ch21-网络安全(5)
——— IPsec & SSL

南京大学计算机系　黄皓教授

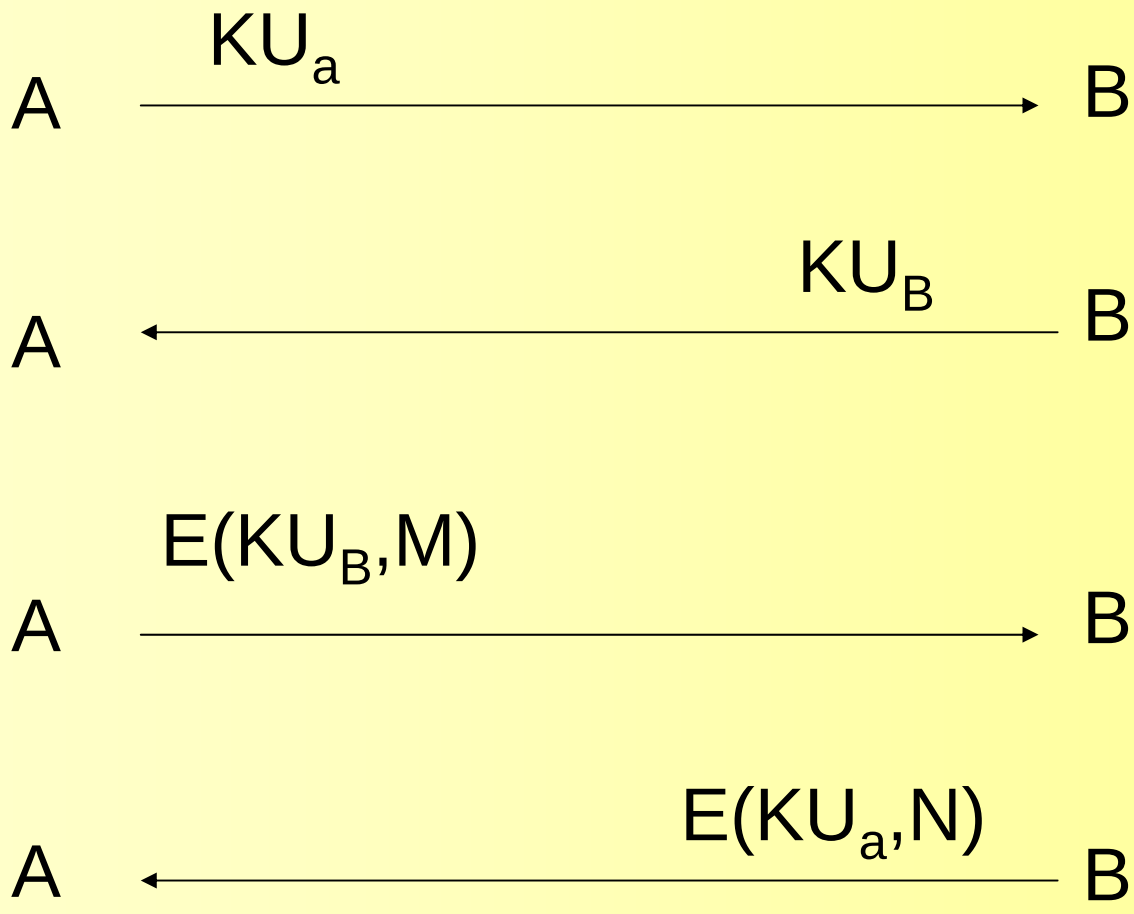
2007年12月21日星期五

1. 证书的基本概念

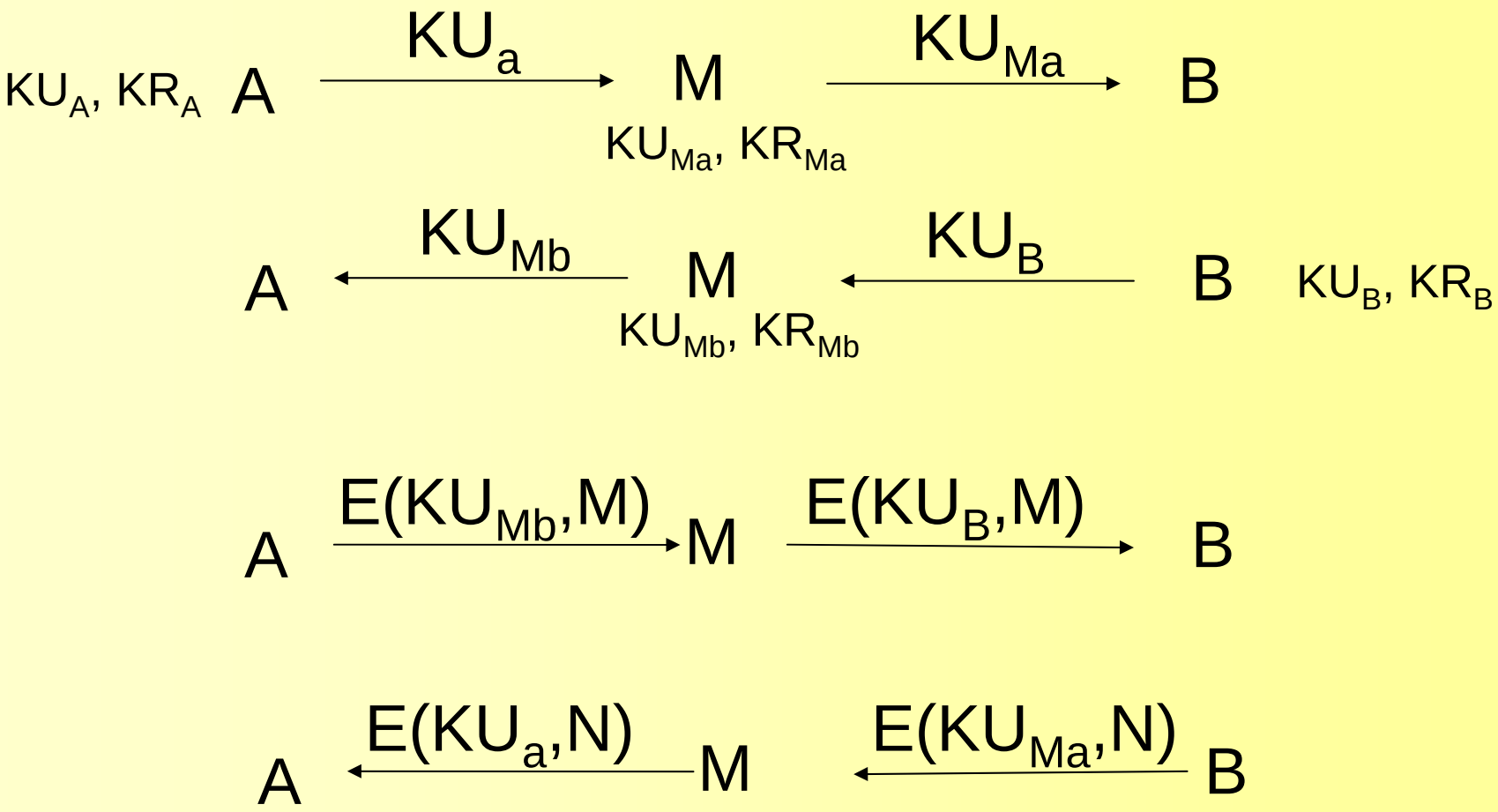
为什么需要公开密钥基础设施

- 中间人攻击
- 抵赖
- 信任

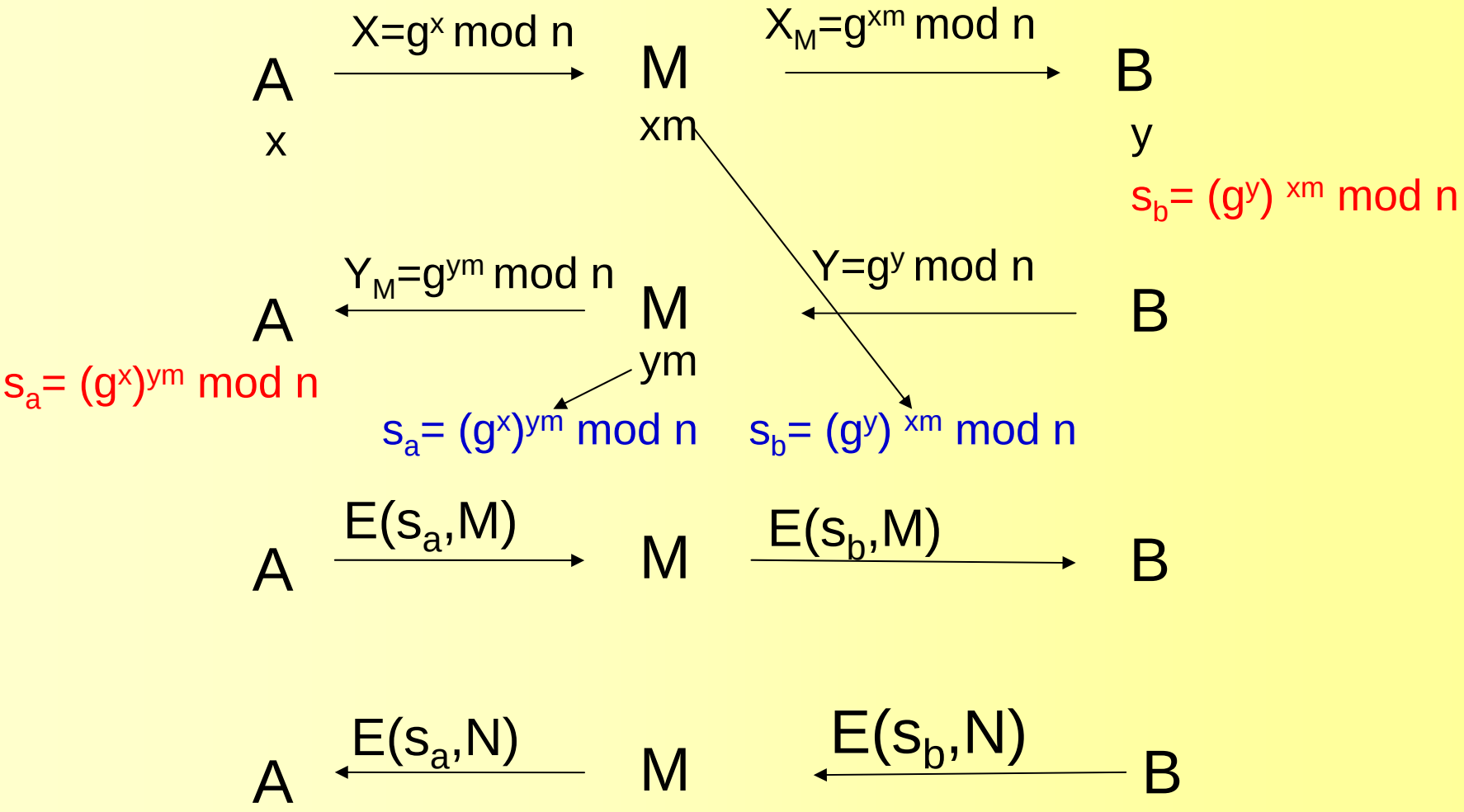
中间人攻击



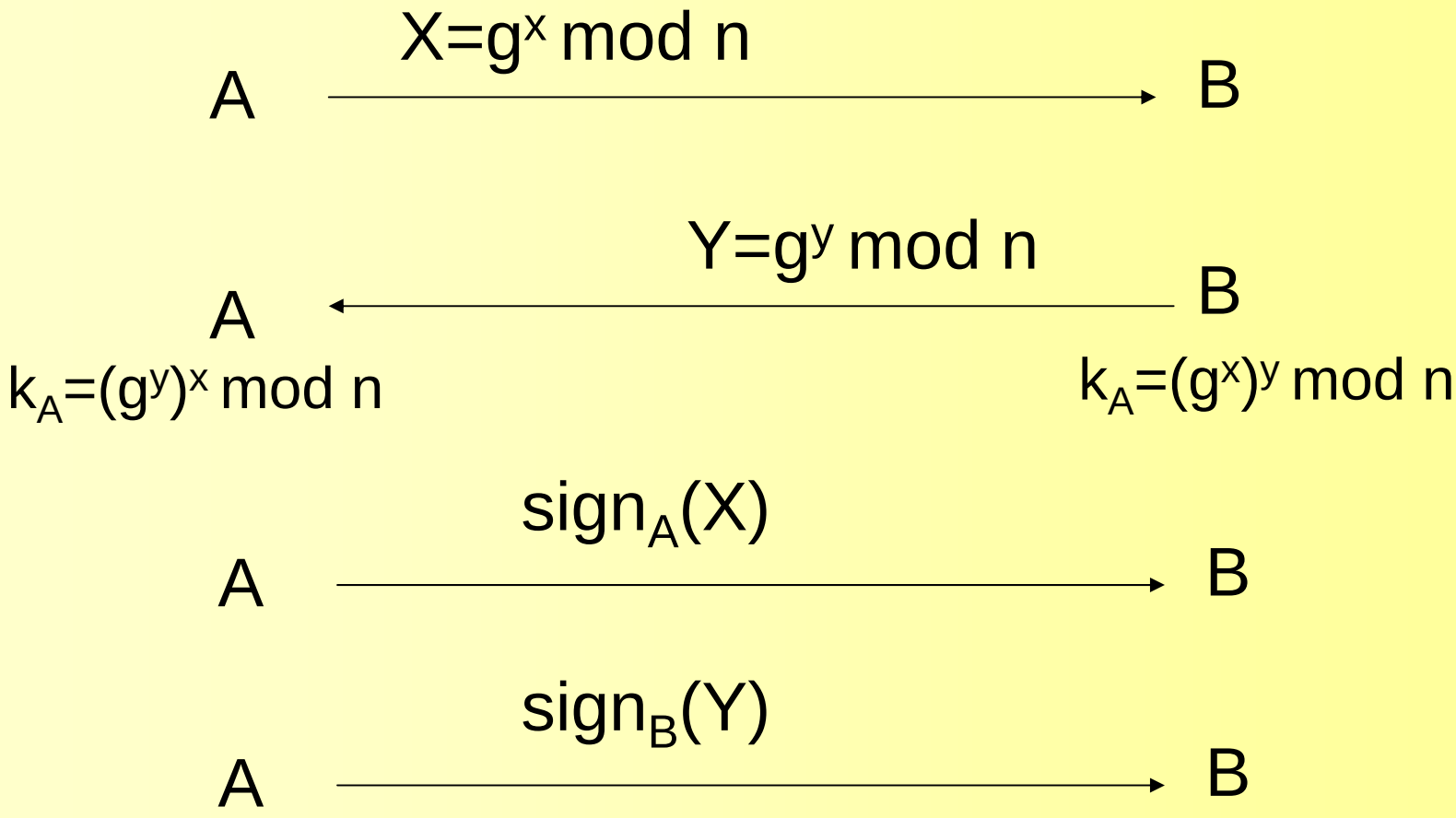
中间人攻击


KU_A, KR_A
A
KU_a
M
KU_Ma, KR_Ma
KU_Ma
B
A
KU_Mb
M
KU_Mb, KR_Mb
KU_B
B
KU_B, KR_B
A
E(KU_Mb,M)
M
E(KU_B,M)
B
A
E(KU_a,N)
M
E(KU_Ma,N)
B

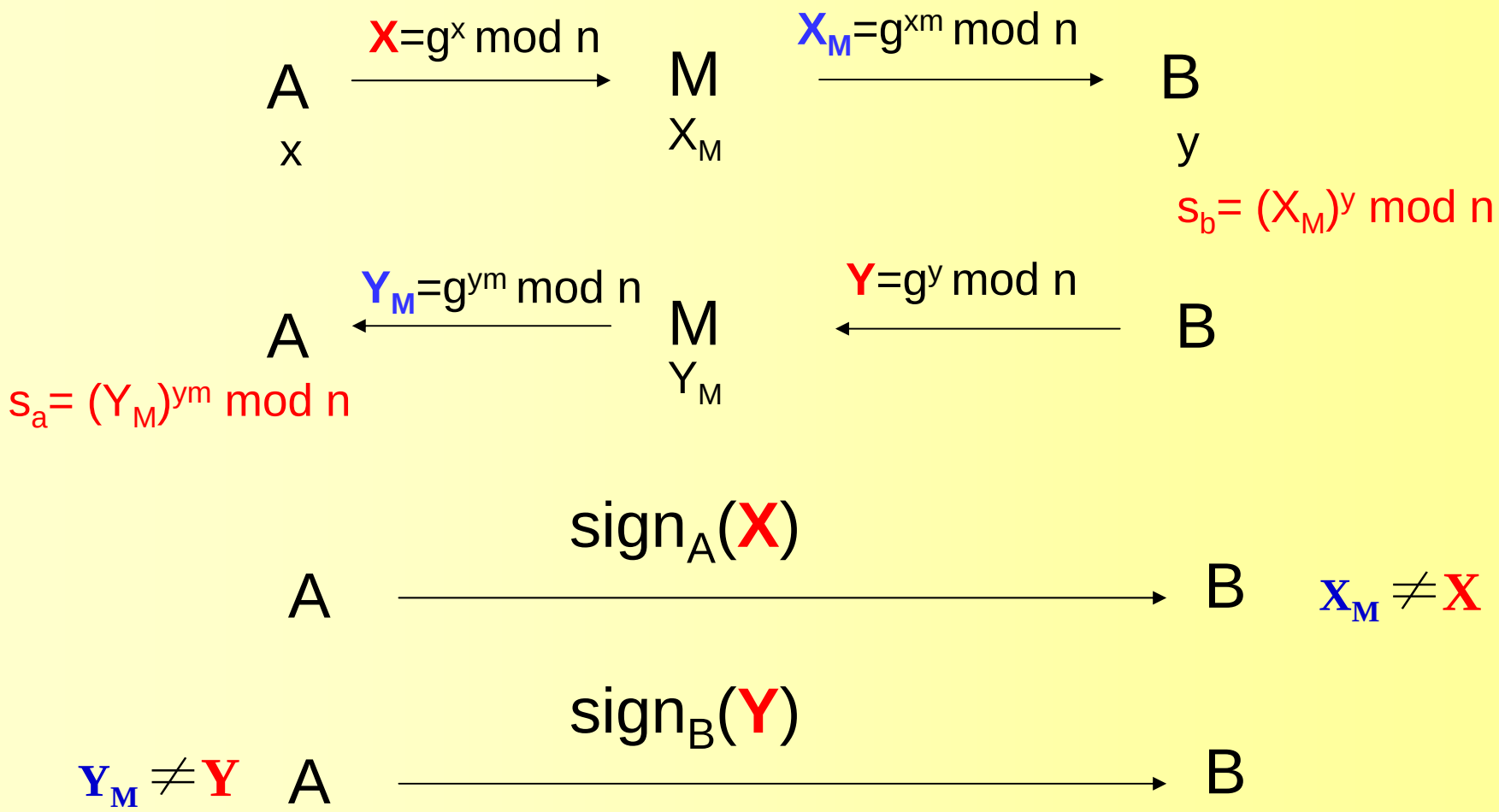
中间人攻击


A
x
X=g^x mod n
M
xm
X_M=g^xm mod n
B
y
s_b= (g^y) ^xm mod n
A
Y_M=g^ym mod n
M
ym
Y=g^y mod n
B
s_a= (g^x)^ym mod n
s_a= (g^x)^ym mod n
s_b= (g^y) ^xm mod n
A
E(s_a,M)
M
E(s_b,M)
B
A
E(s_a,N)
M
E(s_b,N)
B

中间人攻击


X=g^x mod n
A
B
Y=g^y mod n
A
B
k_A=(g^y)^x mod n
k_A=(g^x)^y mod n
sign_A(X)
A
B
sign_B(Y)
A
B

中间人攻击


X=g^x mod n
X_M=g^xm mod n
A
x
M
X_M
B
y
s_b= (X_M)^y mod n
Y_M=g^ym mod n
Y=g^y mod n
A
M
Y_M
B
s_a= (Y_M)^ym mod n
sign_A(X)
A
B
X_M ≠ X
sign_B(Y)
Y_M ≠ Y
A
B

抵赖

- A当面将自己的公钥交给B;
- A用自己的私钥签署一份合同交给B;
- B用A的公钥验证合同签名的有效性;
- B按照合同活动;
- A拒绝履行合同中的承诺
  - 声称他没有签署过合同;
  - B用A的公钥验证合同的签名;
  - 那把公钥不是自己的
  - 那把私钥合同之前就丢了，是别人冒充他签的。
- B ? ? ? ? ? ? ?

