Detection的优点：有可能检测到未知类型的attack

缺点：不能被单独使用(因为是动态的，需要和比如signature detection一起用)；可靠性并不明确，不能指出具体的关于attack的信息，只能给出有被攻击的可能性

Architectures：

Host-based IDS：detection methods会应用于在host上发生的activity，观测一些已知的attack以及可疑的行为，对于network activities关注的比较少或者不关注

Network-based IDS：detection methods会应用于network traffic，观测一些已知的attack和可疑的网络行为，可以用于检测DoS，Network Probes等攻击，较少关注host本地的activities。和firewall有一定交集

也可以有Network & Host based IDS

3.Protocols:

Protocols are rules of interaction，比如网络的协议就是network protocols

同理也会由security protocols：protocols flaws可能是非常细微的，现存的很多协议中都有很多细微的问题；同时也会有implementation error

理想的security protocol应有的性质：

能够满足security requirements；efficient(BW use, delay, computational requirements)；Robust；Easy to implement and use，while being flexible

Simple Security protocols：

简单的插入卡片，输入PIN，对或错，比如说ATM

Authentication Protocols：

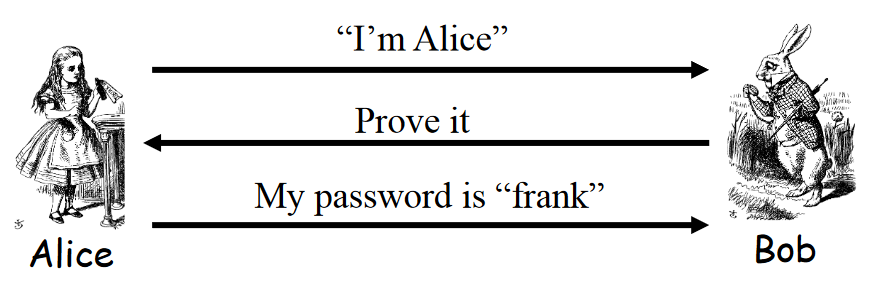
A和B之间互相通信：A需要向B证明自己的身份，也就是authentication；但是有时候也需要B向A证明自己的身份，也就是mutual-authentication：有可能需要session key---用来保证confidentiality和integrity的symmetric key

还有可能需要Anonymity, plausible deniability, perfect forward secrecy, etc.

在单一机器上的authentication相对简单，但是放到一个网络中的话就比较难以实现：

Simple Authentication：

在单一机器上是可行的，但放在网络中很容易被窃听，即受到replay attack(重复相同对话)



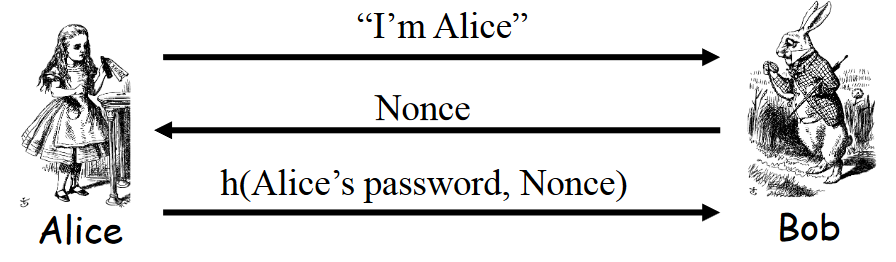
即使password通过hash发送也有同样的问题

Challenge response：为了避免replay而使用的一种strategy

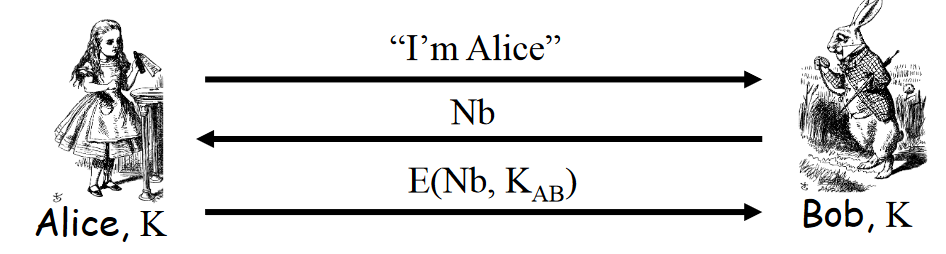
B对A进行challenge，要求A证明自己的身份

在每次challenge的时候，为了保证freshness，会每次随机生成一个nonce(Number Used only once)

