第4章 实体认证

第 4 章 实体认证

- 4.1 实体认证概述
- 4.2 非密码认证机制
- 4.3 基于密码的认证机制
- 4.4 设计认证协议注意的问题(略)
- 4.5 认证协议

4.1 概述

- 实体认证(Entity Authentication)是设计用来让一方验证另一方的一种技术。最重要的安全服务，所有其他安全服务都依赖于该服务，实体(人、程序、客户、服务器等)
- 解决：如何证明某个人就是他所声称的那个人
- 认证可以用来对抗假冒攻击和确保身份
- 认证的方法
  - 声称者证明他知道某事，例如口令
  - 声称者在某一特定场所（也可能在某一特定的时间）提供证据
  - 验证者认可某一已经通过认证的可信方

仅使用上述原理中的任何一种进行认证是不够的，需综合应用这些原理来建立认证系统

相关概念

- 实体认证：实体身份由参加通信连接或会话的远程参与者提交，实体认证包括单向认证和双向认证
  - 单向认证：通信双方中只有一方向另一方进行认证
  - 双向认证：通信双方互相进行认证
- 申请者：需要验证身份的实体
- 验证者：试图证明申请者身份的实体
- 数据源认证与实体认证的区别
- 实体认证与密钥管理的关系

4.2 非密码认证机制

- 实体认证机制包括两大类，一类依赖密码技术，另一类与密码技术无关
- 非密码实体认证机制
  - 口令认证
  - 挑战-应答机制
  - 基于地址的机制
  - 零知识
  - 基于个人特征的机制（生物认证）

4.2.1 口令机制

- 口令或个人识别号(PIN)机制是最实用的一种认证机制，包括固定口令和一次性口令
- 固定口令[Fixed Password]：每次访问中反复使用的口令
- 基本方法
  - 验证者保存用户身份和口令的信息表
  - 申请者以明文方式将自己身份与口令发送给验证者
  - 验证者根据用户身份找出表中口令与用户提供的口令比较

口令机制

- 脆弱性
  - **外部泄露**：指未授权的人借助外边的普通网络或系统操作获取口令。应对措施
  - **口令猜测**：攻击口令机制最常用的方法，应对措施
  - **线路窃听**：窃听者监听合法用户端登录过程，如果过程中包含了一个未被保护的口令或可推导出口令的数据
  - **危及验证者**：通过内部攻击危及验证者的口令文件或数据库
  - **重放攻击**：攻击者利用其观察（监听）到的合法用户认证过程中的交换信息假冒合法用户

外部泄露的应对措施

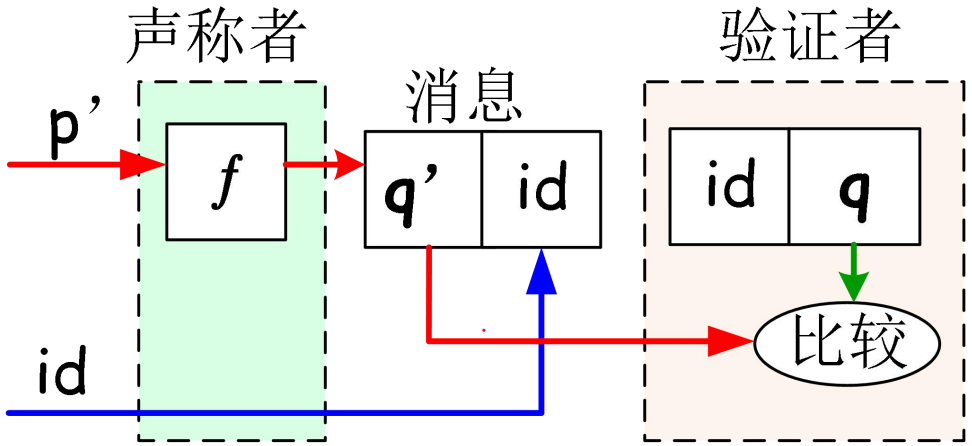
- 对用户和系统管理者进行教育，增强安全意识
- 建立严格的管理办法
- 口令必须被定期修改
- 保证每个口令只与一人有关
- 打印的口令不出现在终端上
- 使用易记的口令