信任

- B 在一个网站上下载了A的公钥;
- A用自己的私钥签署一份合同：采购B机器，并把合同在网上发送给B;
- B用A的公钥验证合同签名的有效性;
- B交付机器;
- B向A索要货款，但B收不到回音，也不知道A住在什么地方;

- B思考:
  - 网站和A在合伙欺骗自己吗?
  - A的私钥是在签署合同之前丢失的吗?
  - 网站会在A丢失私钥之后就立即负责任地告诉我吗?
  - 如果找到了A，法院能根据签名合同判决吗?
  - 网站的工作人员能准确判断申请者的身份而不被欺骗吗?
  - 如果网站也是被欺骗的，谁对找不到A而对A造成损失负责?

PKI提供的服务

- 好密钥的安全生成
- 初始身份的确认
- 证书的颁发、更新与终止
- 证书有效性的检查
- 证书和相关信息的分发
- 密钥的安全存档和恢复
- 签名和时间戳的产生
- 信任关系的建立和管理

公钥基础设施的组件

- 认证机构
- 注册机构
- 证书服务器
- 证书库
- 证书验证
- 密钥恢复服务
- 时间服务器
- 签名服务器

公钥证书

- 在电子化世界里
  - 证书是一个由使用证书的用户群所公认和信任的权威机构签署了其数字签名的信息集合。
- 公钥证书
  - 在该类证书中，一个公钥值与一个特定的实体（人、角色、设备或其他）权威地联系在一起;
  - 公钥证书是由称为证书权威机构（认证机构）的人或实体在确认了相应的私钥持有者的身份或其他属性后用数字签名的方式将一个公钥值与这个私钥持有者权威地联系在一起;
  - 权威机构承诺与公钥配对的私钥的唯一持有者是公钥的主体。
  - 公开密钥技术和数字签名是电子商务安全的基础
  - 那么公钥证书则是将这些技术广泛地应用于大型的、全球性的电子商务社区的关键。

公钥证书的意义

- 公钥证书的使用功能
  - 消息发送
    - 当某一消息发送方希望使用基于公开密钥技术的加密方法来发送加密信息时，该消息的发送方必须拥有每一个接收方的公钥拷贝。
    - 如果用的不是接受方的公钥(可能攻击者替换了)，则会造成信息泄漏。
  - 验证数字签名
    - 任一方想要验证另一方的数字签名时，验证方也需要拥有签名方的一个公钥拷贝。
    - 如果用的不是真正的声称的签名者的公钥，则会发生欺骗事件。
    - 如果不能证实数字签名发生时刻，私钥在证书的主体手中，则公钥的主体可以不承认做了签名，可能是事实，也可能是抵赖。

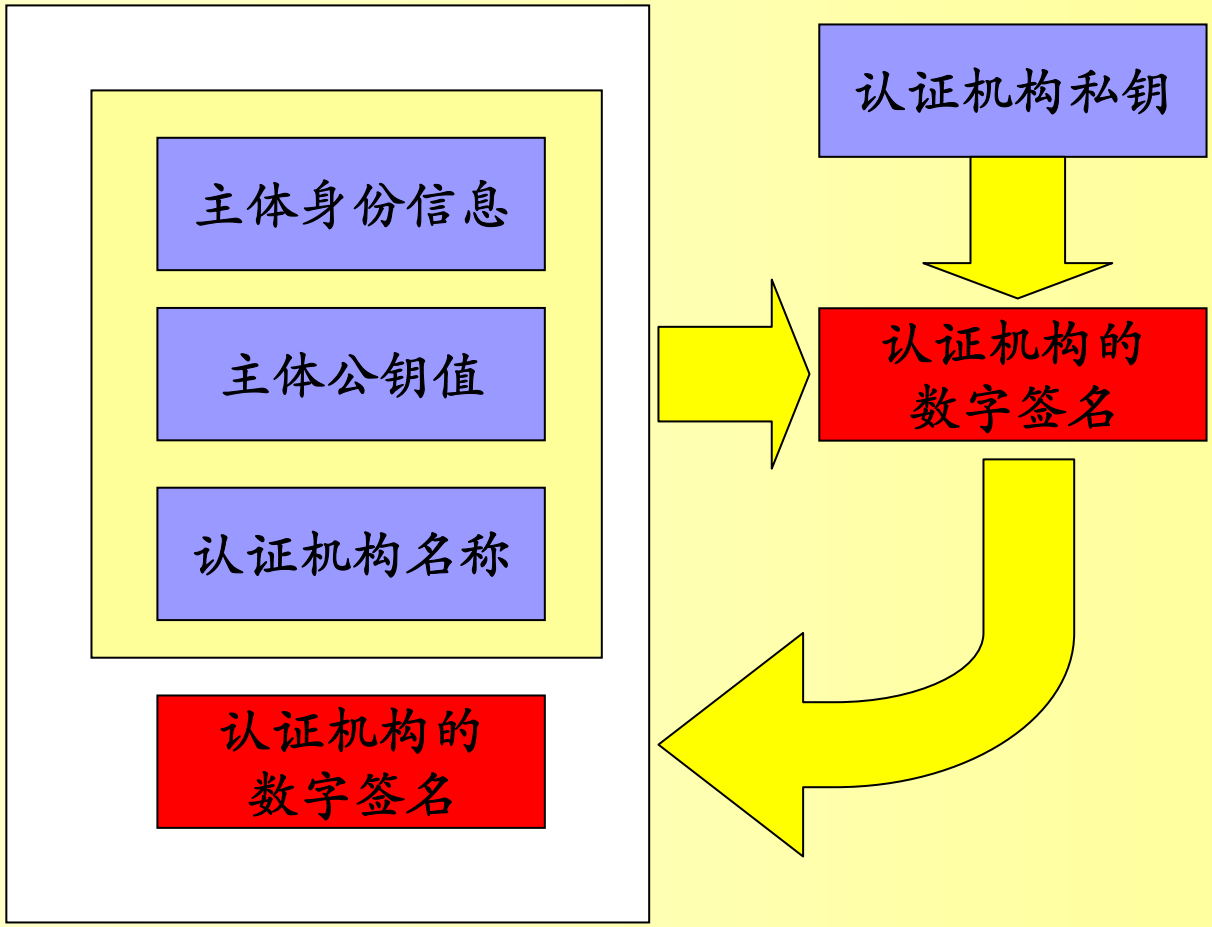
1. 消息的发送者必须知道
   - 公钥没有被第三方替换
   - 私钥没有泄漏
2. 数字签名的验证者必须确认证书的主体（对方，即协议的参加方）不能否认签名的发生时刻，私钥在他手中。
   - 密钥对是对方申请的
   - 私钥仍然在对方手中（没有泄漏

证书机构提供
验证证书主体的服务

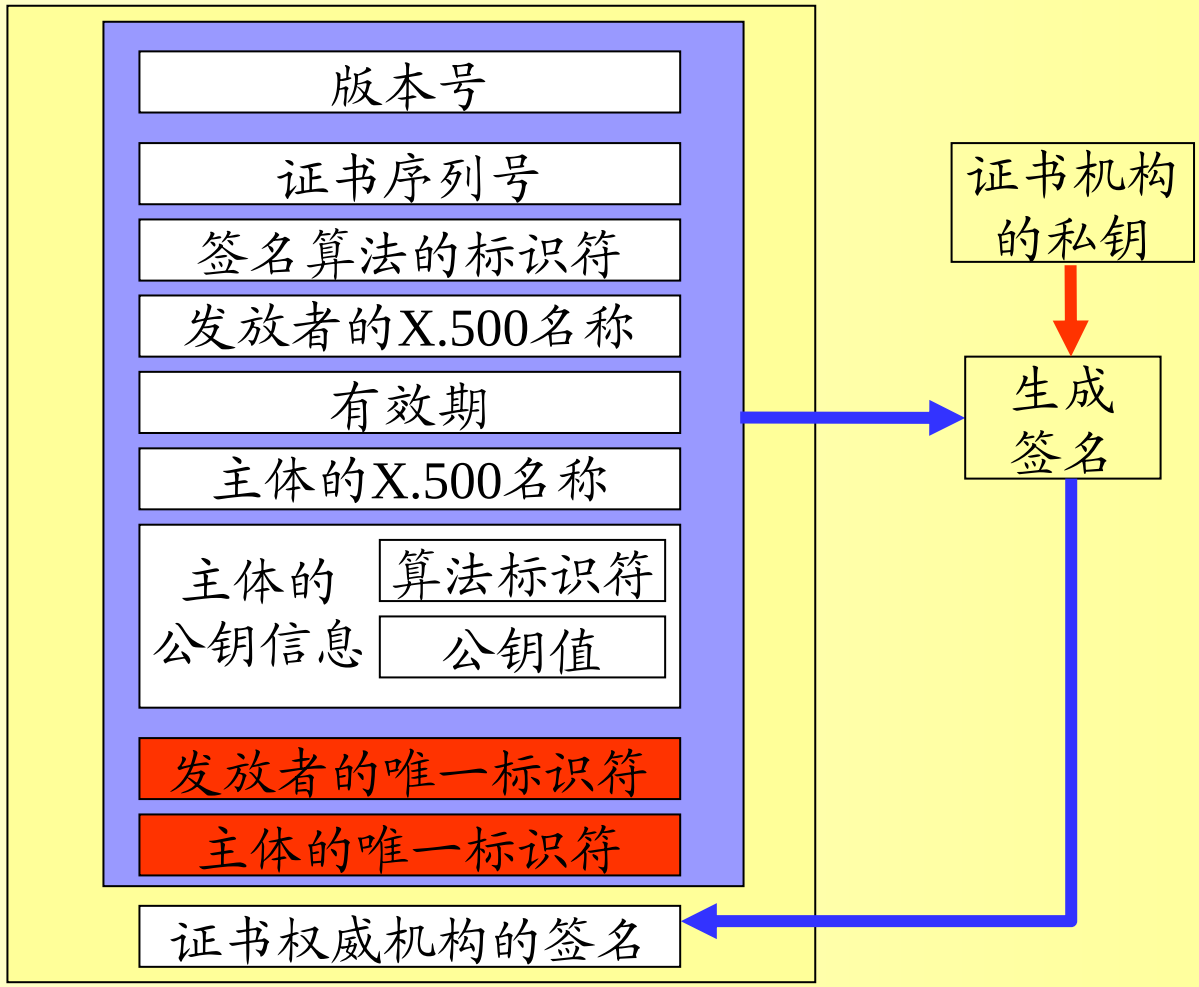
证书机构提供
证书撤销、档案等服务

证书机构提供的是特种服务，资质需要严格管理

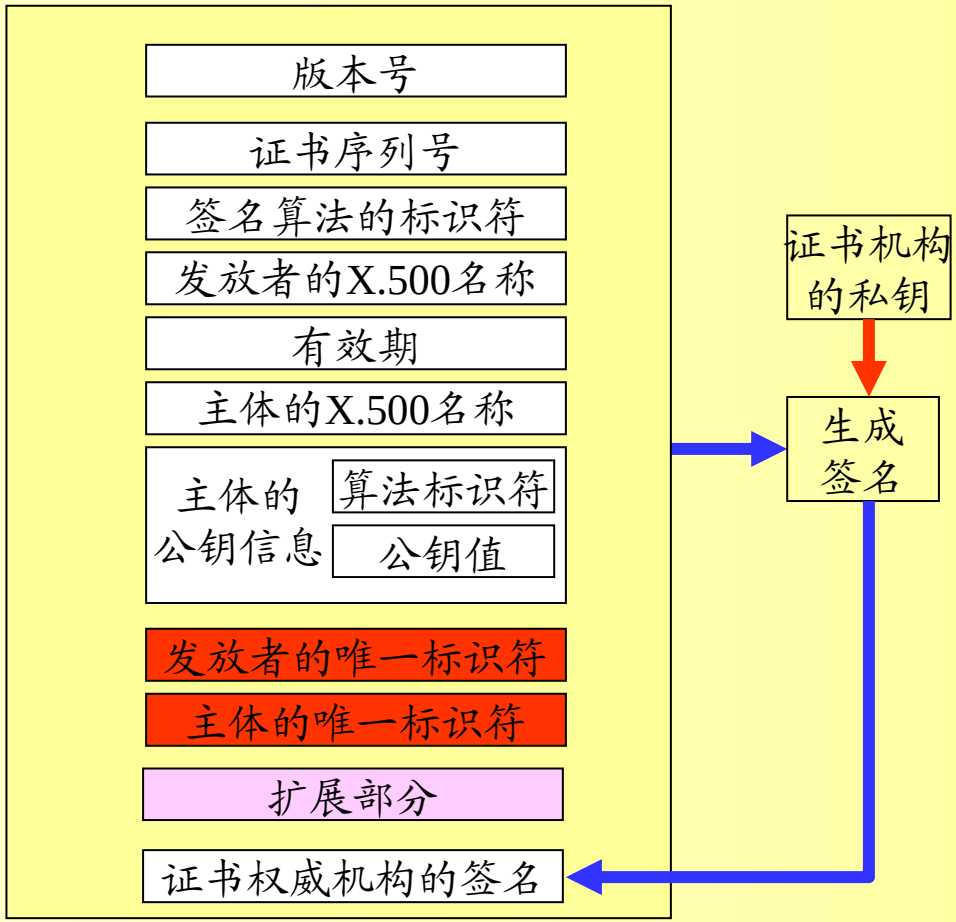
公钥证书的结构



证书 — v1&v2证书格式


版本号
证书序列号
签名算法的标识符
发放者的X.500名称
有效期
主体的X.500名称
主体的公钥信息
算法标识符
公钥值
发放者的唯一标识符
主体的唯一标识符
证书权威机构的签名
证书机构的私钥
生成签名

证书的扩展项 — X.509 证书格式v3



公钥证书系统的三类实体

- 认证机构
  - 为公-私密钥对的持有者发放证书。
  - 提供各种服务。
- 证书主体
  - 持有相应私钥的个人、设备或其他实体。
  - 当证书的主体是个人或法人实体时，一般将这类持有证书的主体称为证书机构用户。
- 证书用户
  - 利用证书验证签名；
  - 利用证书发送机密消息。

证书的使用

- 证书用户(公钥用户)
  - 获取证书中的公钥，按照指定的用途使用该公钥的用户；
- 证书用户信任证书权威机构
  - 证书用户相信证书权威机构证实的证书的公钥是某个实体的，即：与这个公钥相对应的私钥只有这个实体拥有；
- 证书用户要验证证书的真伪
  - 证书用户用证书权威机构的公钥验证验证获得的证书是否是他信任的证书机构发放的；
  - 证书用户验证证书是否过期了；
- 证书用户必需安全地获得证书权威机构的签名公钥。