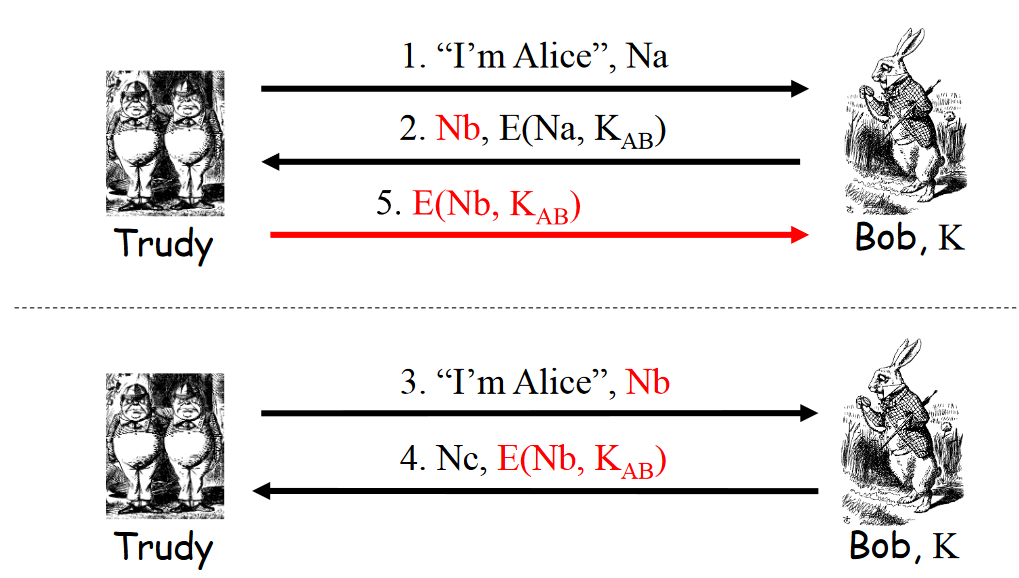
Nonce在这个过程中就是challenge：但是注意在这个过程中B必须要知道A的password才能够进行验证



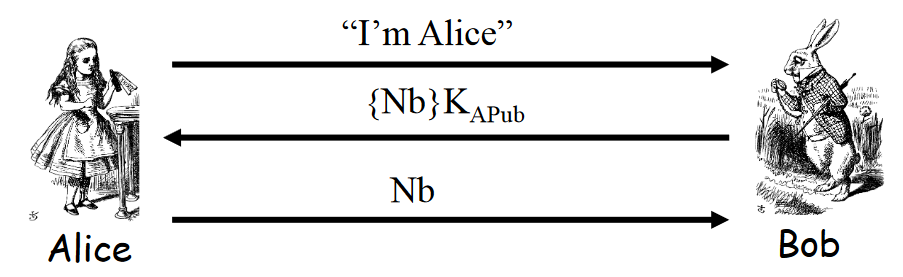
但是这个过程还不如用加密来的方便和好用，所以使用symmetric key的方法如下：