口令猜测的应对措施

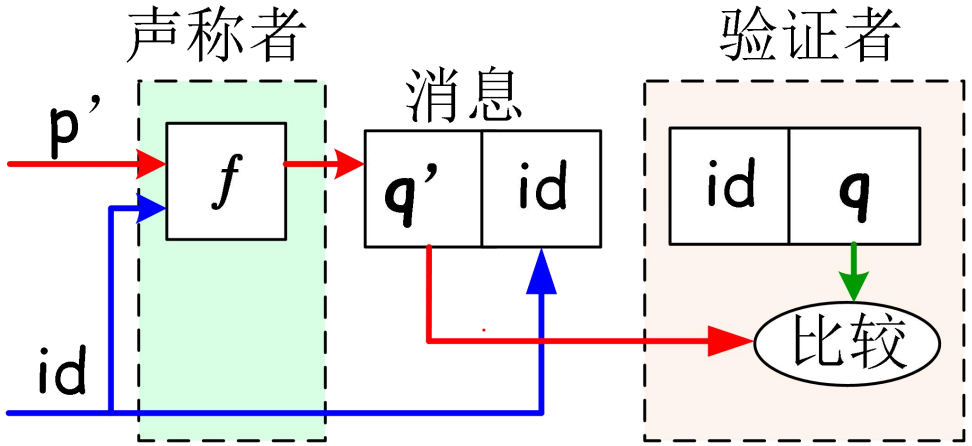
- **口令的脆弱性**：特定字母串、短口令、预设口令、使用与用户个人信息相关的口令。
- **应对措施**
  - 安全意识教育
  - 限制从一个终端或接入通道非法登录的次数
  - 把实时延迟插入到口令验证过程中
  - 确保口令被定期修改
  - 取消预设口令
  - 使用机器产生而不是用户选择的口令
- 注意：平衡**避免外部泄露所采取的措施**与**避免口令猜测采取的措施**之间的**冲突**。