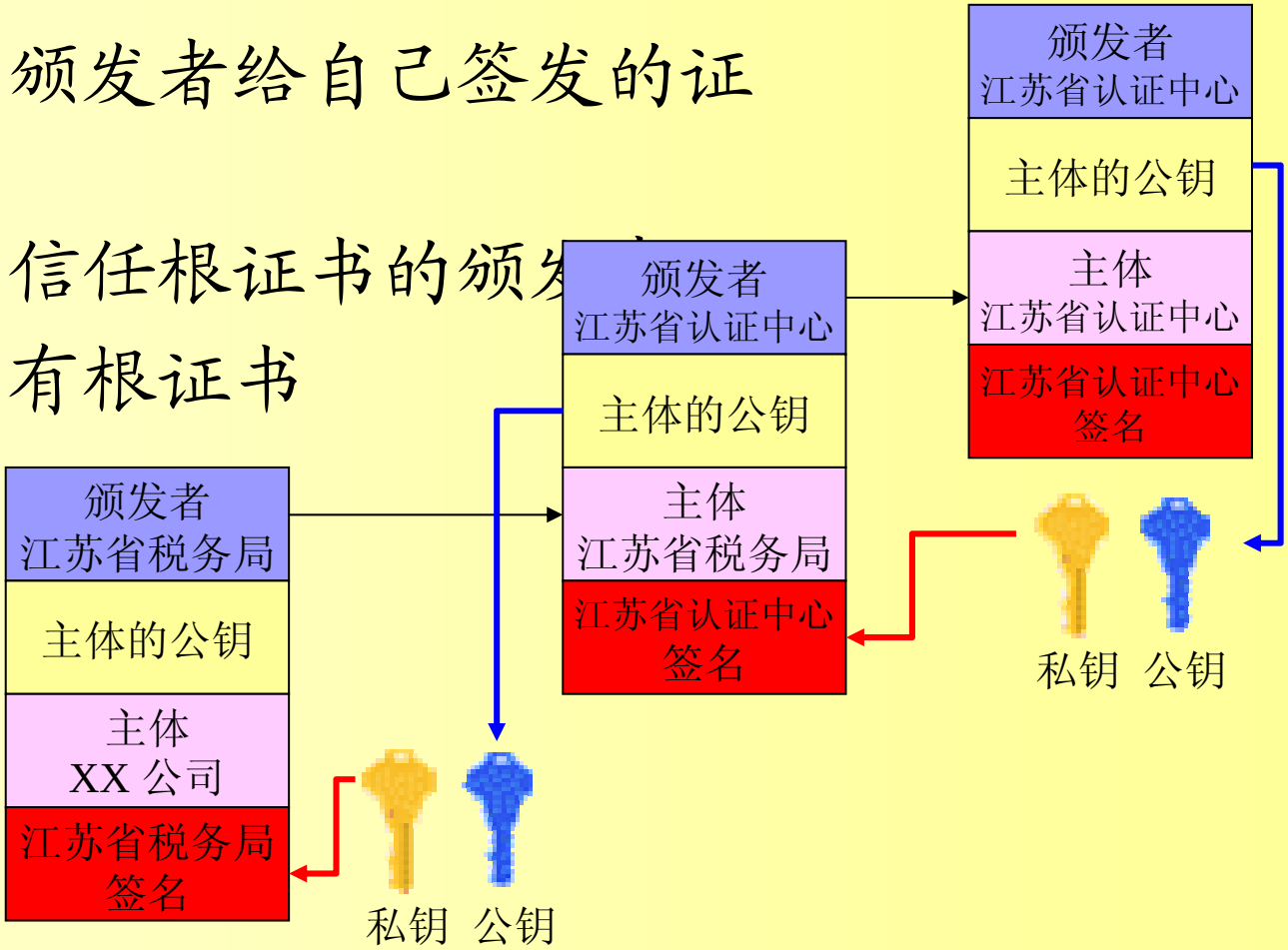
- 公钥证书可以公开的的文件服务器、目录服务系统以及未提供安全性保护的通信协议来分发。
- 使用证书好处主要在于，公钥用户只要知道认证机构的公钥，就可以获得其他很多协议参与方的公钥。因而可以获得很好的规模效应，也就是说，只要能运用公钥技术，就可以推广公钥证书的运用。
- 要注意的是，只有当公钥用户相信认证机构的目的仅仅是发放证书时，证书才是有用。

证书的验证

- 证书的主体
- 证书是否在有效期内
- 证书的签名是否正确
- 证书当前是否可用（是否被撤销）
- 证书的用途是否正确

证书的认证 — 认证路径

- 根证书：颁发者给自己签发的证书
- 证书用户信任根证书的颁发
- 证书用户有根证书



证书 — 法律关系

- 服务对象
  - 专门为某个社团的内部人员或其附属人员提供服务，如一个组织的内部认证机构可能只管理该组织中雇员的证书;
  - 认证机构为包括非附属人员在内的广大社区提供服务;
- 法律关系
  - 纸上签名与一个特定的人具有内在的联系，因为它们留有签名者的唯一笔迹;
  - 一个用于数字签名的密钥对不与任何人具有内在的联系;
  - 这种联系必须由认证机构确定了持有特定密钥对的人的身份后建立;
  - 利用私钥创建的每一个数字签名的可靠性部分地依赖于将该私钥与签名者相联系的认证机构的可靠性;
- 原则
  - 只有证书主体才能控制相应的私钥;
  - 证书主体必须经过适当的验证

2. 认证协议导引

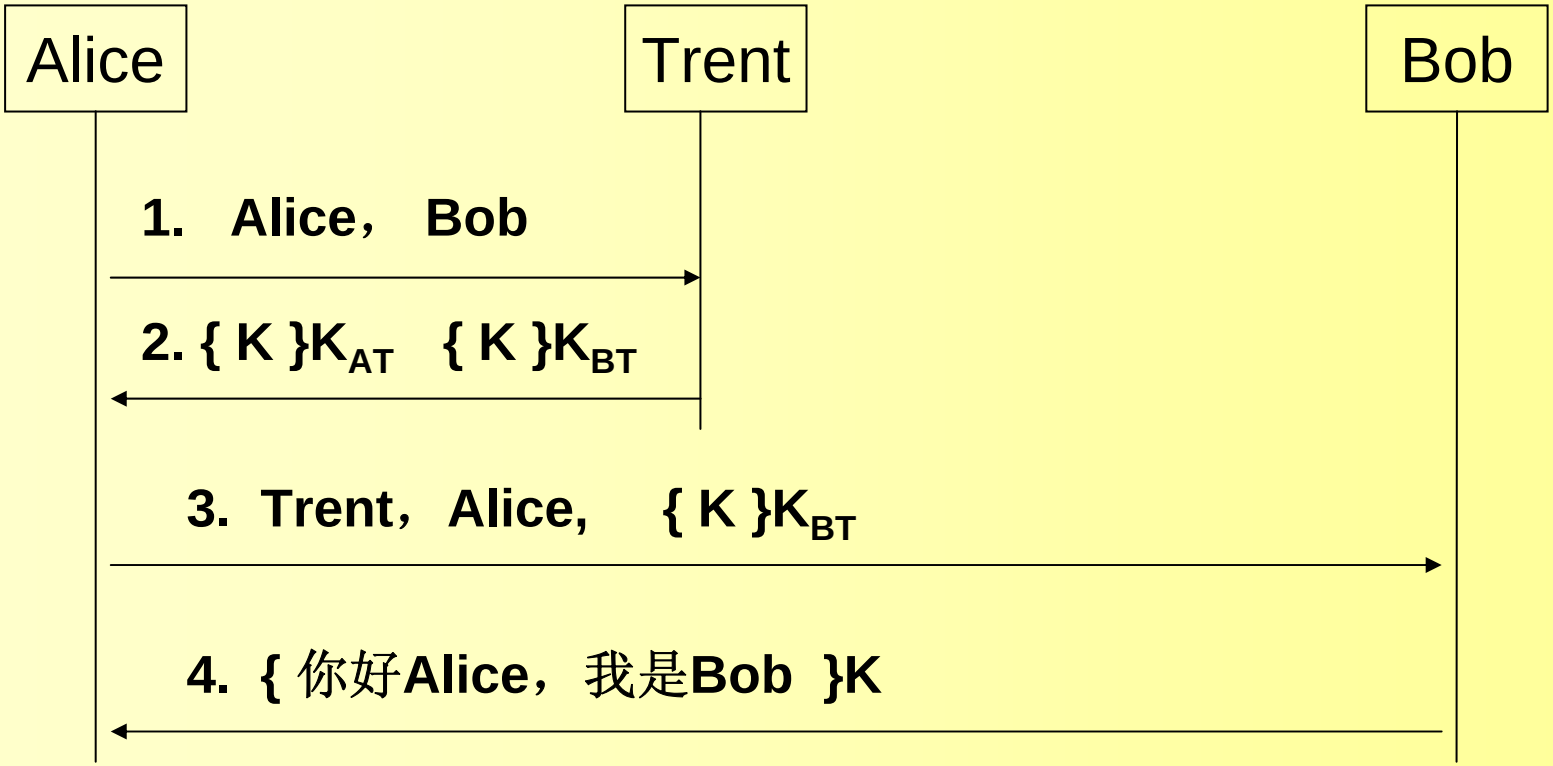
符号约定

- A：Alice，B：Bob，T：Trent
- K_{AT}：Alice与Trent的共享密钥;
- K_{BT}：Bob与Trent的共享密钥;
- { M } K：用密钥k加密消息M。
- $E_k(M)$，$D_k(M)$， E(k, M)，D(k, M)

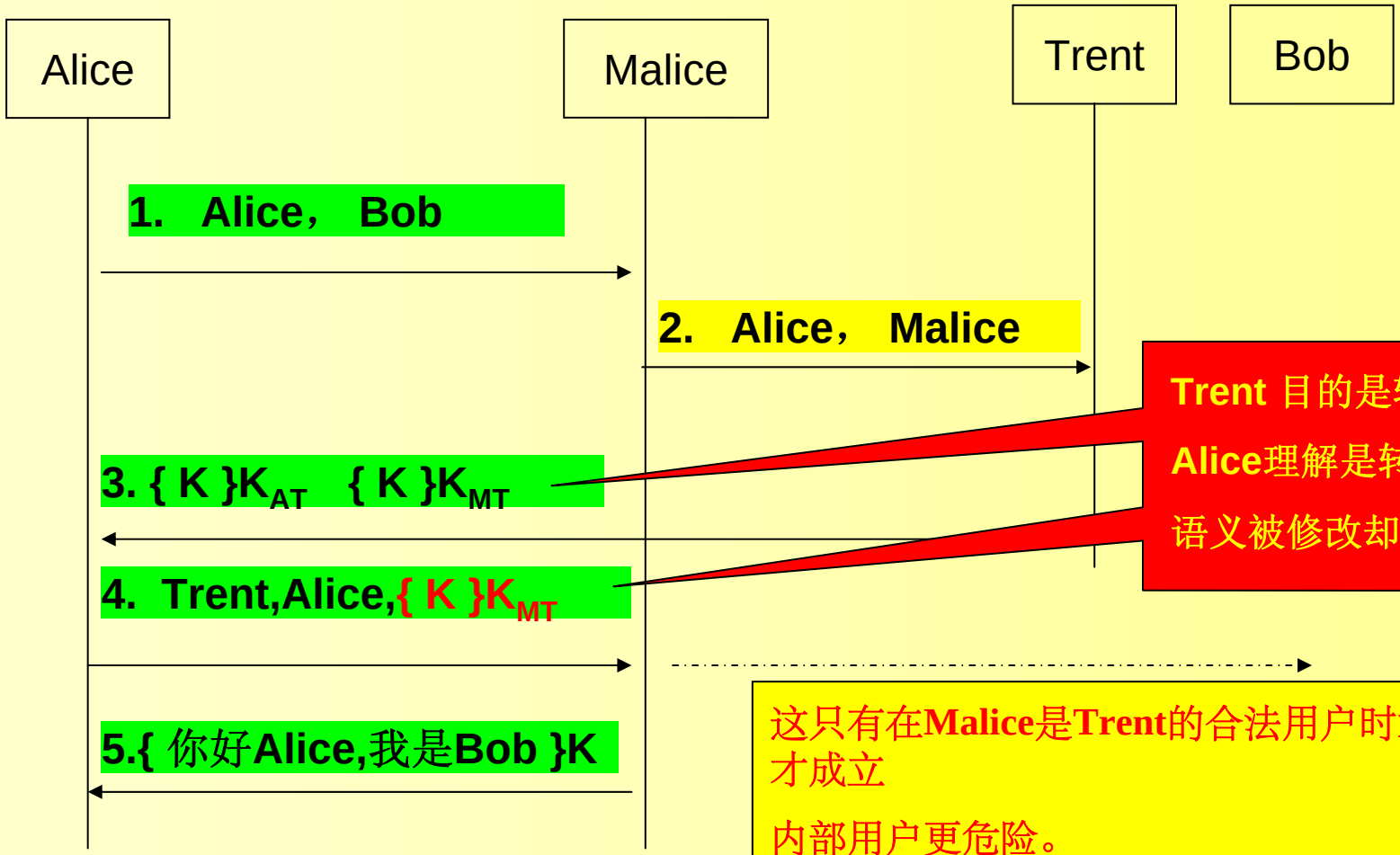
- K_U, K_R,
- $K_{U,Alice}$，$K_{U,Bob}$，$K_{R,Alice}$，$K_{R,Bob}$

- [M] K = M || HMAC(K, M)

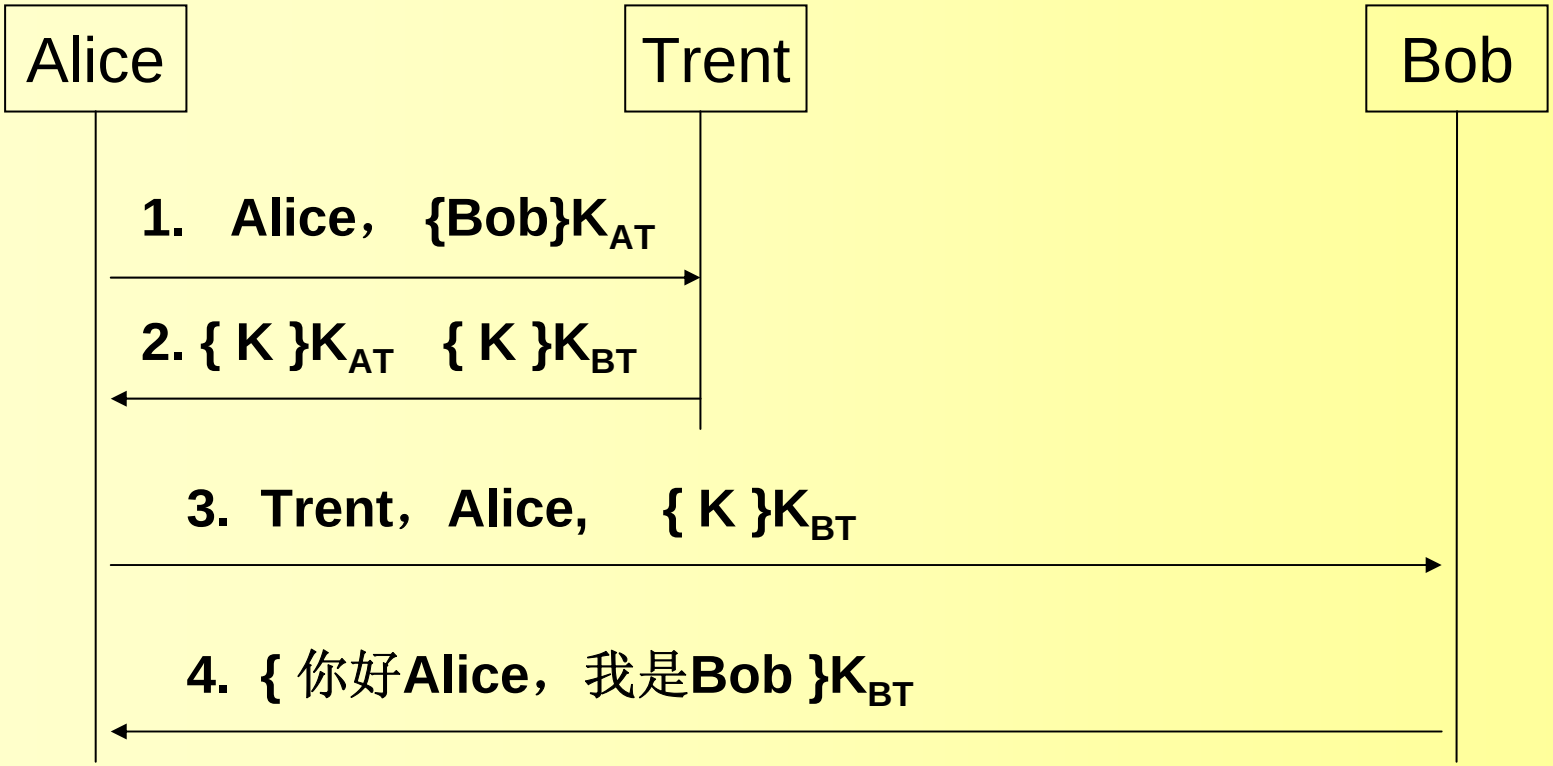
协议1

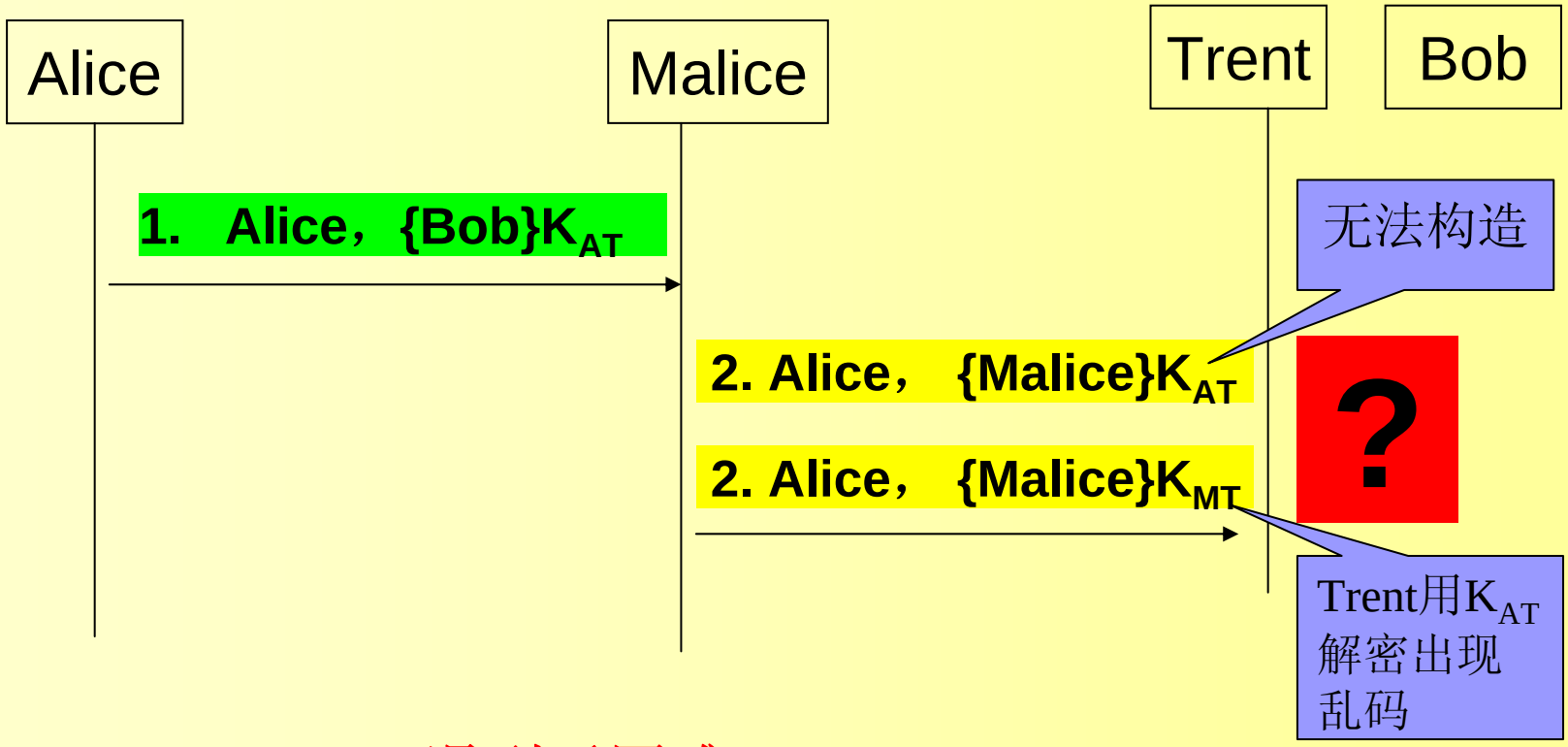

Alice
Trent
Bob
1. Alice， Bob
2. { K }K_{AT} { K }K_{BT}
3. Trent，Alice， { K }K_{BT}
4. { 你好Alice，我是Bob }K