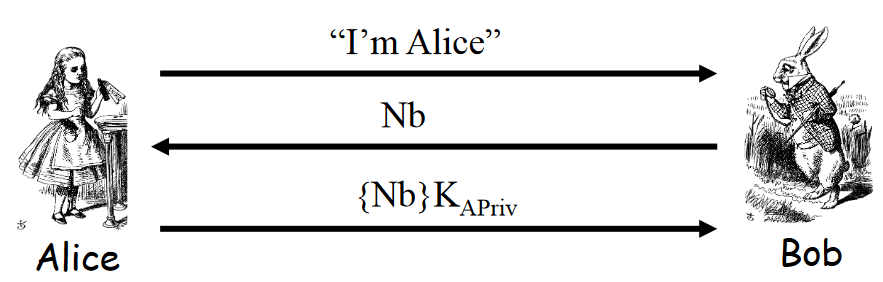
但是这样子没法实现mutual authentication：



如果使用公钥私钥的方法来进行签名的话：



这种情境下可以通过给A发送任意的值让A进行解密，因此加密和authentication使用的密钥一定要是不同的

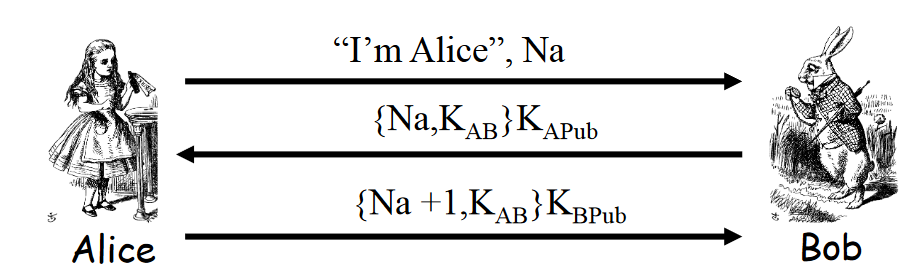


入侵者在这种情境下可以通过给A发送任意的值让A进行签名，解决办法还是加密和authentication使用不同的密钥

Session key：在对话持续过程中双方使用的symmetric key

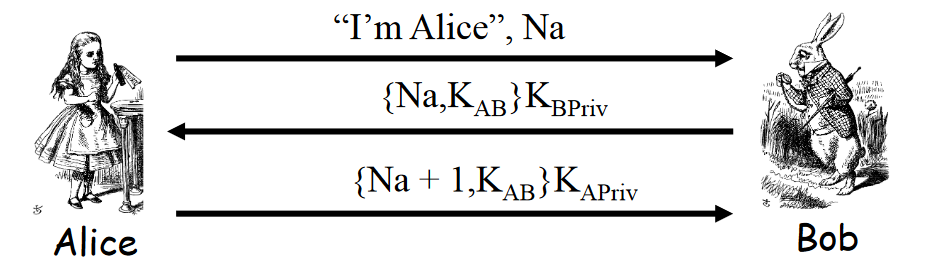
如何在确立session key的同时完成双方之间的互相认证：

第一种方式：



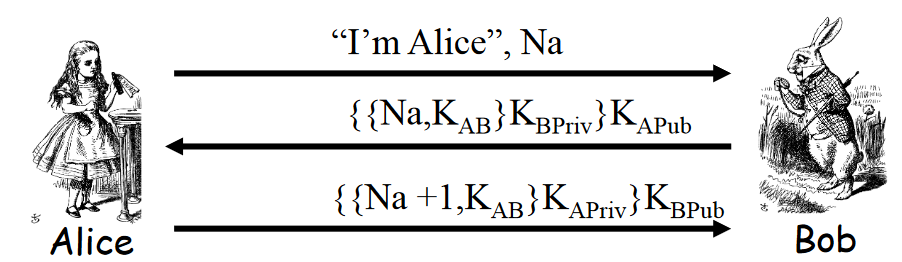
在这个过程中Nonce Na完全没有作用，只完成了对A的authentication，而B完全没有被认证；在这个过程中KAB相当于充当了一个Nonce的作用

第二种方式：



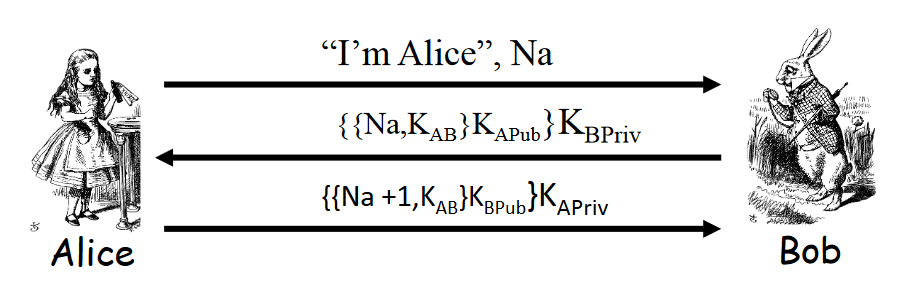
比较没用的方式，任何人都可以通过公钥来得到KAB，所以不能用

第三种方式：



可以在实现互相认证的同时得到一个安全的session key

第四种方式：



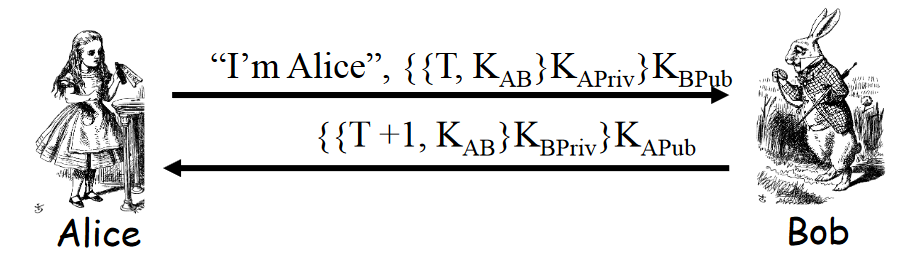
也可以在实现互相认证的同时得到一个安全的session key

但是所有人都能看见{Na, KAB}KAPub（虽然说并没有什么太大的隐患）

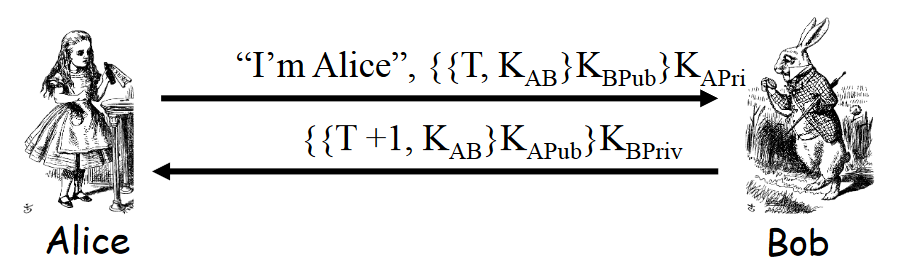
Timestamp：时间戳，通过记录当前的时间来避免replay attack，可以当成nonce使用