线路窃听的应对措施

- 方法：**单向函数f**
- 基本的保护口令方案

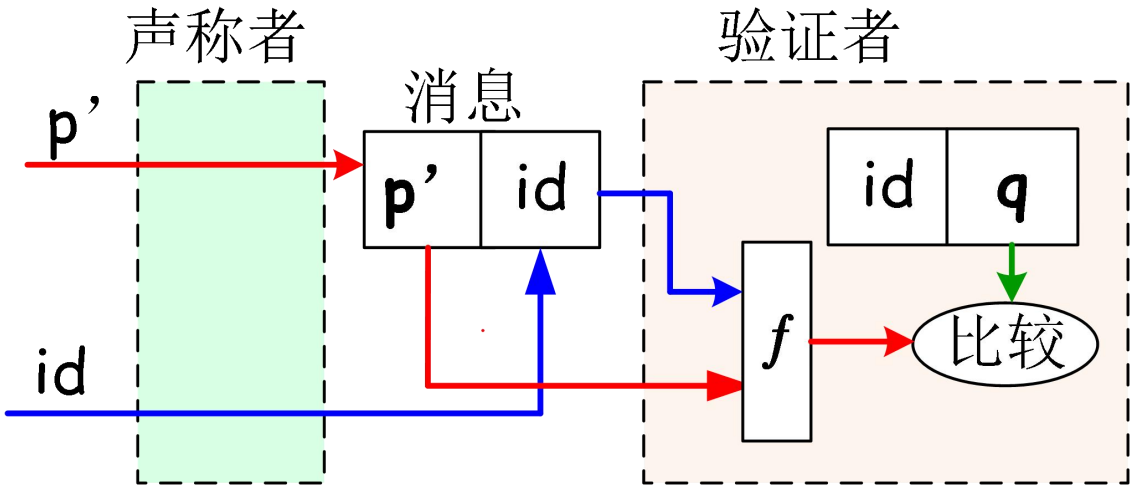


- 改进的保护口令方案

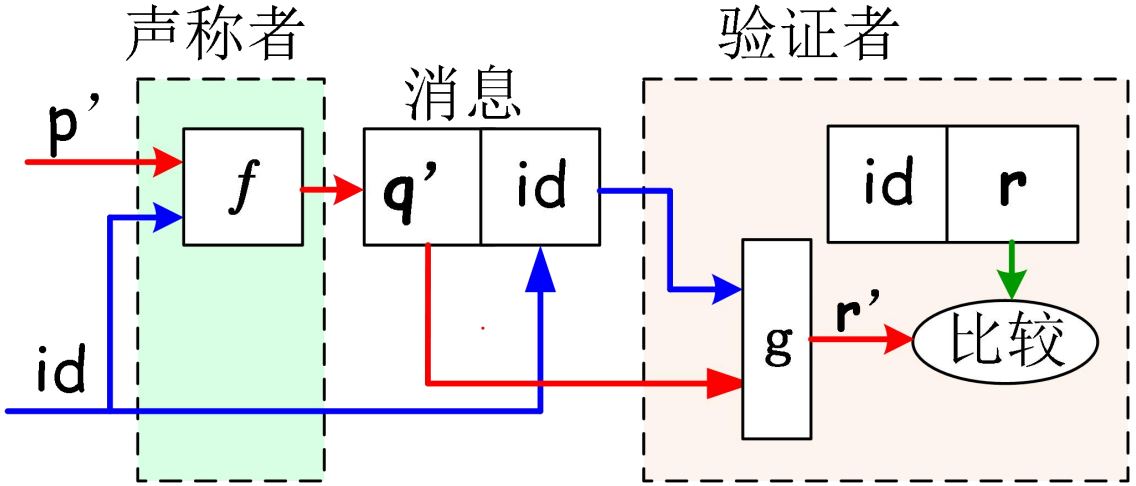


危及验证者的应对措施

- 方法：单向函数
- 基本方案

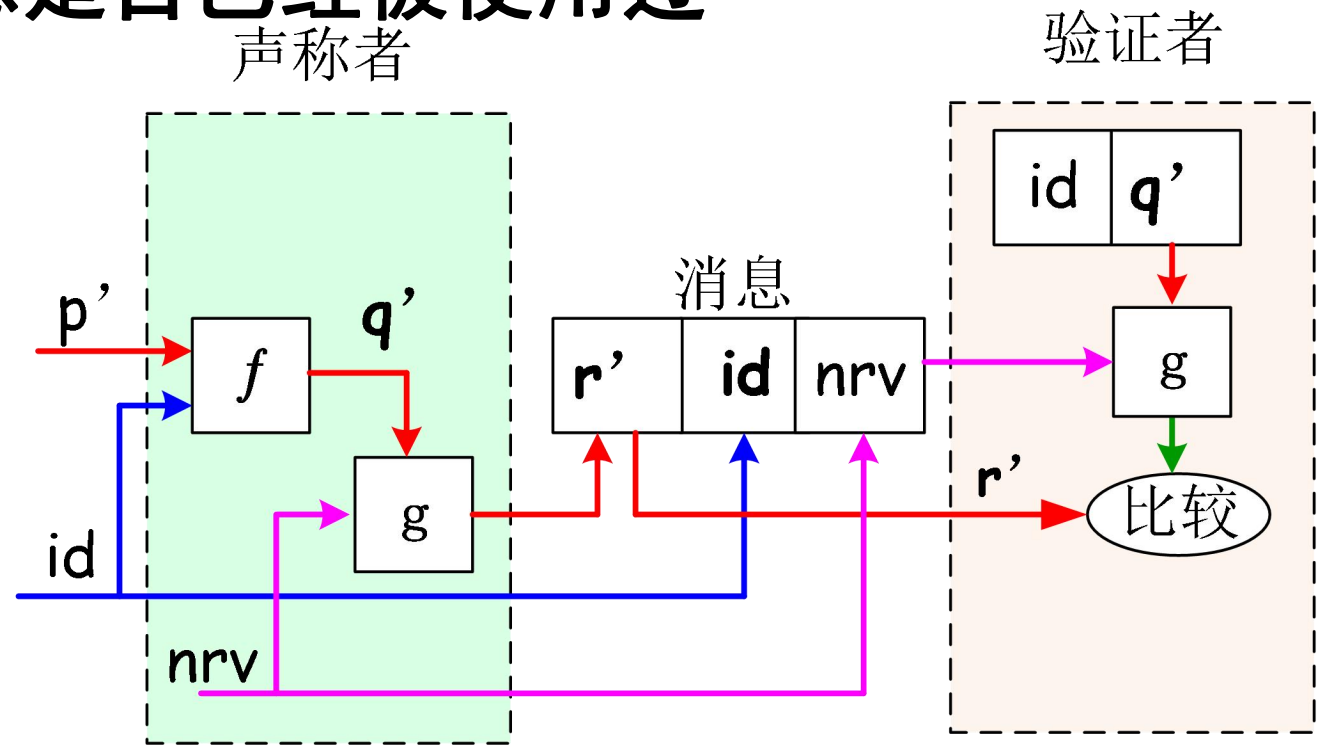


- 改进的方案



抗重放攻击的保护机制