攻击1 — 对协议1的一个攻击



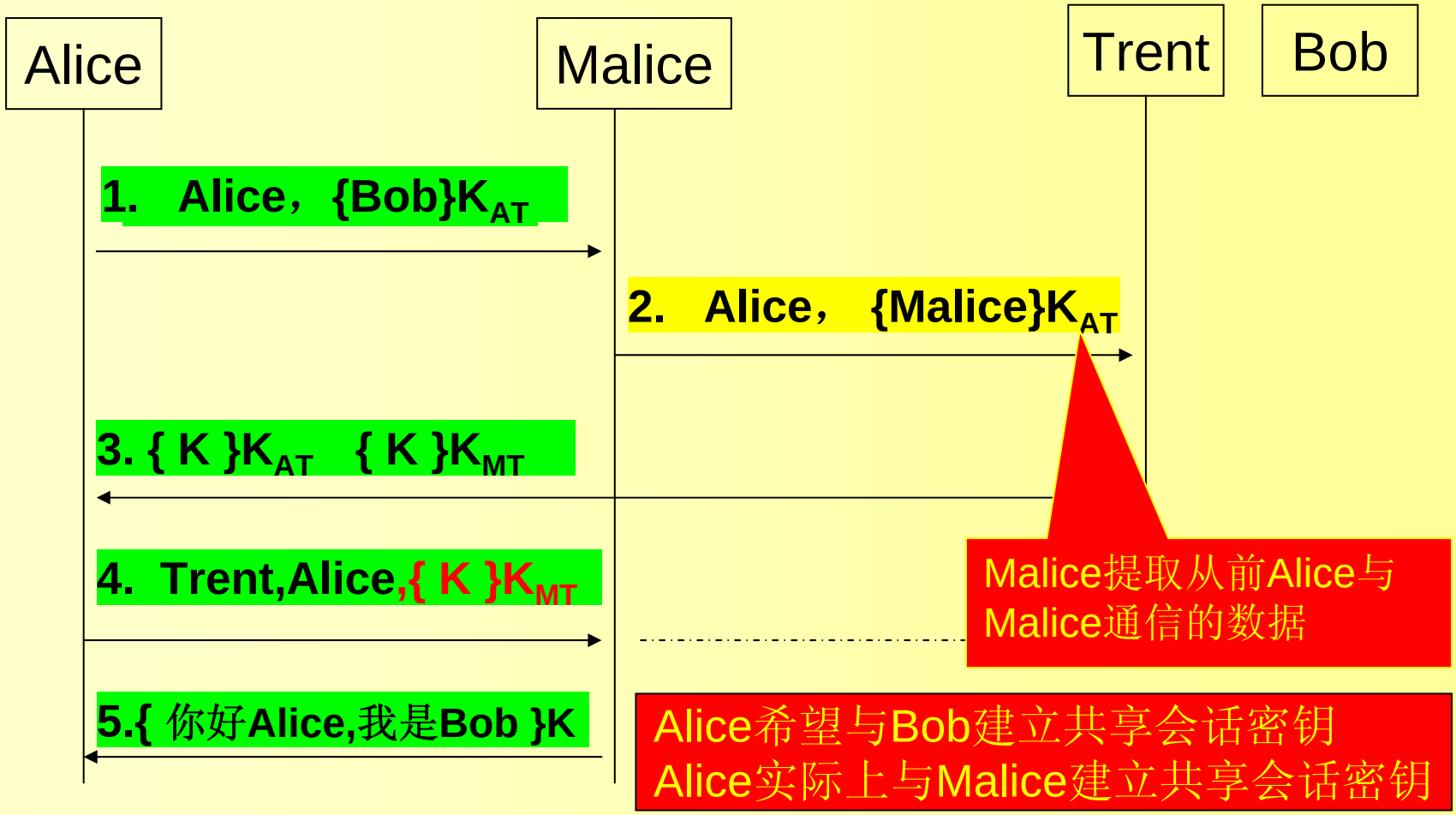
协议2 — 对协议1的一个修改



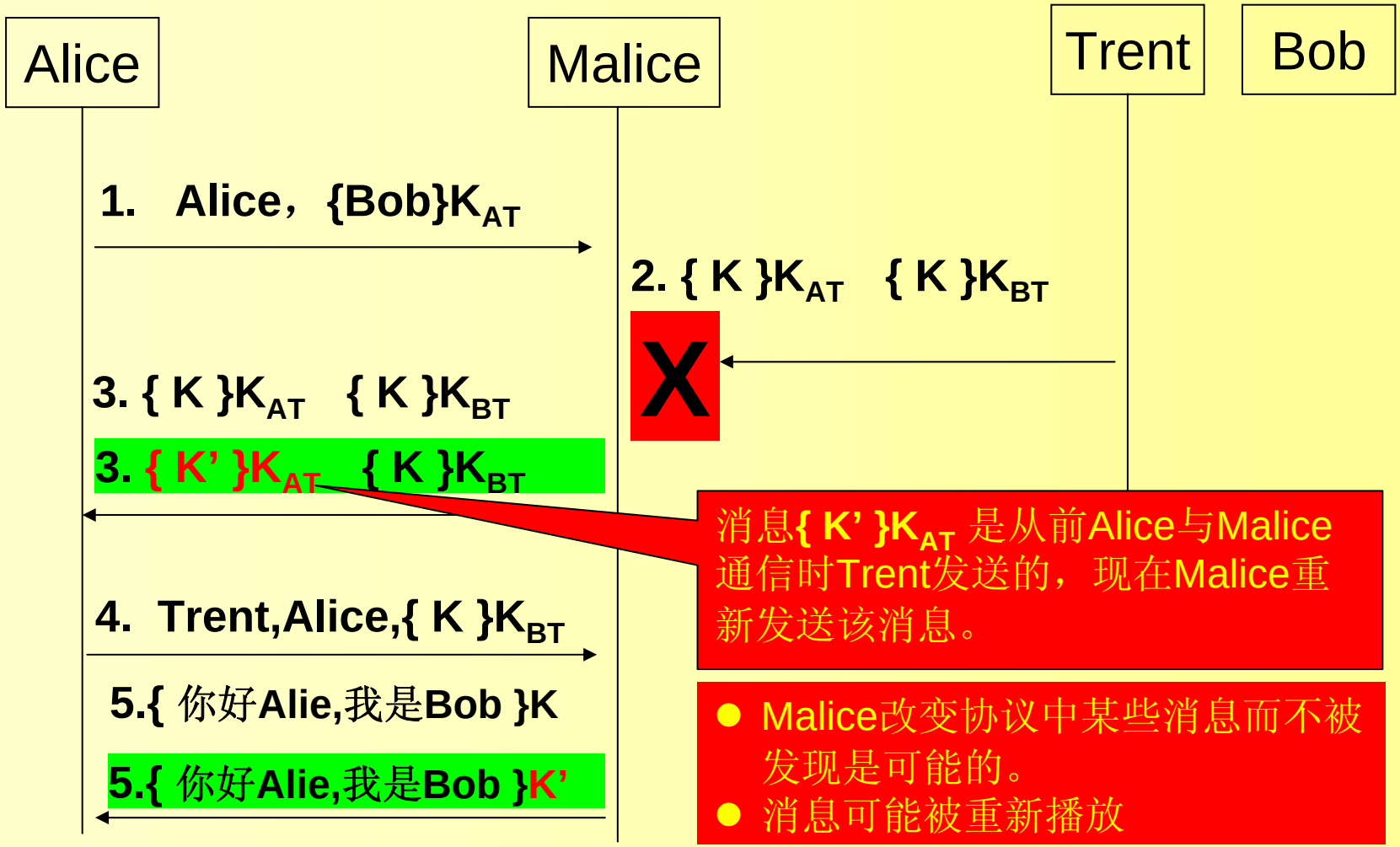


Malice遇到了困难？

攻击2 — 对协议2 的一个攻击

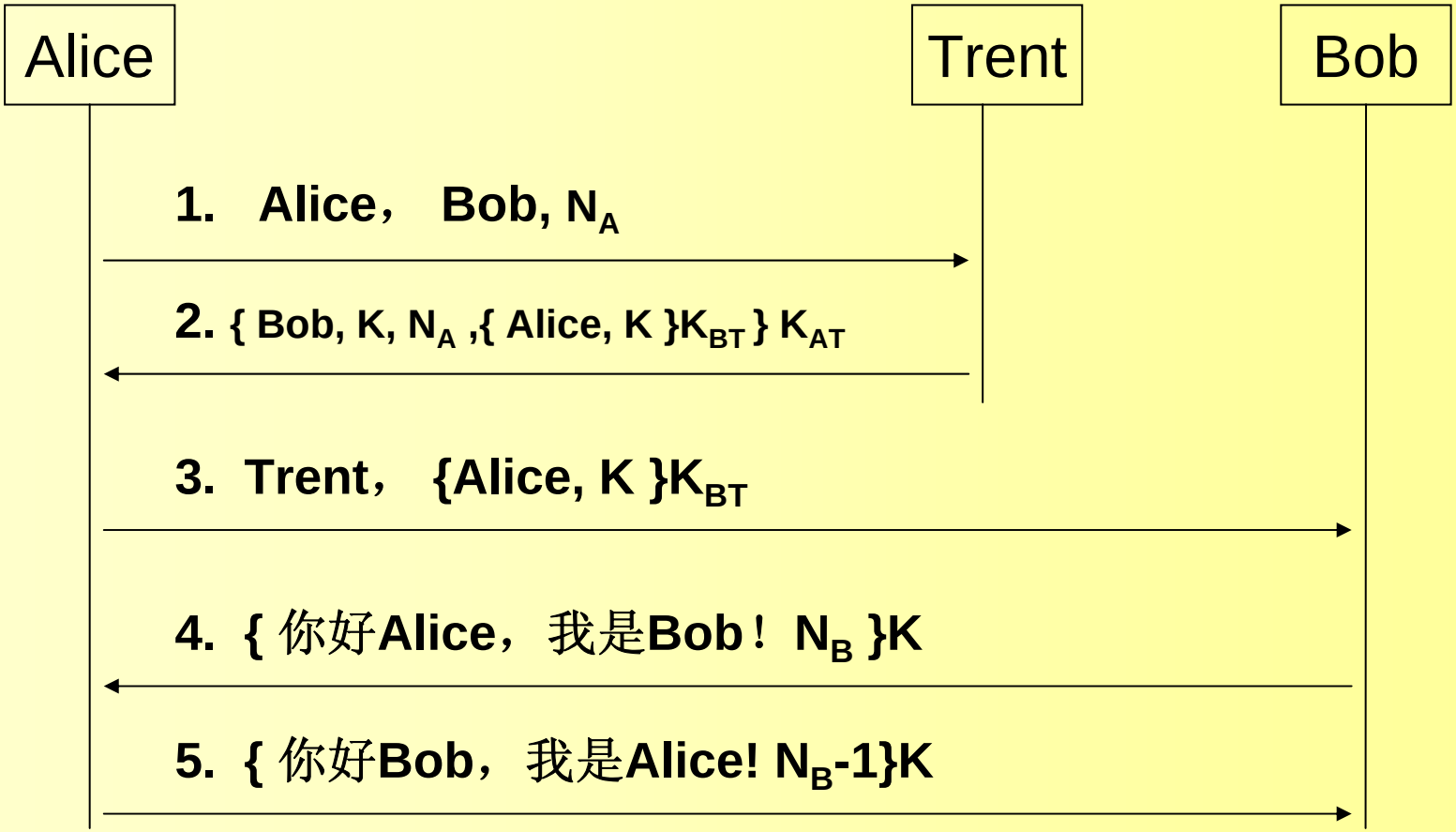


攻击3 — 对协议2的又一个攻击



协议3 — 询问应答协议

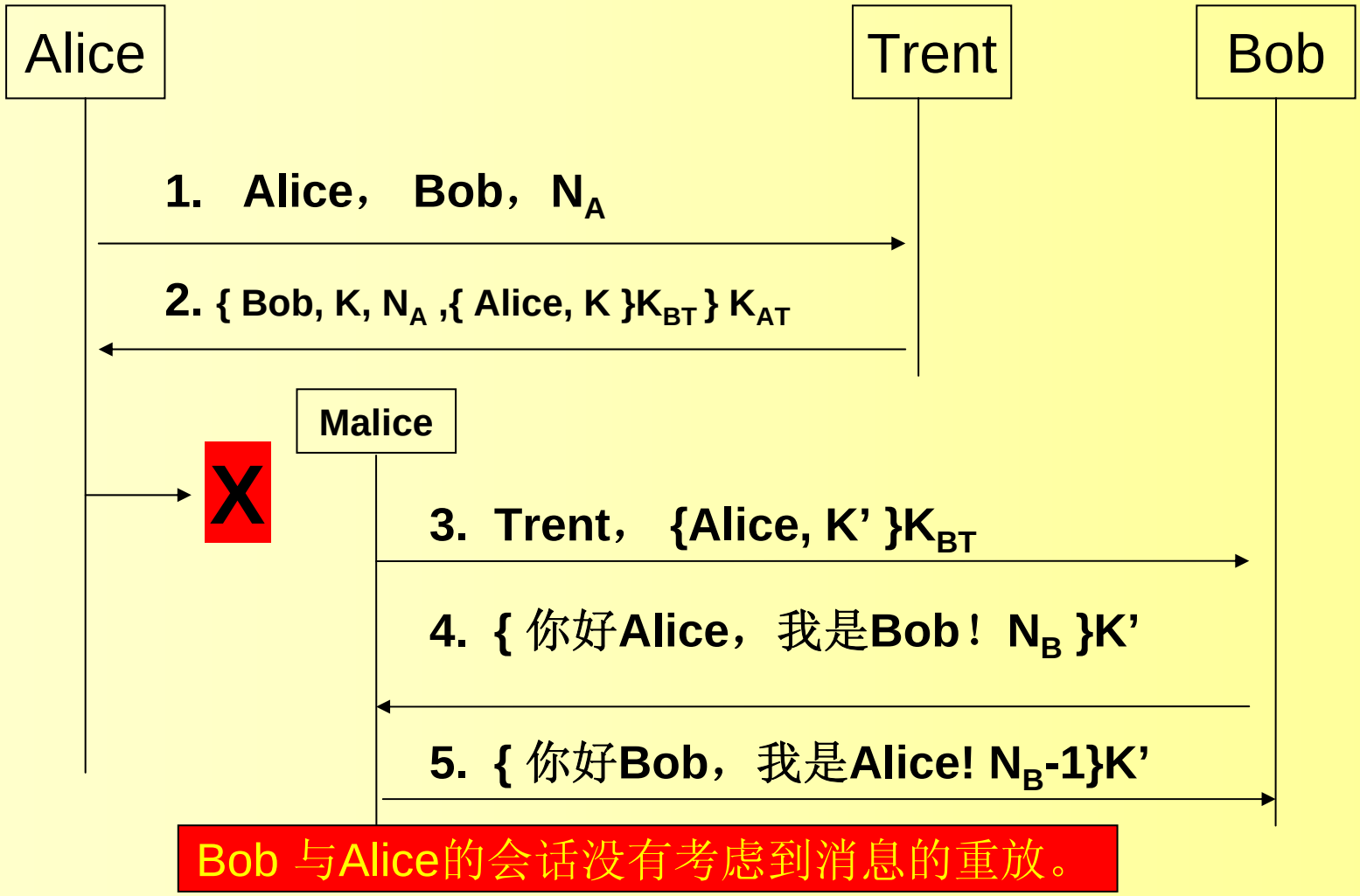
Needham-Schroeder 对称密钥认证协议



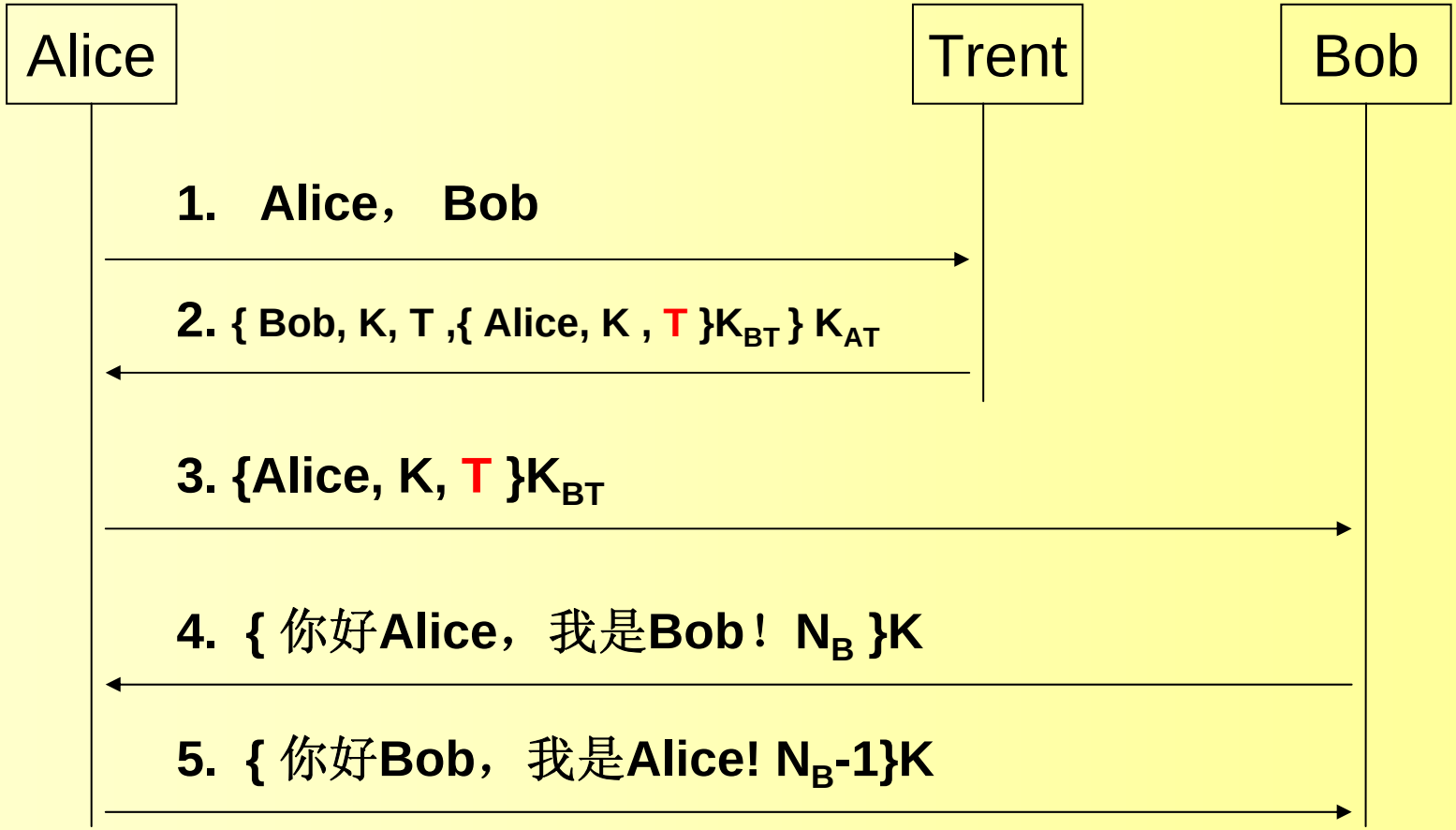
Needham-Schroeder对称密钥认证协议

- R.M. Needham and M.D. Schroeder, using encryption for authentication in large networks of computers, communications of the ACM, 21(12): 993-999, 1978.