由于延迟的存在，需要允许一定的time skew

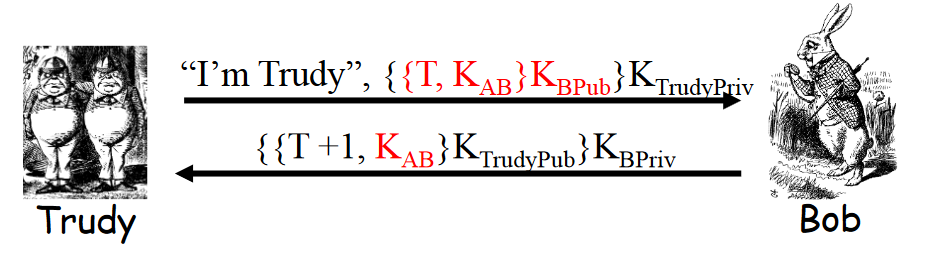
一种可行的方法：



另一种方法的尝试：



对于这种情况，Trudy可以在timestamp允许的skew内进行操作获取对称密钥：

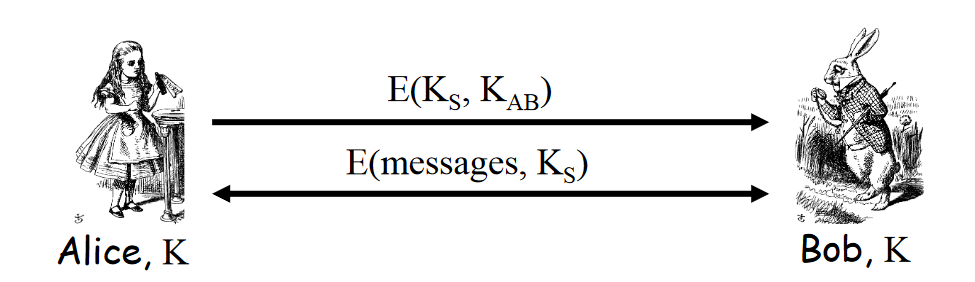


Perfect Forward Secrecy(PFS):

比如现在截取了某人发送的一段ciphertext，在一段时间后破解获得了Key然后用来获取这时候发送的信息，通过PFS可以避免这种攻击

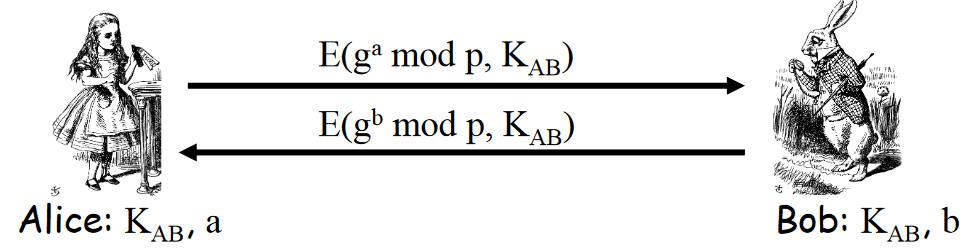
可以通过session key来实现PFS：

第一种是Naïve Session Key Protocol：



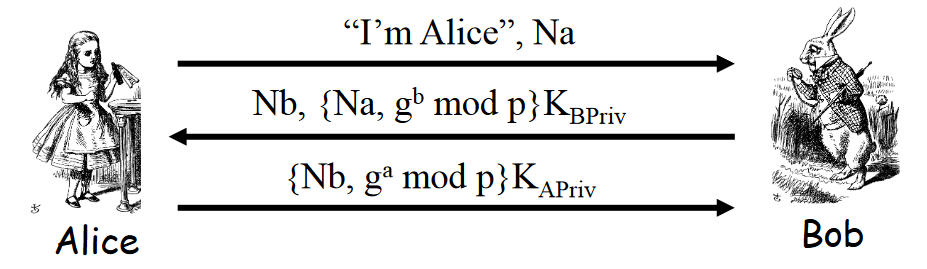
明显很呆 可以通过记录ciphertext并且随后破译来获取session key，从而破译message

假如使用Diffie-Hellman key distribution: 但是DH会受到MiM攻击，所以需要另外的策略



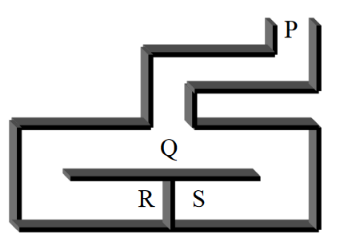
使用这种方法并且在完成session之后忘掉a和b，Session Key是无法重新被获取的

一种同时实现mutual authentication，Session Key和PFS的方式：



Zero Knowledge Proof(ZKP)：

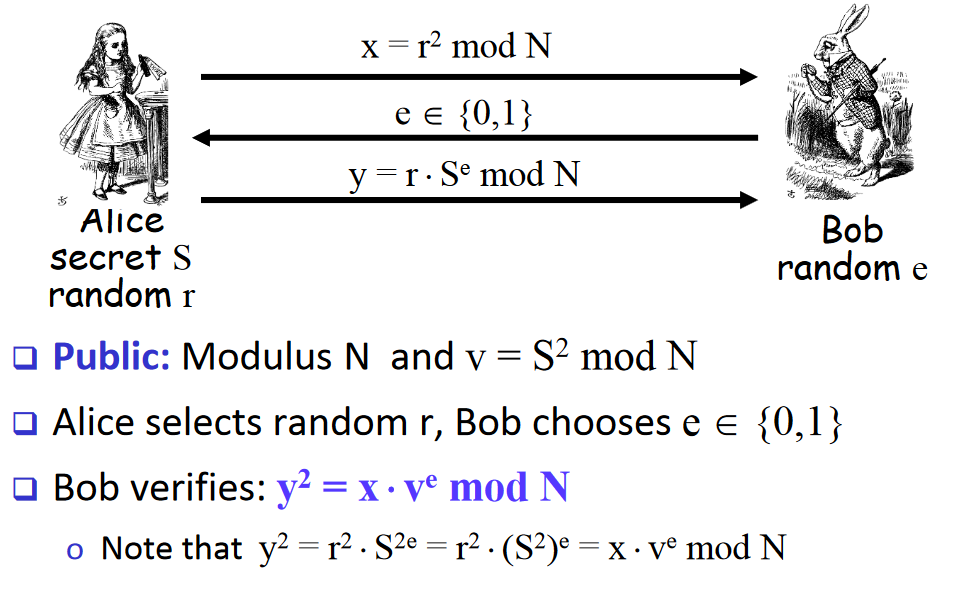
假设A想让B知道它有某个secret，但是B不知道关于这个secret的任何信息并且无法验证，因此ZKP实际上是一个随机的过程，有几率蒙混过关（较低）