- 思路：利用非重复值(nrv，也称为时参变量)检验该消息是否已经被使用过
- 方案



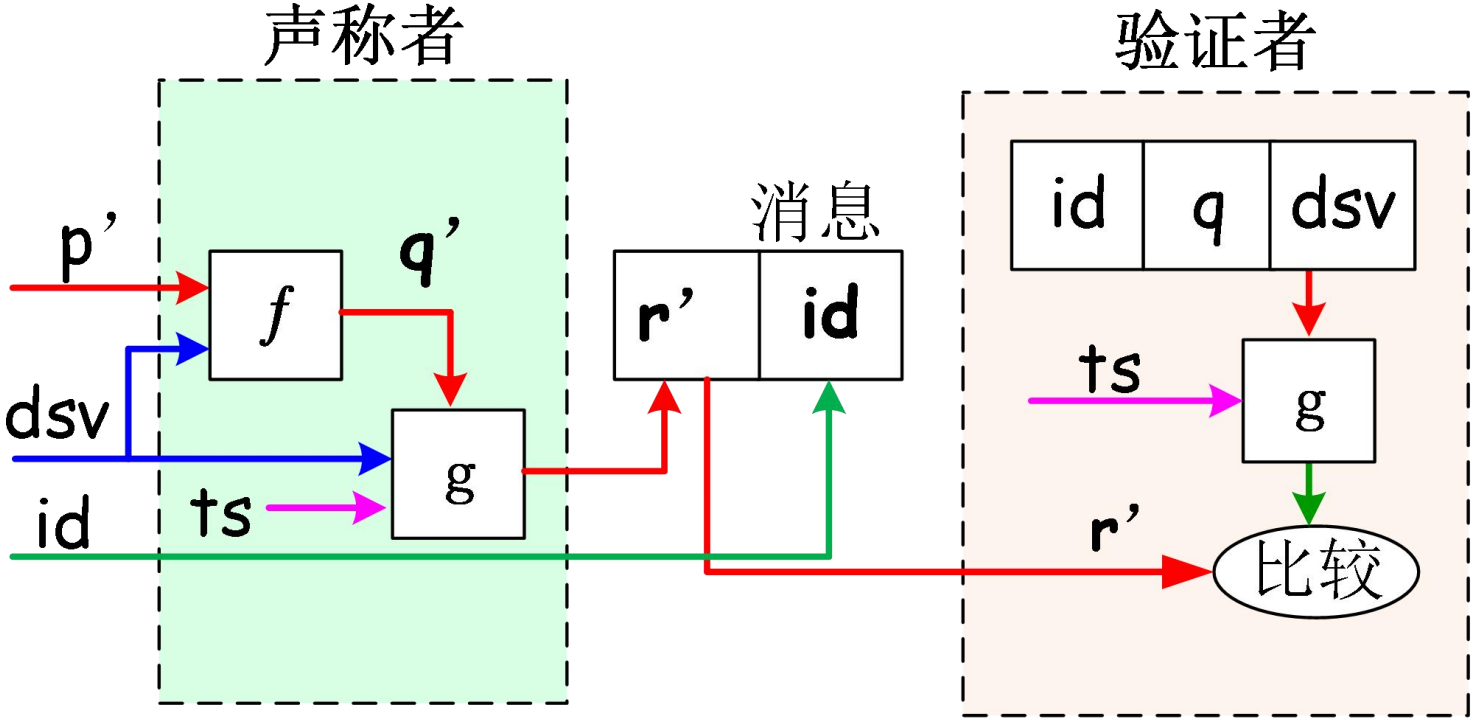
- 非重复值：时间戳、随机数等

一次口令机制

- 目的：确保每次认证使用的口令不同，以便对付重放攻击。
- 确定口令的方法
  - 两端共同拥有一串随机口令，在该串的某一位置保持同步
  - 两端共同使用一个随机序列生成器，在该序列生成器的初态保持同步
  - 使用时戳，两端维持同步的时钟