攻击3— 对协议3的攻击


Alice
Trent
Bob
1. Alice， Bob， N_A
2. { Bob, K, N_A ,{ Alice, K }K_{BT} } K_{AT}
Malice
X
3. Trent， {Alice, K' }K_{BT}
4. { 你好Alice，我是Bob！ N_B }K'
5. { 你好Bob，我是Alice! N_B-1}K'
Bob 与Alice的会话没有考虑到消息的重放。

Denning & Sacco的建议— 时间戳



验证是要求： $|Clock - T| < \Delta t_1 + \Delta t_2$

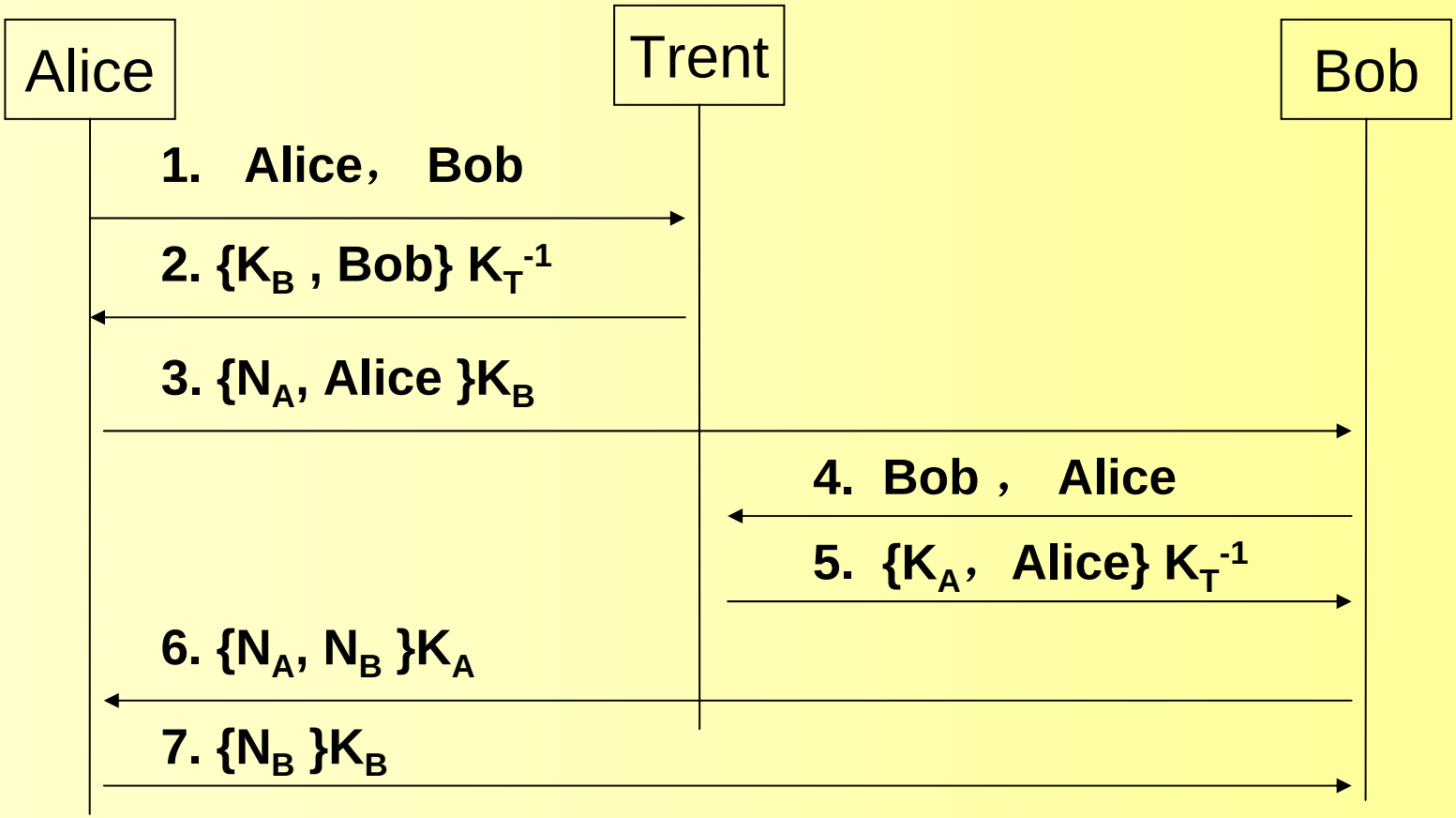
关键的问题

- 在错误的主体之间共享密钥
- 建立一个错误的会话密钥
- 消息在传递过程中被篡改了而未被发现

- 中间人攻击
  - 修改消息：消息没有鉴别
  - 消息重放：消息不是新鲜的
  - 解密验证：验证是通过解密运算进行的

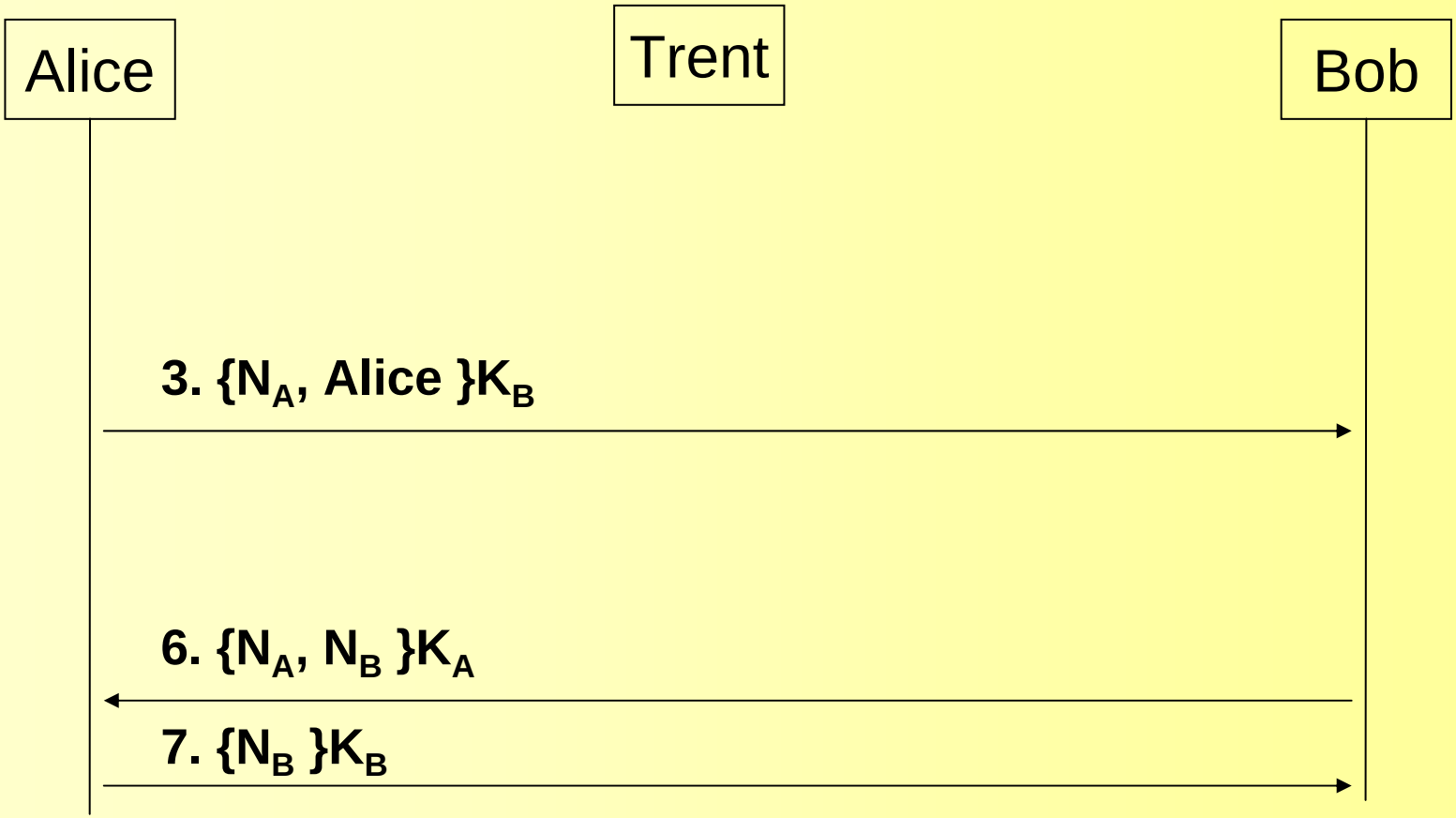
Needham-Schroeder公钥认证协议

- 假定：Alice、Bob、Trent的密钥对分别为$\langle K_A, K_A^{-1}\rangle$，$\langle K_B, K_B^{-1}\rangle$，$\langle K_T, K_T^{-1}\rangle$，$K_A^{-1}$等为私钥。



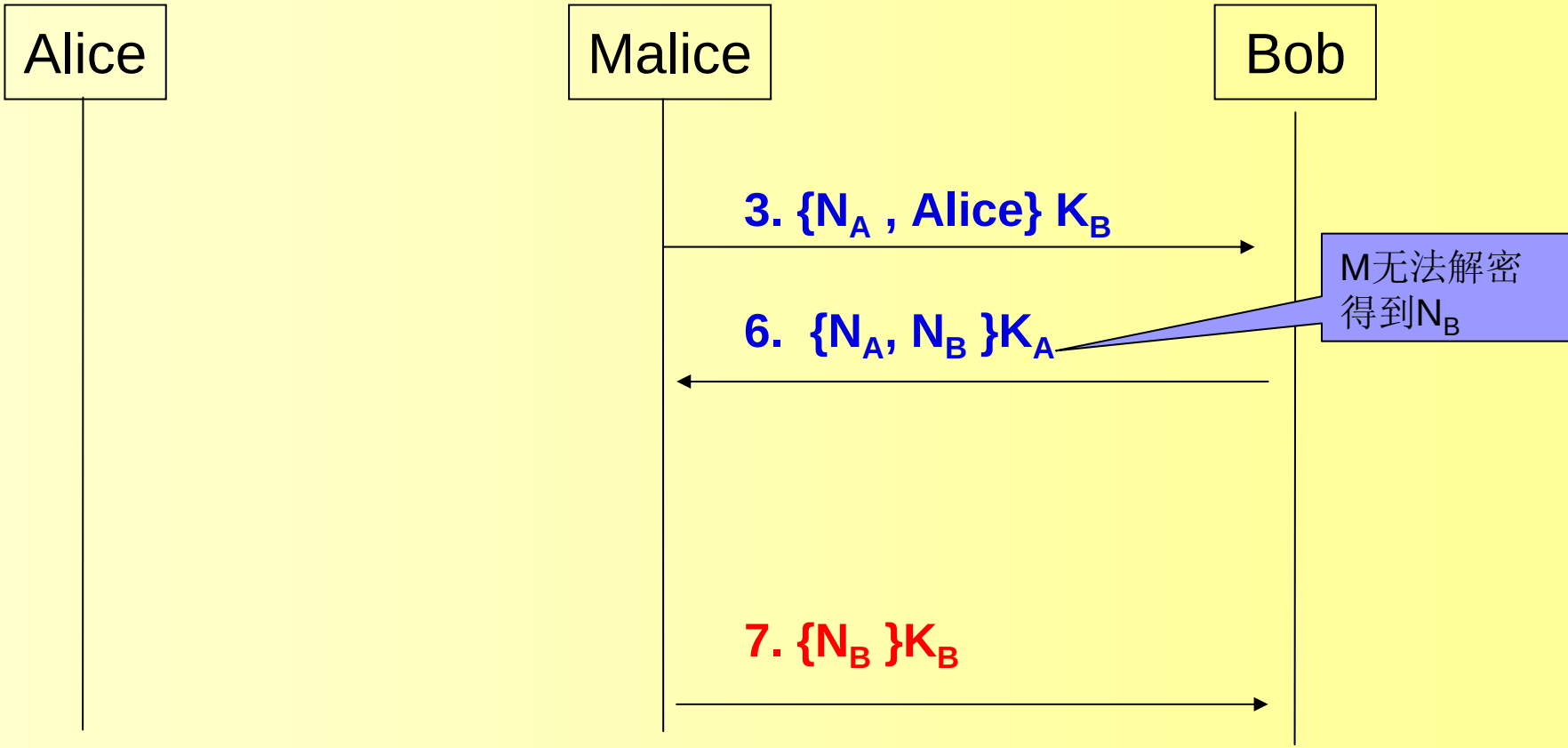
Needham-Schroeder公钥认证协议

- 假定：Alice、Bob、Trent的密钥对分别为$\langle K_A, K_A^{-1}\rangle$，$\langle K_B, K_B^{-1}\rangle$，$\langle K_T, K_T^{-1}\rangle$，$K_A^{-1}$等为私钥。



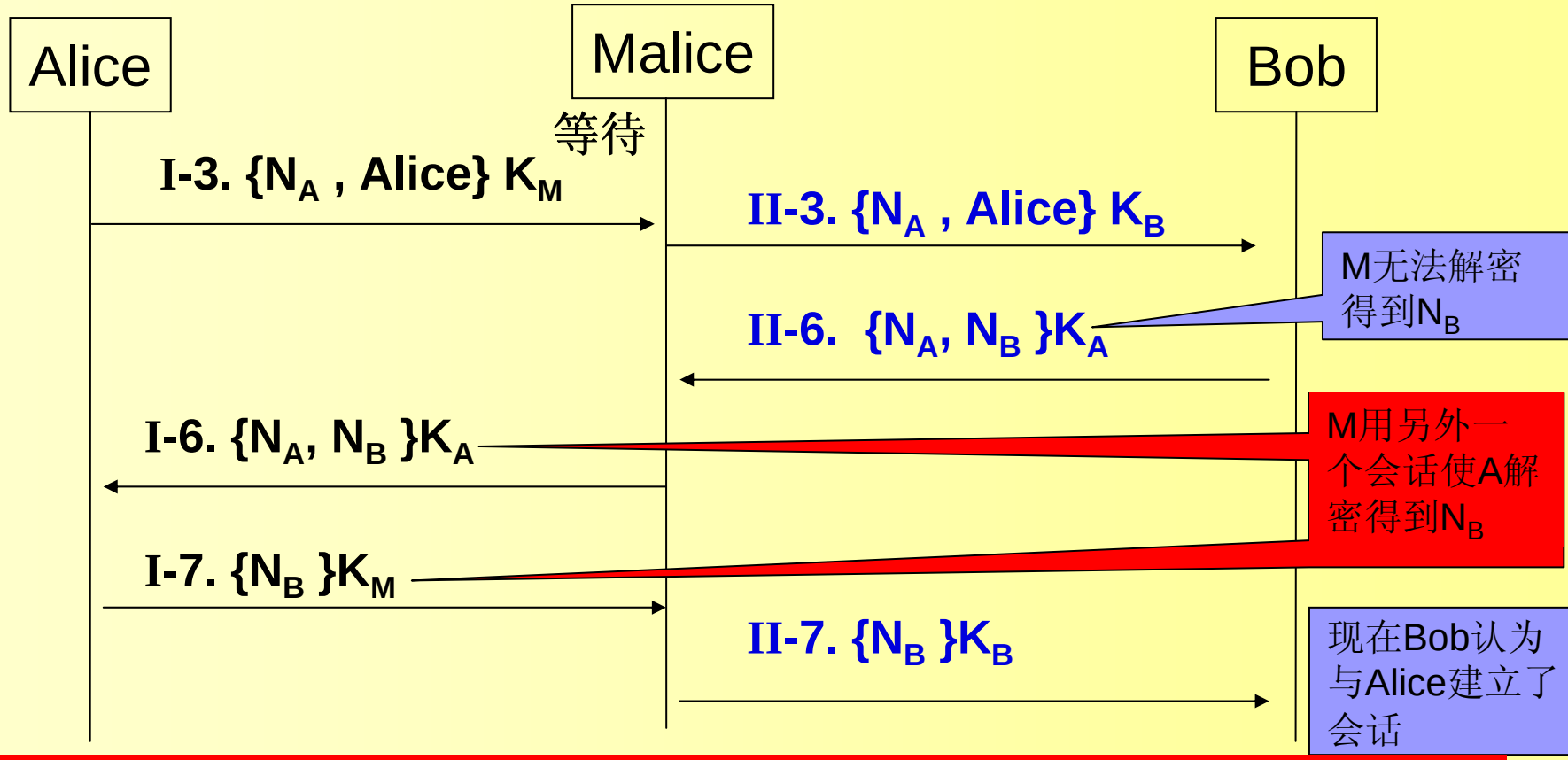
Lowe 的对Needham-Schroeder公钥认证协议攻击

- G. Lowe, A attack on Heedham-Schroeder public-key authentication protocol, Information Processing Letters, 56(3), pp131-133, 1995.