比如在右图中，假设A在R或者S的某个位置，B通过在Q观察A出来的位置来判断A是否有打开RS之间的门的密钥，每次B要求A从左或者右中的位置出来，如果A没有这个密钥，则重复n次之后能够正确的几率会变得非常低，i.e. 1/2n

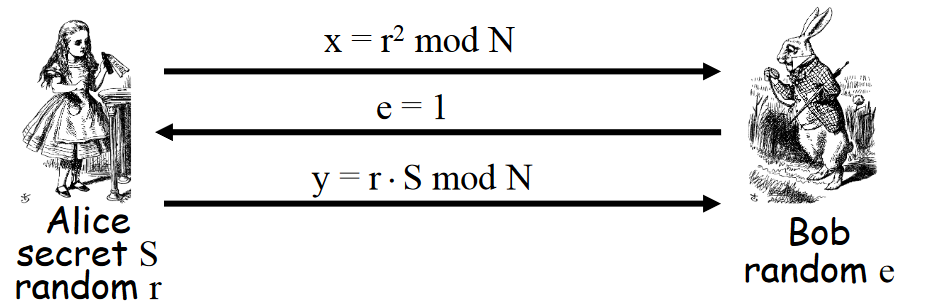
一个基于上面的cave的协议：Fiat-Shamir Protocol

假设N = pq，p和q为质数，N和v = S2 mod N是公开的，S是A的secret

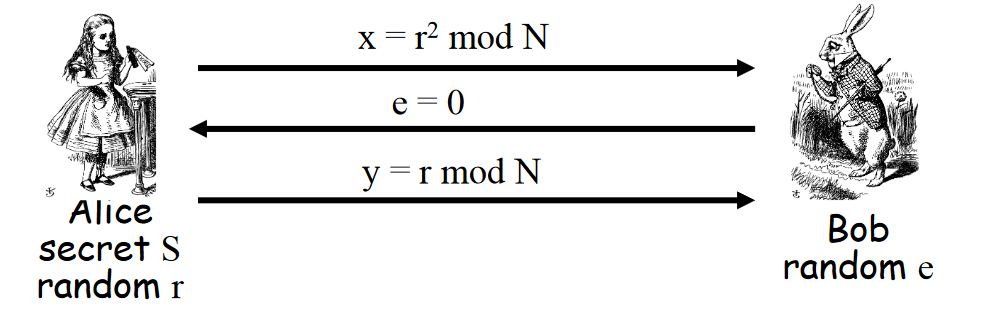
A需要在不揭露任何关于S的信息的情况下让B知道A知道S



e=1的情况：A在知道S的情况下给出正确的y，B可以得到正确的y2



e = 0的情况下：A不需要知道S就可以得到正确的y2



R必须频繁更迭，不然在知道r \* S mod N和r mod N的情况下可以破解出S

每次入侵者可以有1/2的概率猜对：假设预期的e是0，则在1和3步中分别发送X = r2和y = r，如果预期的e是1则在1和3步中分别发送X = r2 \* v-1和y = r，因此e一定是不可预测的

在实际计算中，会用一串随机的二进制流作为e，所以实际上入侵者全部猜对的概率实际上只有1/2n

破解的唯一方法是通过第一条的 X = r2 mod N求解出r然后得到S，但是开方求解非常难算，还不如直接去算v = S2 mod N

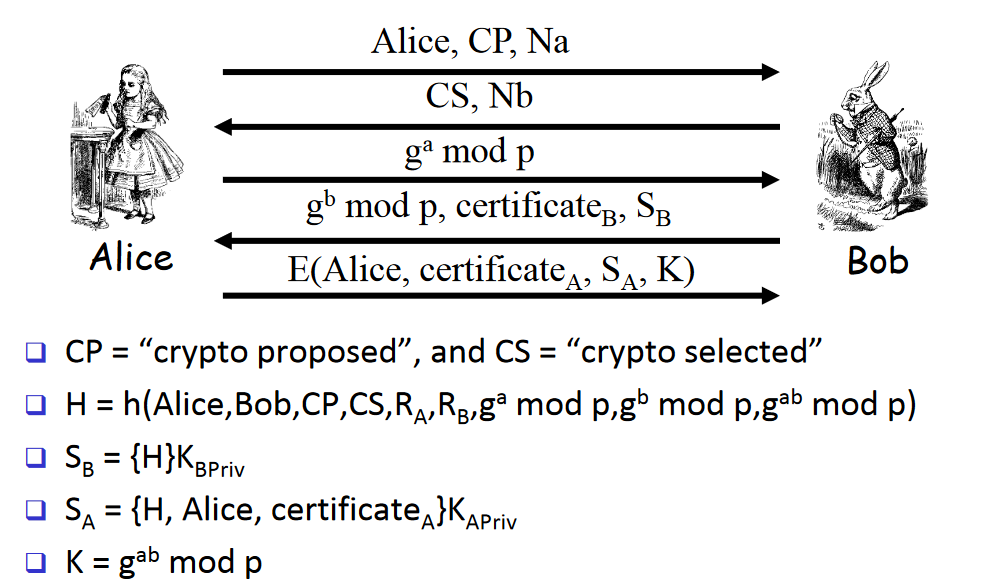
在实际运用中，ZKP还可以用来实现在隐藏身份的情况下的验证

真实世界中的协议：

SSH：Secure Shell

创建一个secure tunnel，通过这个安全的隧道发送不安全的command，一般应用在rlogin也就是远程登陆中，是相对简单的协议

SSH authentication可以通过Public Keys, Digital Certificates和Password实现，这里只讨论通过digital certificate实现：



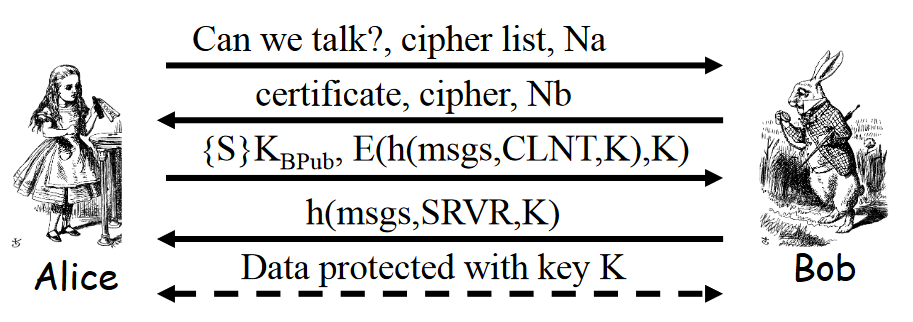
前两步中首先A发送自己的identity，希望使用的加密算法，key length等数据和nonce，B返回选择的加密算法参数和nonce

第三四步中A和B互相交换了DH value，SB和SA是经过签名的哈希值，B发送给A---B的certificate

最后A发送给B经过K加密过的签名的哈希值、身份和certificate

MiM attack不能成功，因为在SB的部分有哈希值的计算，如果上一步中入侵者将gt传给b，则返回的签名哈希值中的gt部分是不对的

SSL：Secure Socket Layer，通过Application layer和transport layer之间的socket层安全实现secured communication（HTTP协议和TCP协议之间）



Cipher list就是支持的cipher种类，A首先请求SSL链接，B返回希望选用的cipher类型和B的certificate；随后A向B发送使用B的公钥加密的S(pre-master secret，其实是一串随机生成的数字)和哈希值(CLNT为常数值)：这里的哈希值是为了验证前面发送的信息的integrity，B向A返回一串类似的哈希值：A通过计算收到的msg可以判断出B是否正确接收了之前的message，K = h(S, Na, Nb)，session key被成功建立

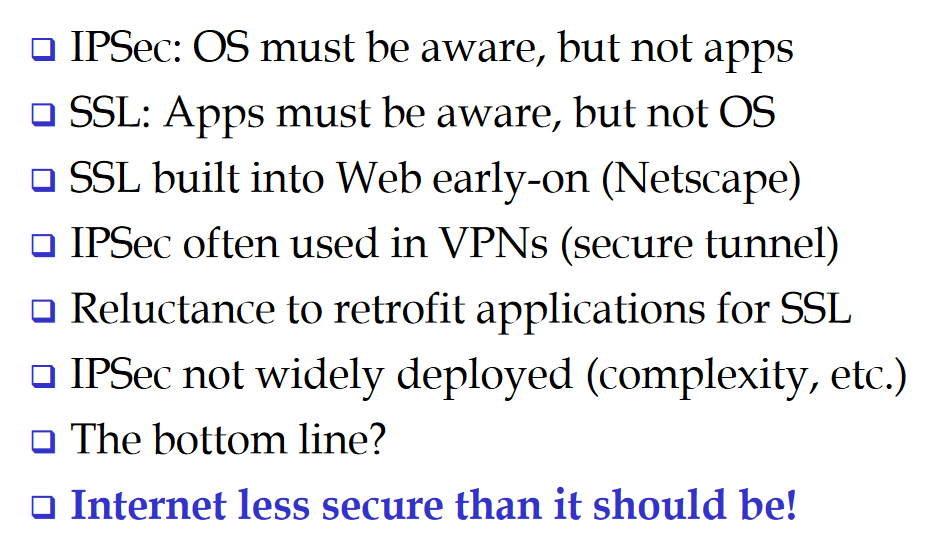
从K中可以得到6个keys：两个用来做encryption，两个用来做integrity，两个用来当成IV使用。一对key会被用来在不同的方向进行传输