Lowe 的对Needham-Schroeder公钥认证协议攻击

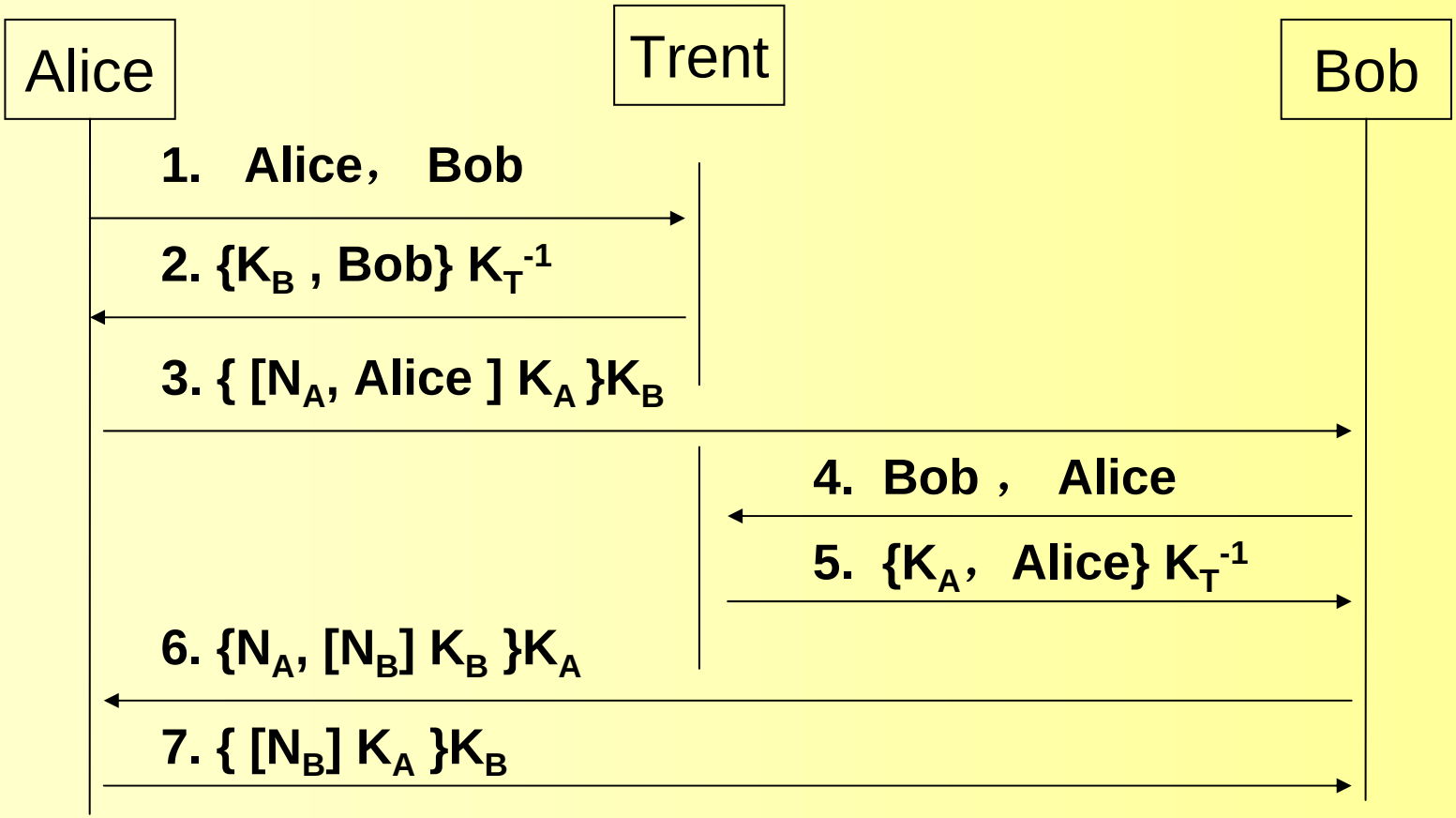
- G. Lowe, A attack on Needham-Schroeder public-key authentication protocol, Information Processing Letters, 56(3), pp131-133, 1995.

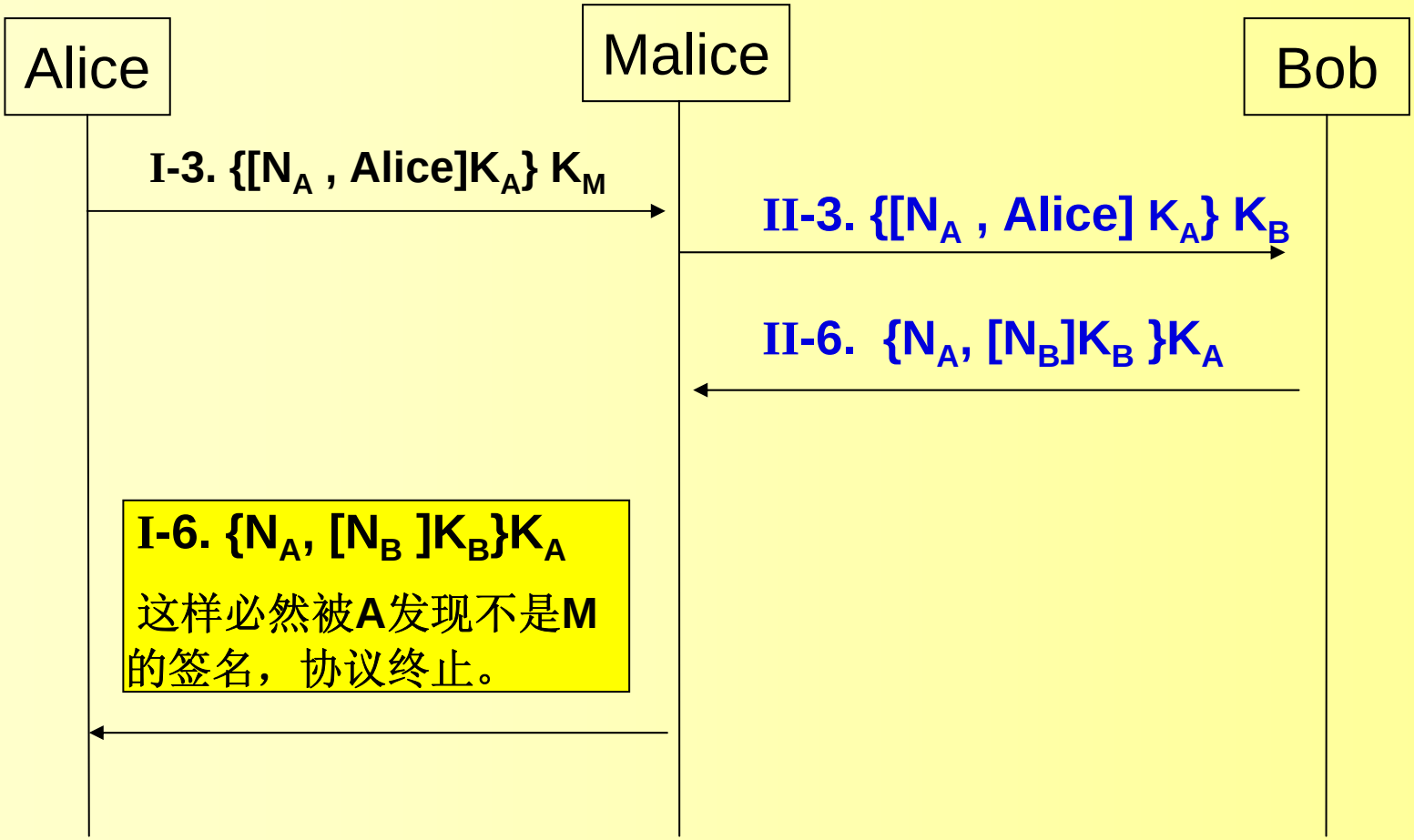


密码协议应该设计成：即使用户为攻击者提供预言服务(oracle service)它也是安全的。

Needham-Schroeder公钥认证协议的改进

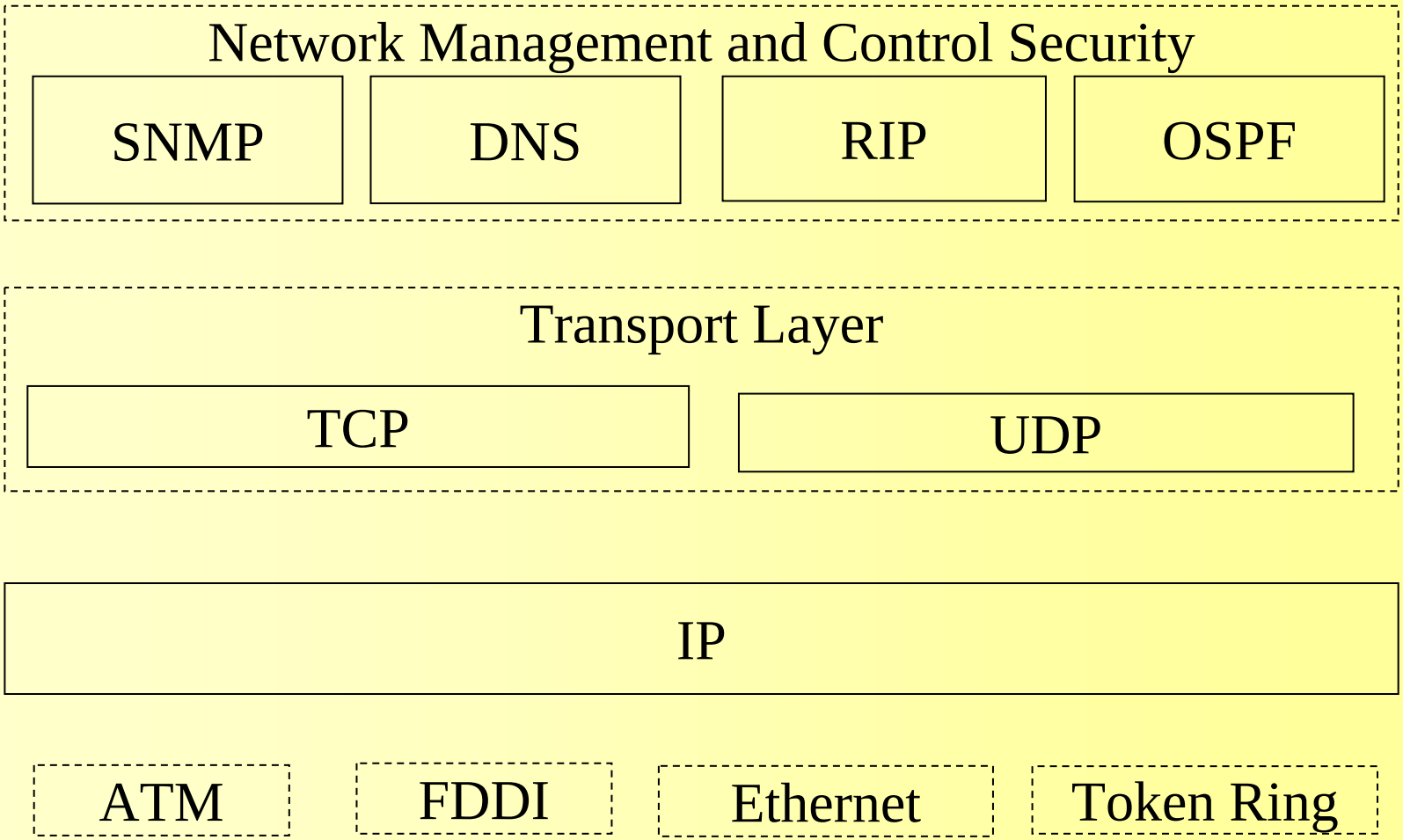
- 假定：Alice、Bob、Trent的密钥对分别为$\langle K_A, K_A^{-1}\rangle$，$\langle K_B, K_B^{-1}\rangle$，$\langle K_T, K_T^{-1}\rangle$，$K_A^{-1}$等为私钥。




Alice
Malice
Bob
I-3. {[N_A , Alice]K_A} K_M
II-3. {[N_A , Alice] K_A} K_B
II-6. {N_A, [N_B]K_B }K_A
I-6. {N_A, [N_B]K_B}K_A
这样必然被A发现不是M的签名，协议终止。

3. IP Security

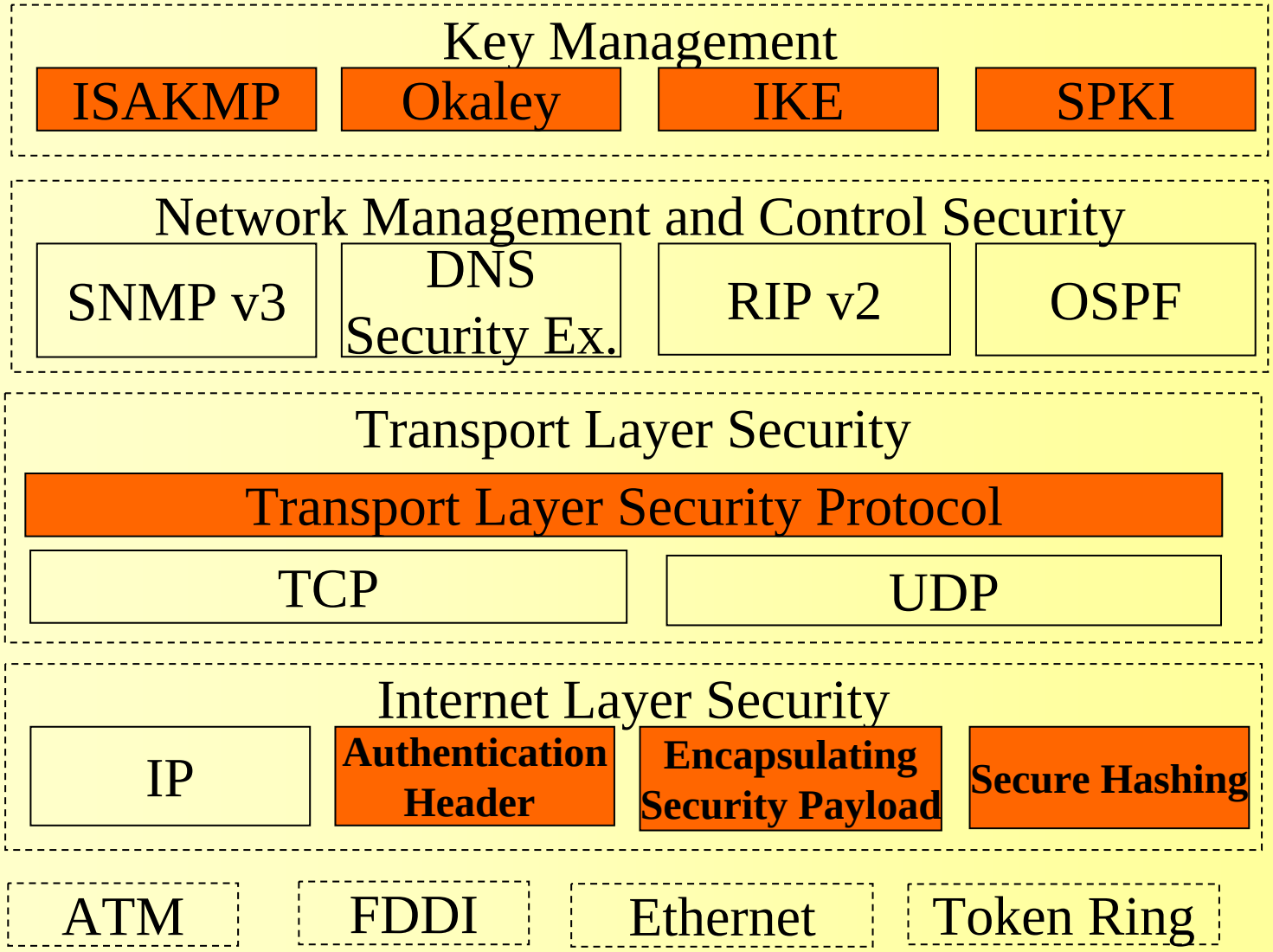
TCP/IP Protocols



IP数据包不具有安全性

- 攻击者可以:
  - 伪造IP包的地址
  - 查看IP包的内容
  - 重播以前的包
  - 修改IP包的内容
- 通信者不能:
  - 保证IP包的真实来源
  - 保证IP包没有泄密
  - IP包发送者的当前意图
  - 发送者的真实数据

Internet 安全协议栈



IPsec提供的安全服务

- 访问控制
- 无连接通信的完整性
- 数据源的验证
- 拒绝重播的数据包
- 机密性
- 有限的通信量的机密性
- 直接为上层协议服务： TCP, UDP,ICMP, BGP等。

IPsec的组成

- 安全协议
  - Authentication Header (AH)
  - Encapsulating Security Payload (ESP)
- 自动密钥协商
  - The Internet Key Exchange (IKE)
  - Internet Security Association and Key Management Protocol (ISAKMP)
  - The OAKLEY Key Determination Protocol
  - The Internet IP Security Domain of Interpretation for ISAKMP

IPsec 如何工作?