在SSL中，这个协议只对B进行了认证，但是B其实并没有对A进行认证。主要原因是在网络环境下，一般A用户都会有一个valid certificate，B可以通过密码等别的形式对A进行认证，而不需要在第二步加上对A的certificate request

SSL通过由CA签发的B的certificate避免MiM attack，因为A一定会通过CA验证certificate，因此无法伪造这个证书

SSL session是非常消耗资源的，因为是public key operation，并且SSL是基于HTTP1.0的，因此会同时存在多个concurrent connections，因此一个session可能包含多个connections：为此，SSL提供了在已经存在session的情况下高效创建connection的方法

SSL和IPsec的对比：IPsec在网络层工作，相当于安全版的IP协议



GSM:

在最初的时候，手机通信是完全insecure的，最大的问题就是有cloning的存在：当进行通话的时候，identity是通过无线传输的明文发送的，因此极其容易被捕捉到，而且账单的计费是通过这个identity进行的，因此非常容易白嫖打电话。

4.Software:

由于大部分的安全都是由软件实现的，所以软件的安全也是至关重要的

软件的complexity和安全性基本上是成反比的，越复杂的代码一般来说bug越多

一般可以用5 bugs/10,000 LOC来进行估计，也就是说基本上一个3万个节点的网络有45亿左右的bug，其中10%会造成安全问题，再其中10%可以被远程利用，也就是45million security flaws

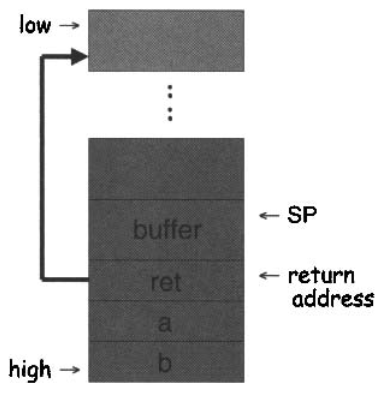
Program Flaws：比如StackOverflow等等，会造成program failure，可以从外界观测

一般分为三种：

Buffer Overflow：

在向服务器提交表单时如果input过长(假设不检查)则可能会对于没有权限的数据进行覆写

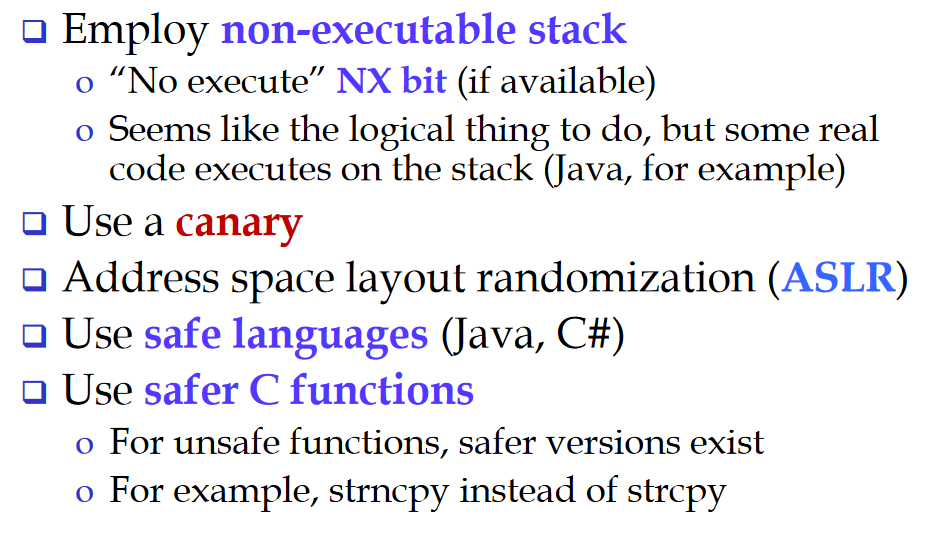
也可能导致程序在返回的时候返回错误的位置，因为Stack Pointer指向的位置是stack的顶端，buffer被放在栈的上方，如果overflow有可能会覆写return address



同理，可以通过code injection来让目标机器运行一段指定的代码

但是如果Trudy并不知道具体的代码的位置，那么如何实现精准的跳转就成为了问题：解决方案是在大概的return区间加上一大串的NOP代码，这样就会在执行完NOP后到达入侵者想要执行的代码部分。这种方式被称为stack smashing

可以使用下面的方法来防范stack smashing：



Canary---金丝雀，在ret的下面放置一个较小的integer，经常性地检查这一位来保证没有覆写发生，因为如果想要通过buffer overflow来进行覆写必定会经过这个位置(从上向下覆写)

ASLR：通过随机分配在内存中的地址来让buffer overflow attack变成概率事件，比如window vista使用了256-random layouts

Incomplete mediation：将sensitive data放置在exposed & uncontrolled condition下造成的问题，这也是导致buffer overflow的根本问题

Input validation---检测并确保输入是有效的，比如buffer overflow中检测输入的长度是否超出了buffer的长度，如果超出了则禁止输入