- IPsec利用两个安全协议 (AH,ESP)提供安全服务
  - AH 提供无连接运输的完整性、数据源验证、抗重播
  - ESP机密性、通信量保密、无连接运输的完整性、数据源验证、抗重播
  - AH 和ESP 单独或者联合使用

IP的实现模式

- 两种模式
  - 运输模式(transport mode)　为上层协议提供安全服务

IP head	AH head	ESP head	payload

  - 隧道模式(tunnel mode)　封装IP报文;

IP head	**AH head**	**ESP head**	**IP head**	**payload**

- IPsec的管理机制
  - 采用哪些安全服务设施
  - 选择安全信道的粒度;
  - 选择密码算法;

- AH 和 ESP 都要利用安全关联进行工作
- IKE的主要任务就是建立和管理安全关联

定义

- 安全关联SA是两个通信实体建立起来的“连接”，它们用来保护数据的一系列参数：IPsec的协议类型、加密算法、认证方式、加密和认证的密钥。密钥的生成时间和防重播的序列号等。
- SA可以手工建立。也可以利用密钥交换协议自动建立
- SA是单向的
  - 如果主机A和主机B利用ESP通信，则主机A的SA(out)和主机B(in)共享相同的”连接”，主机A的SA(in)和主机B(out)共享另外一个相同的”连接”
- 协议相关性
  - SA可以用于AH，也可以用于ESP；但是不能同时用于AH和ESP两个协议。

SA 参数

- 序列号
  - 32位数用来AH和ESP的头部序列号域，在一个安全信道的发出报文时填充。
- 序列号溢出处理标志位
  - 当序列号溢出时是否要产生一个审计事件并禁止利用这个SA进一步传输数据

SA 参数

- 防重播的窗口
- AH 相关的参数
  - 认证算法、密钥、密钥生存期
- ESP 相关参数
  - 加密和认证算法
  - 密钥、初始化向量
- SA 的生存期
- IPsec的模式
- 路径MTU

SAD

- 名义安全关联数据库 (SAD)
  - 每一个记录表示一个安全关联
- 出发处理
  - 网络连接－>安全策略数据库的记录－>安全信道的SA
  - 如果适合当前连接的安全策略库中相应的记录中没有指出相应的SA的记录，则协议发起产生相关SA的过程。
- 接收处理
  - SAD中的每一个记录位置由<目标IP 地址, IPsec 协议类型, SPI>确定。

SA 索引

- SA 可以由以下三个元素唯一地确定
  - 安全参数索引 (SPI);
  - 目标IP 地址;
  - 安全协议 (AH or ESP) ;

安全策略数据库(SPD)

The Security Policy Database

- 安全策略数据库
  - SPD 指定给 IP 数据流提供的安全服务
    - 根据源地址、目标地址、数据流的网络协议th等确定。
  - 选择子： 将IP包头和上层协议的包头域映射到SPDB中的一条策略。

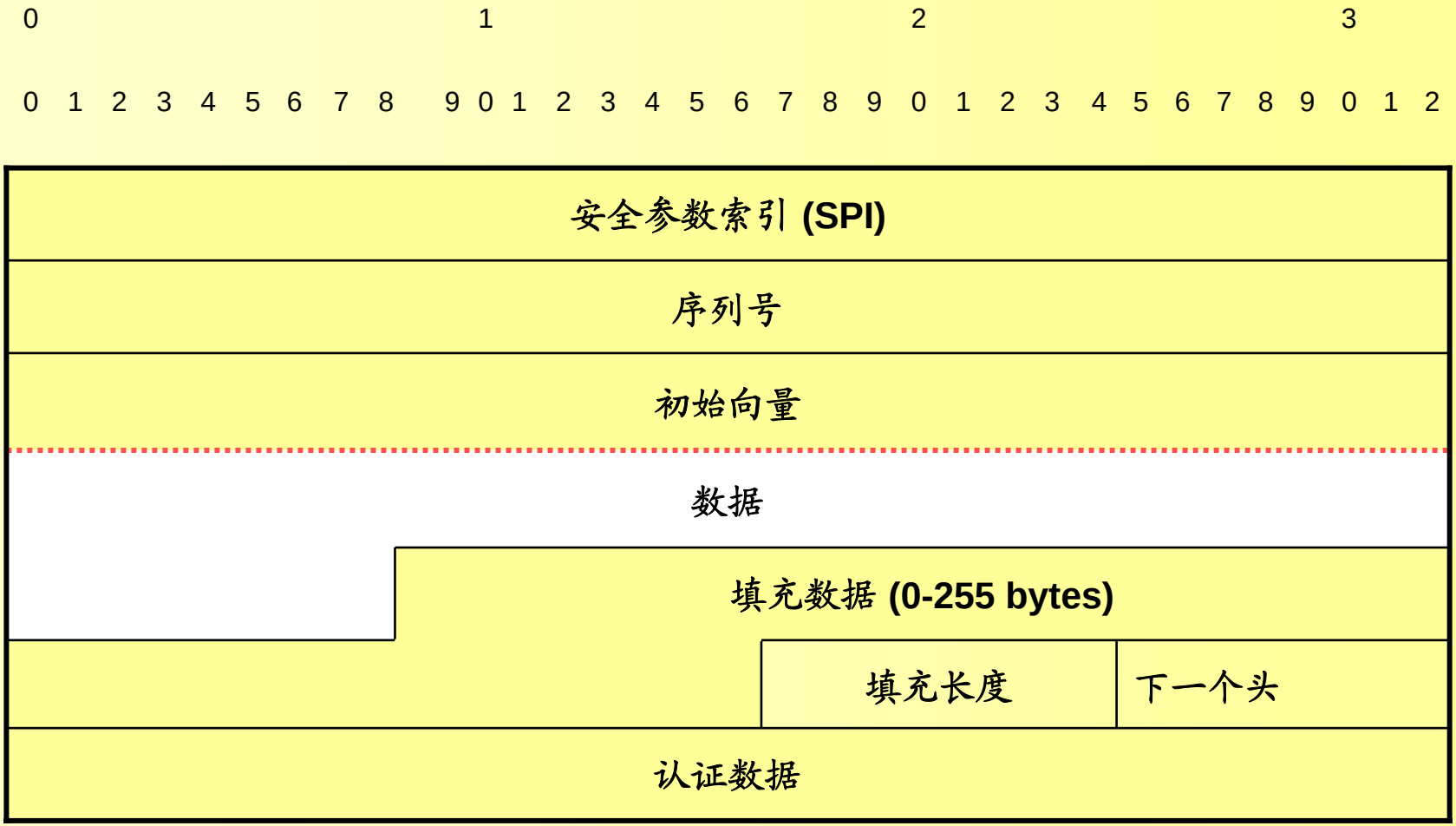
SPD

- 在发送和接收数据的时候都需要检索SPD，以确定数据处理的策略;
- 接收和发送的策略需要分别存储在SPD的不同的策略记录中;
- 在不同的接口上，需要建立不同的SPD;
- SPD对所有的数据流都必须给出策略;
- SPD对一个数据流有三种可能的策略
  - 丢弃、绕过、实施IPsec处理

选择子 (Selector)

- 一个给定的SA有一个确定的“粒度”，其“粒度”由选择子确定，选择子决定哪些数据流由哪个SA处理；
- 选择子由以下因素确定：
  - 目标IP地址
  - 源IP地址
  - 名字
    - 用户名；例如邮件地址mozart@foo.bar.com
    - 系统名称：例如域名等。
  - 安全等级运输层协议
  - 源端口和目的端口

4. 安全载荷封装ESP


0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2
安全参数索引 (SPI)
序列号
初始向量
数据
填充数据 (0-255 bytes)
填充长度
下一个头
认证数据

(1) 安全参数索引SPI

- 32位
- 1到255 由 IANA保留
- 由安全信道的接收方在建立SA时顺序;

(2) 序列号

- 32位的无符号整数，表示单调增加的计数
- 收发两端的计数器都从0开始计数.

(3) 载荷数据

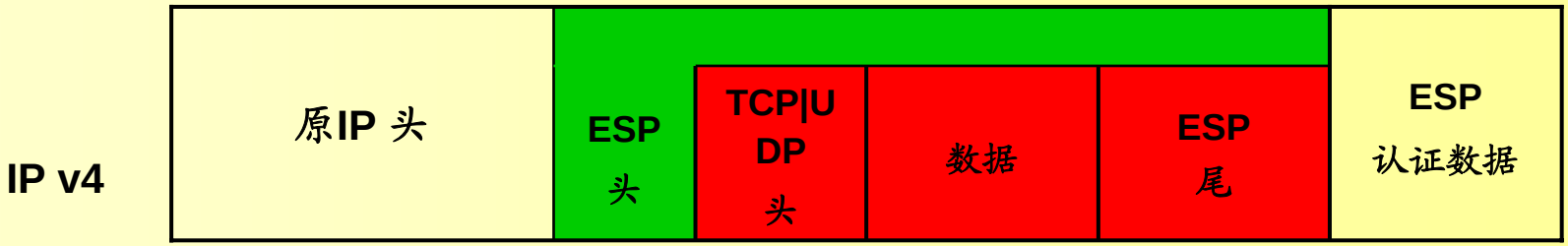
- 由Next Header 域指定的协议类型的数据，长度可变。
- 如果加密模式需要初始向量IV，则IV显式地出现在ESP的数据域中

(4) 填充

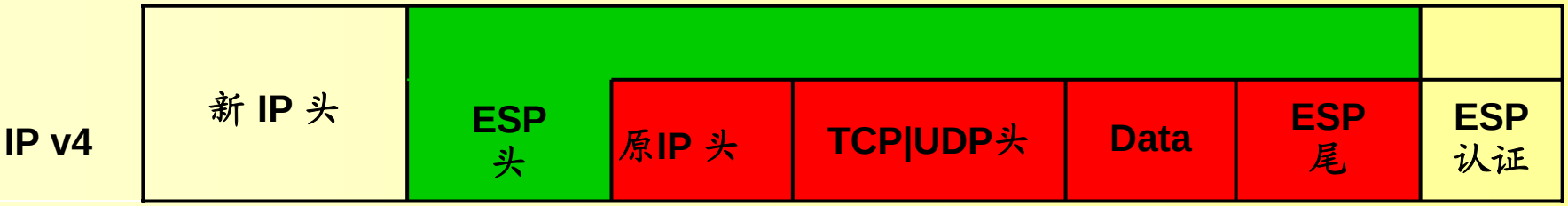
- 填充域用来填充明文域（包括载荷数据、填充长度、下一个头和填充数据），以满足加密算法的需要。
- 填充数据也可以用来隐藏数据的真实长度，支持通信量的保密;

ESP 头的位置

运输模式： ESP处理之后



隧道模式 ESP处理之后



出发包处理

- 安全关联查找
  - 对每一个出发的包，首先检查SPD，找到处理这个包的相关策略。
  - 如果需要丢弃，则记录这个丢弃事件。
  - 如果策略要求绕过IPsec 的处理，则这个报文继续“正常”协议处理。
  - 如果需要进行 IPsec 处理，则将连接映射到一个存在的SA，或激活一个SA的 协商过程。

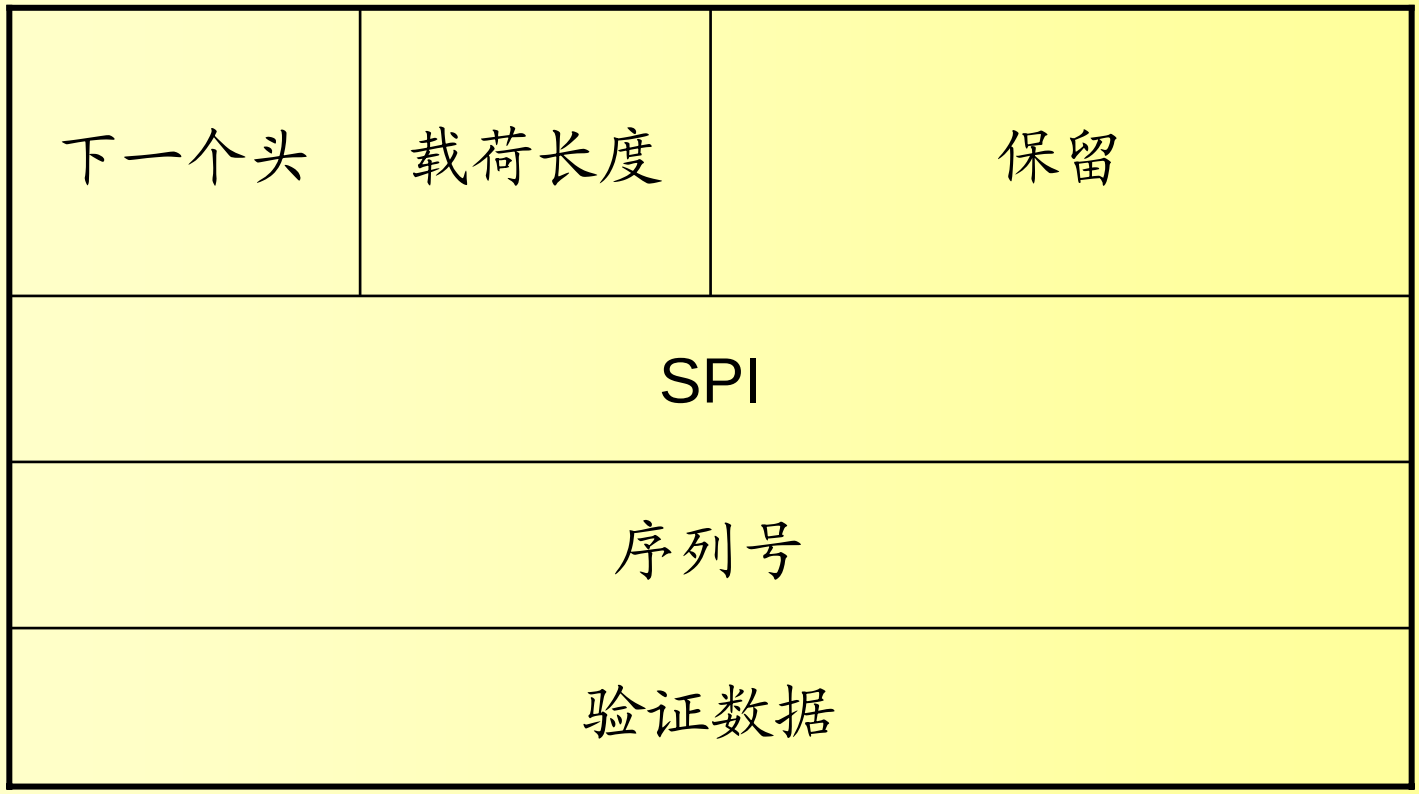
出发包处理

- 加密报文
- 产生序列号
- 完整性验证码产生
- 分片处理

接收包处理

- 组装报文
- 安全关联查找
- 序列号验证
- 完整性检查
- 报文解密

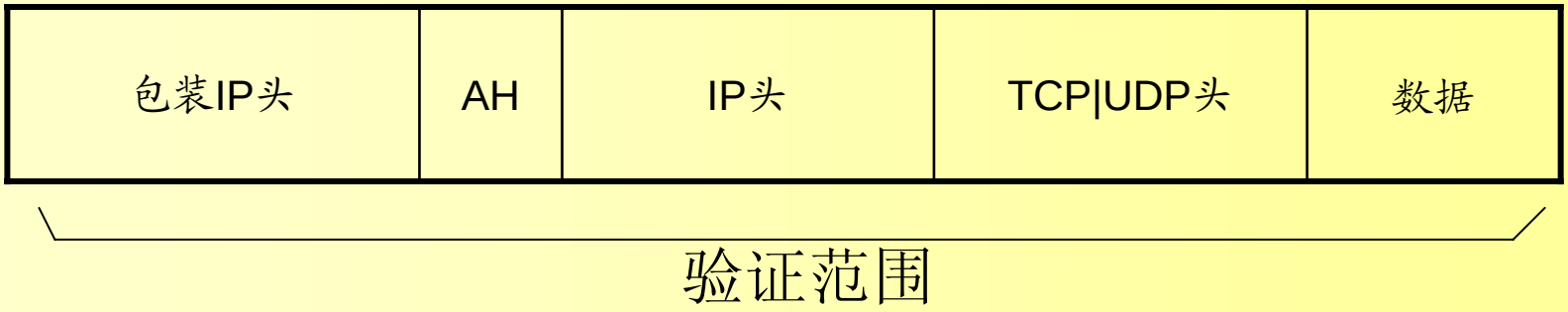
认证头 (AH)



AH 头的位置

- 运输模式 AH

- 隧道模式 AH

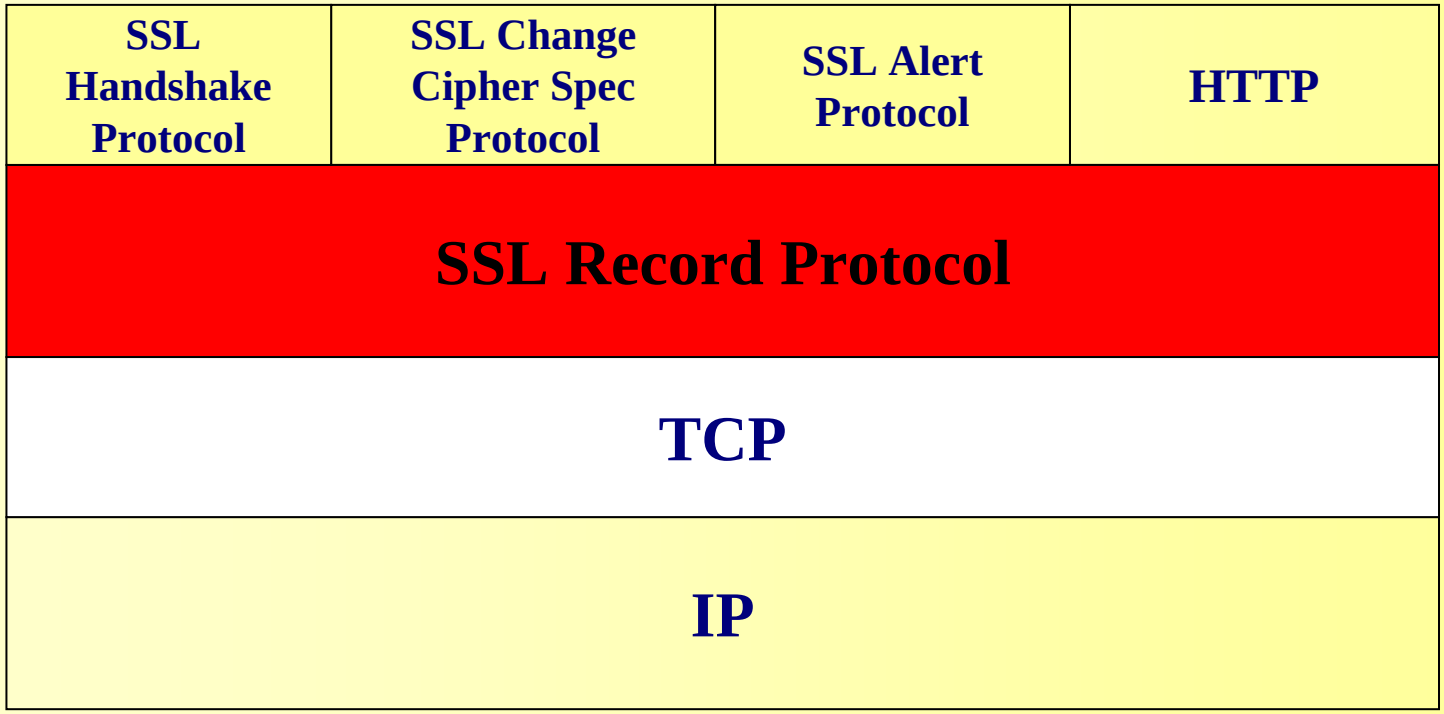


4. Security Socket Layer

参考文献

- The SSL protocol, version 3.0, Netscape Communications Corporation, March 1996.