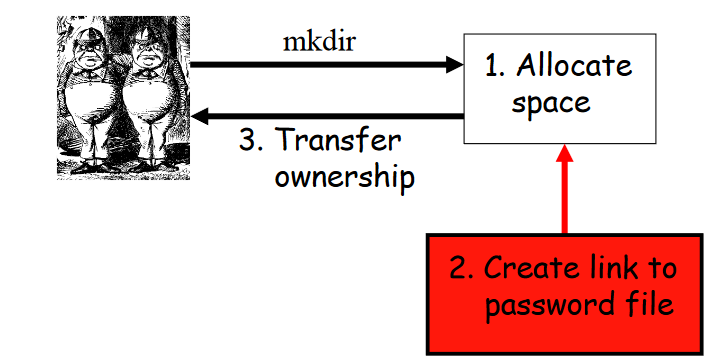
不仅要在client端进行input validation，在server端也要进行同样的操作，以防止在网络传输过程中的恶意篡改

Linux Kernel在input validation这方面就做的很好

Race conditions：

理论上讲所有的security operation都应当是原子操作，但是有些操作却不是atomic的，因此会造成一些问题

比如老版本的Unix的mkdir指令，首先创建一个文件夹并进行authorization，随后再transfer ownership：因此Trudy可以利用中间的空档期创建一个链接，依此获取对本来他不应该有访问权限的文件的拥有权

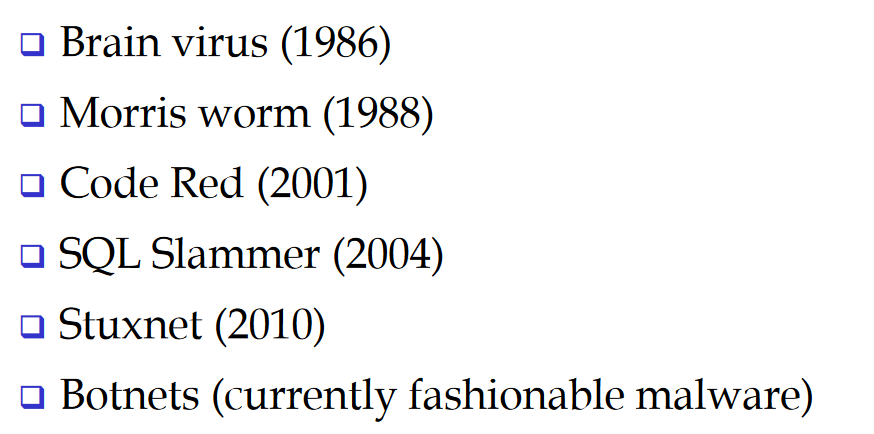


其实这里并不是真正意义上的race

虽然非常常见(比buffer overflow还常见)，但是对Trudy的timing要求及其严苛，因此能够藉此实现的攻击并不多

Malware---Malicious software

Malware可细分为很多种：virus---通过别的系统进行传播的恶意软件；worm---自行传播的virus；Trojan---通过伪装成正常的软件来执行恶意软件；trapdoor---获取unauthorized access to system；rabbit---通过耗尽系统资源进行攻击；spyware---通过键盘监听获取信息



这些Malware避免被检测到的方法：如果传输被中断，则删除所有代码；下载时加密下载的软件；编译后删除代码源文件；运行时不断地变更PID和名称

Malware detection：

通常有三种检测的方式：

signature detection---通过检测某些恶意软件特有的signature来鉴别，比如W32/Beast virus has signature 83EB 0274 EB0E 740A 81EB 0301 0000，错误检测的概率极其微小

优点是对于用户/管理员没有什么负担；并且对于已经发现的经典的恶意软件很有效

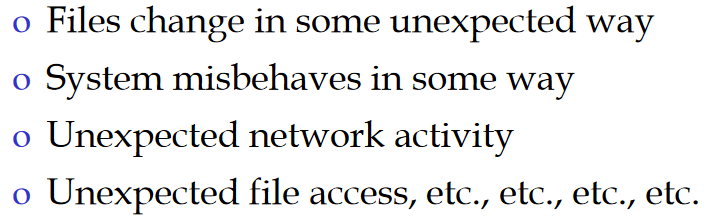
缺点是signature file可能很大，scanning process可能会很缓慢；signature必须持续更新；无法检测新的和某些高级的恶意软件

change detection---储存文件的hash value，周期性地重新计算哈希值并且对比以检测是否有变化；如果有变化，则可能被恶意软件感染

优点是基本上不会检测不到恶意软件的入侵；有可能检测到未知的Malware

缺点是文件是在经常变化的，可能会有很多误报；对于用户/管理员来说负担很大；检测到了啥也做不了，可能还是需要通过signature来进行检测

anomaly detection---检测一些异常的现象，比如：



但是需要首先定义normal behaviors，并且需要不断更新

优点是可能检测到未知的malware

缺点是没有可靠的记录；通过放慢某些异常的操作可以让其看上去变得正常；需要和别的方法结合使用

在IDS中也很常用(参见前面IDS的部分)

仍然是一个未解的难题，未来可能需要求助于AI？probably