SSL 协议栈


SSL Handshake Protocol
SSL Change Cipher Spec Protocol
SSL Alert Protocol
HTTP
SSL Record Protocol
TCP
IP

SSL 的目标

- SSL的主要目标是在两个应用程序之间提供机密、可靠服务
- SSL协议包含两个协议层
- The protocol is composed of two layers.
  - 底层是SSL记录层： **SSL Record Protocol 。这个协议层主要用于为高层协议提供保密包装服务。**
  - SSL握手协议为服务器和客户端之间提供在收发数据之前协商加密算法、密钥的服务。
- 与应用协议无关
  - SSL对高层的应用协议透明.

SSL *连接安全*

- 通信数据是保密的.
- 通信的对方可以验证
- 通信连接是可靠的.
- 提供数据完整性服务

会话状态

- 会话标识
- 对方证书
- 压缩算法
- 密码参数
  - 加密算法 、 消息鉴别码算法、摘要值长度
- 主密钥
- 可重用性

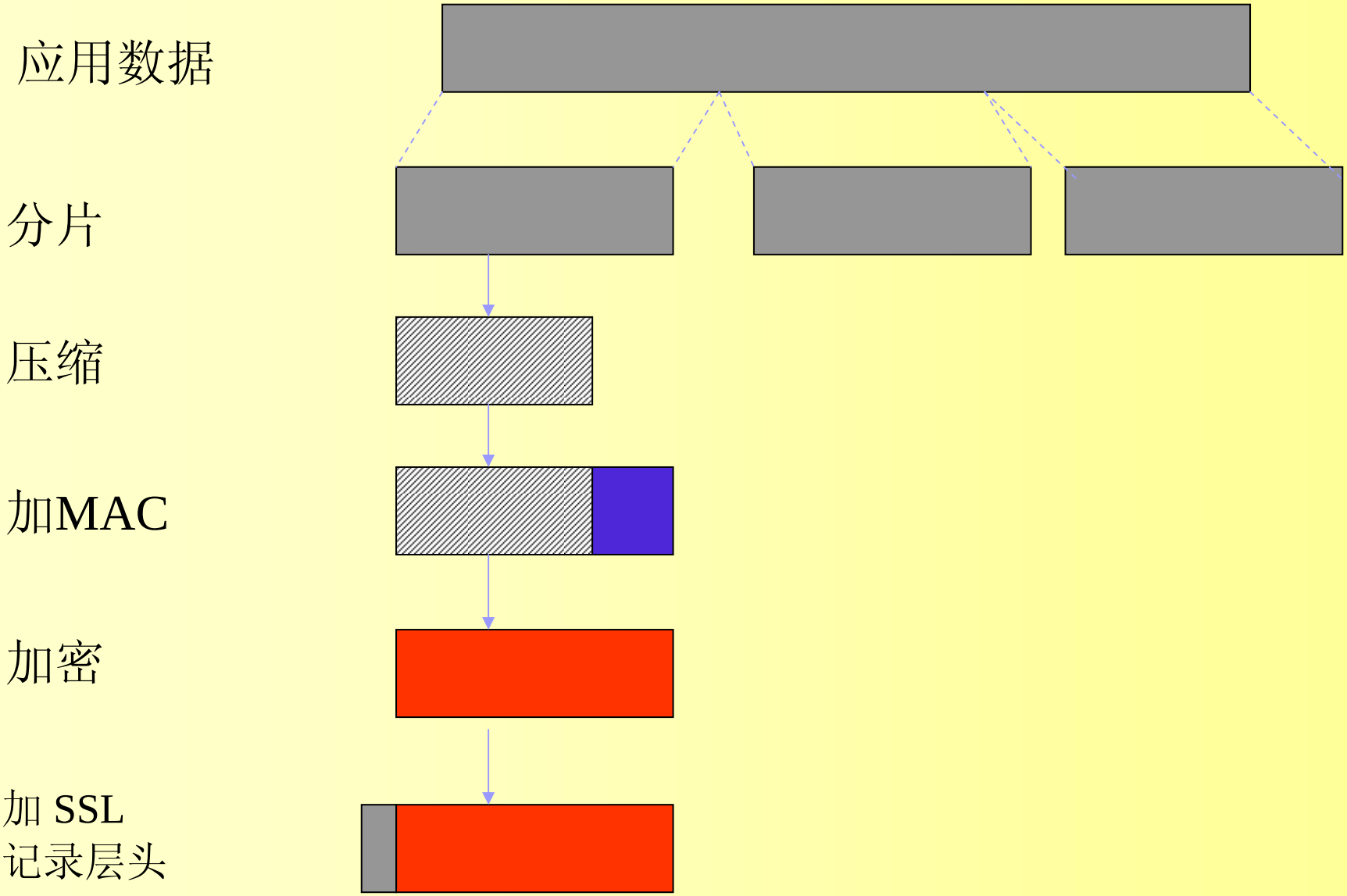
连接状态

- 服务器和客户端随机数
- 服务器写MAC密钥
- 客户端写MAC密钥
- 服务器写密钥
- 客户端写密钥
- 初始化向量
- 序列号

SSL 记录层

- SSL Record Protocol为SSL连接提供两种服务
  - 保密性。利用Handshake Protocol定义一个共享的保密密钥用于对SSL有效负载加密。
  - 消息完整性。利用Handshake Protocol定义一个共享的MAC写密钥用于形成MAC。

SSL记录层



SSL 记录层 — 支持的上层协议

- 改变密码规格消息 (ChangeCipherSpec)
- 报警协议 (Alert protocol)
- 握手协议 (Handshake protcol)
- 应用协议

CipherSpec

CipherSpec 由Hand Shake协议协商确定，Record Layer协议使用。

```
struct {
    BulkCipherAlgorithm bulk_cipher_algorithm;
            // null, rc4, rc2, des, 3des, des40, fortezza
    MACAlgorithm mac_algorithm; // null, md5, sha
    CipherType cipher_type;          //  stream, block
    IsExportable   is_exportable     // true, false
    uint8    hash_size;
    uint8    key_material;
    uint8    IV_size;
} CipherSpec;
```

改变密码规格消息 (ChangeCipherSpec)

- 使连接的挂起状态转变为当前状态。
- 改变连接将要使用的密文族。

握手协议 (Handshake protcol)

- 运行在SSL 记录层之上
- 生成一个会话的密码参数
  - 协商协议的版本号
  - 协商加密算法
  - 认证
  - 协商共享密钥

协商一个新的会话

客户端		服务器
ClientHello	→	
	←	ServerHello
		Certificate*
		CertificateRequest*
		ServerKeyExchange*
Certificate*	→	
ClientKeyExchange		
CertificateVerify*		
change cipher spec		
Finished		
	←	change cipher spec
		Finished
Application Data	→	
	←	Application Data

更新一个旧的会话

客户端		服务器
ClientHello	→	
	←	**ServerHello**
		change cipher spec
		Finished
change cipher spec	→	
Finished		
Application Data	→	
	←	**Application Data**

握手协议支持的消息类型

hello_request
client_hello
server_hello
certificate
server_key_exchange
certificate_request
server_hello_done
certificate_verify
client_key_exchange
finished

ClientHello

server_version
random
session_id
cipher_suite
compression_method

- 客户能理解的最高**SSL**版本
- **32bits**时间戳和**28**字节随机数
- **0:** 在新的会话上创建新的连接
  非**0:** 修改已存在的连接的参数或在这个会话上创建新的连接
- 客户支持的加密参数组序列
- 客户支持的压缩算法组序列

CipherSuits

```
SSL_RSA_WITH_NULL_MD5                    = { 0x00,0x01 };
SSL_RSA_WITH_NULL_SHA                    = { 0x00,0x02 };
SSL_RSA_EXPORT_WITH_RC4_40_MD5           = { 0x00,0x03 };
SSL_RSA_WITH_RC4_128_MD5                 = { 0x00,0x04 };
SSL_RSA_WITH_RC4_128_SHA                 = { 0x00,0x05 };
SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5       = { 0x00,0x06 };
SSL_RSA_WITH_IDEA_CBC_SHA                = { 0x00,0x07 };
SSL_RSA_EXPORT_WITH_DES40_CBC_SHA        = { 0x00,0x08 };
SSL_RSA_WITH_DES_CBC_SHA                 = { 0x00,0x09 };
SSL_RSA_WITH_3DES_EDE_CBC_SHA            = { 0x00,0x0A };
SSS_DH_RSA_WITH_DES_CBC_SHA              = {           }
```

认证算法_加密算法_MAC算法

CipherSuite	IsExport-table	Key Exchange	Cipher	Hash
SSL_NULL_WITH_NULL_NULL	*	NULL	NULL	NULL
SSL_RSA_WITH_NULL_MD5	*	RSA	NULL	MD5
SSL_RSA_WITH_NULL_SHA	*	RSA	NULL	SHA
SSL_RSA_EXPORT_WITH_RC4_40_MD5	*	RSA_EXPORT	RC4_40	MD5
SSL_RSA_WITH_RC4_128_MD5		RSA	RC4_128	MD5
SSL_RSA_WITH_RC4_128_SHA		RSA	RC4_128	SHA
SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5	*	RSA_EXPORT	RC2_CBC_40	MD5
SSL_RSA_WITH_IDEA_CBC_SHA		RSA	IDEA_CBC	SHA
SSL_RSA_EXPORT_WITH_DES40_CBC_SHA	*	RSA_EXPORT	DES40_CBC	SHA
SSL_RSA_WITH_DES_CBC_SHA		RSA	DES_CBC	SHA
SSL_RSA_WITH_3DES_EDE_CBC_SHA		RSA	3DES_EDE_CBC	SHA
SSL_DH_DSS_EXPORT_WITH_DES40_CBC_SHA	*	DH_DSS_EXPORT	DES40_CBC	SHA
SSL_DH_DSS_WITH_DES_CBC_SHA		DH_DSS	DES_CBC	SHA
SSL_DH_DSS_WITH_3DES_EDE_CBC_SHA		DH_DSS	3DES_EDE_CBC	SHA
SSL_DH_RSA_EXPORT_WITH_DES40_CBC_SHA	*	DH_RSA_EXPORT	DES40_CBC	SHA
SSL_DH_RSA_WITH_DES_CBC_SHA		DH_RSA	DES_CBC	SHA
SSL_DH_RSA_WITH_3DES_EDE_CBC_SHA		DH_RSA	3DES_EDE_CBC	SHA
SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA	*	DHE_DSS_EXPORT	DES40_CBC	SHA
SSL_DHE_DSS_WITH_DES_CBC_SHA		DHE_DSS	DES_CBC	SHA
SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA		DHE_DSS	3DES_EDE_CBC	SHA
SSL_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA	*	DHE_RSA_EXPORT	DES40_CBC	SHA
SSL_DHE_RSA_WITH_DES_CBC_SHA		DHE_RSA	DES_CBC	SHA
SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA		DHE_RSA	3DES_EDE_CBC	SHA
SSL_DH_anon_EXPORT_WITH_RC4_40_MD5	*	DH_anon_EXPORT	RC4_40	MD5
SSL_DH_anon_WITH_RC4_128_MD5		DH_anon	RC4_128	MD5
SSL_DH_anon_EXPORT_WITH_DES40_CBC_SHA		DH_anon	DES40_CBC	SHA
SSL_DH_anon_WITH_DES_CBC_SHA		DH_anon	DES_CBC	SHA
SSL_DH_anon_WITH_3DES_EDE_CBC_SHA		DH_anon	3DES_EDE_CBC	SHA
SSL_FORTEZZA_DMS_WITH_NULL_SHA		FORTEZZA_DMS	NULL	SHA
SSL_FORTEZZA_DMS_WITH_FORTEZZA_CBC_SHA		FORTEZZA_DMS	FORTEZZA_CBC	SHA

ServerHello

server_version
random
session_id
cipher_suite
compression_method

- 版本字段包含了客户建议的最低版本和服务器支持的最高版本。
- 随机数字段是服务器生成的，独立于客户的随机数字段。
- 如果客户的会话ID非零， 服务器察看会话cache ，如果能够匹配并且服务器愿意利用已有的会话状态建立一个新的联接，则服务器返回同样的值；否则，服务器的会话ID字段包含了一个新会话的值。
- 密文族字段包含了服务器从客户建议密文族列表中选择的一个密文族。压缩字段包含了服务器从客户建议压缩列表中选择的一个压缩方法。

Certificate

- If the server is to be authenticated (which is generally the case), the server sends its certificate immediately following the **server hello** message.

- The certificate is a sequence (chain) of X.509.v3 certificates, ordered with the sender's certificate first and the root certificate authority last.

三种认证模式

- 双向认证
- 服务器认证
- 匿名

匿名

- 匿名RSA
  - 客户端从Server Key Exchange消息获得服务器的公开密钥。
  - 客户端利用服务器公开密钥加密一个pre_master_secret。
  - 在 client key exchange 消息中传输。
  - 匿名RSA可能受到中间人攻击，中间人可能冒充服务器发送公开密钥。
  - pre_master_secret是保密的。

客户端		服务器
ClientHello	→	
	←	**ServerHello** **ServerKeyExchange***
ClientKeyExchange **change cipher spec** **Finished**	→	
	←	**change cipher spec** **Finished**
Application Data	→	
	←	**Application Data**

匿名RSA

```
struct {
  select (KeyExchangeAlgorithm) {
  case diffie_hellman:
     ServerDHParams params;
     Signature signed_params;
  case rsa:
     ServerRSAParams params;
     Signature signed_params;
  case fortezza_dms:
     ServerFortezzaParams params;
  };
} ServerKeyExchange;
```

```
digitally-signed struct {
  select(SignatureAlgorithm) {
  case anonymous: struct { };
  case rsa:
     opaque md5_hash[16];
     opaque sha_hash[20];
  case dsa:
     opaque sha_hash[20];
  };
} Signature;

struct {
     opaque rsa_modulus<1..2^16-1>;
     opaque rsa_exponent<1..2^16-1>;
} ServerRSAParams;
```

匿名 Diffie-Hellman

- 服务器的公开数在**server key exchange** 消息中传输
- 客户端的公开数在**client key exchange** 消息中传输

客户端		服务器
ClientHello	→	
	←	**ServerHello** **ServerKeyExchange***
ClientKeyExchange **change cipher spec** **Finished**	→	
	←	**change cipher spec** **Finished**
Application Data	→	
	←	**Application Data**

匿名 Diffie-Hellman

```
struct {
  select (KeyExchangeAlgorithm) {
  case diffie_hellman:
     ServerDHParams params;
     Signature signed_params;
  case rsa:
     ServerRSAParams params;
     Signature signed_params;
  case fortezza_dms:
     ServerFortezzaParams params;
  };
} ServerKeyExchange;
```

```
digitally-signed struct {
  select(SignatureAlgorithm) {
  case anonymous: struct { };
  case rsa:
     opaque md5_hash[16];
     opaque sha_hash[20];
  case dsa:
     opaque sha_hash[20];
  };
} Signature;

struct {
     opaque dh_p<1..2^16-1>;
     opaque dh_g<1..2^16-1>;
     opaque dh_Ys<1..2^16-1>;
} ServerDHParams;
```

RSA 密钥交换和认证

- 公开密钥可能包含在
  - 服务器的证书中
  - **server key exchange** 消息中的临时公钥
- 如果用RSA临时公开密钥来交换会话密钥，则用服务器Certificate中的公钥相应的私钥进行签名。
- 客户端用这个RSA临时公开密钥来加密pre_master_secret.
- 服务器解密得到pre_master_secret，生成finished 消息

RSA 密钥交换和认证

```
struct {
  select (KeyExchangeAlgorithm) {
  case diffie_hellman:
     ServerDHParams params;
     Signature signed_params;
  case rsa:
     ServerRSAParams params;
     Signature signed_params;
  case fortezza_dms:
     ServerFortezzaParams params;
  };
} ServerKeyExchange;
```

```
digitally-signed struct {
  select(SignatureAlgorithm) {
  case anonymous: struct { };
  case rsa:
     opaque md5_hash[16];
     opaque sha_hash[20];
  case dsa:
     opaque sha_hash[20];
  };
} Signature;
struct {
     opaque rsa_modulus<1..216-
     1>;
     opaque rsa_exponent<1..216-
     1>;
} ServerRSAParams;
```

RSA 密钥交换和认证

客户端		服务器
ClientHello	→	
	←	ServerHello Certificate CertificateRequest* ServerKeyExchange*
Certificate* ClientKeyExchange change cipher spec Finished	→	
	←	change cipher spec Finished
Application Data	→	
	←	Application Data

Diffie-Hellman 密钥交换与认证

- D-H公开数的传递
  - 在证书中提供D-H公开数
  - 在**client key exchange** 消息和**server key exchange message** 中提供D-H公开消息，用证书签名；
- 公开数用hello.random连接后计算散列，防止重放攻击。

master_secret

master_secret =

MD5(pre_master_secret +

SHA(`A' + pre_master_secret +

ClientHello.random+ ServerHello.random))

+MD5(pre_master_secret +

SHA(`BB' + pre_master_secret + ClientHello.random +

ServerHello.random))

+ MD5(pre_master_secret +

SHA(`CCC' + pre_master_secret +ClientHello.random +

ServerHello.random));

key_block

key_block =

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