一次口令认证机制

- 方案

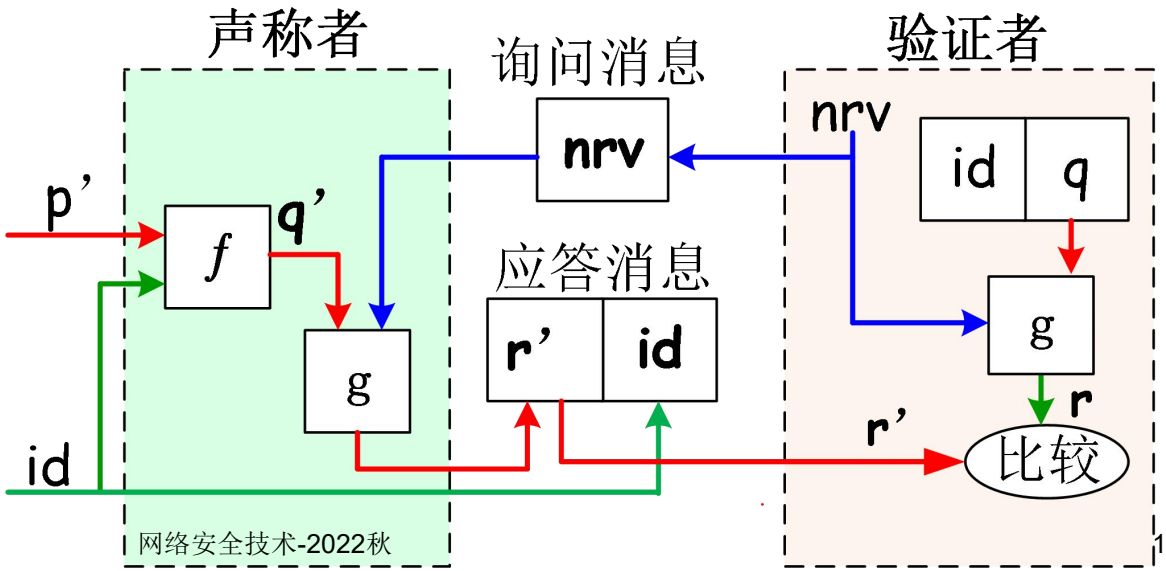


- 机制的安全强度
  - ¬dsv∩p 或者 dsv∩¬p 或者¬dsv∩q，无法验证通过
  - dsv∩q∩ts 或者dsv∩p∩ts，才可以通过验证

4.2.2 挑战-应答机制

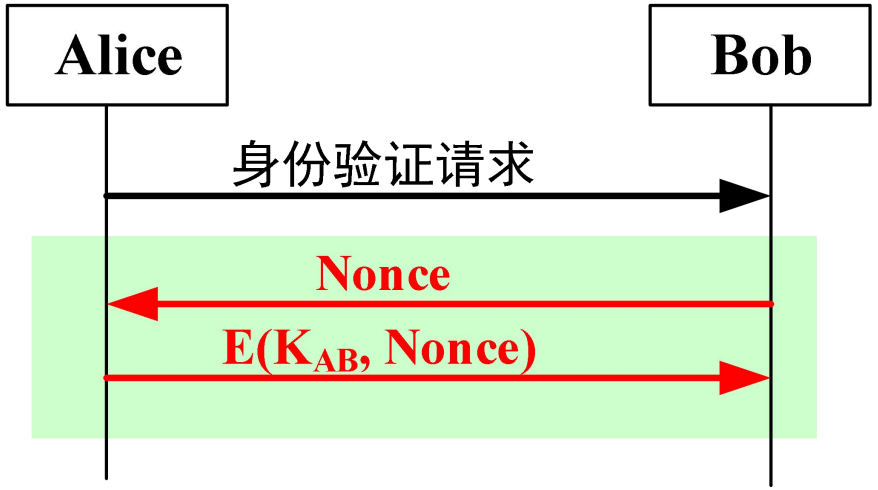
- 抗重放攻击机制中存在两个重要的问题：
  - 为了两端都知道nrv值，需要维持同步
  - 验证者要知道nrv值是否被重复使用过（比较困难）
- 使用挑战-应答机制可以克服这些问题，而且可以提高抗重放攻击的能力，但通信代价较高

- 挑战应答机制

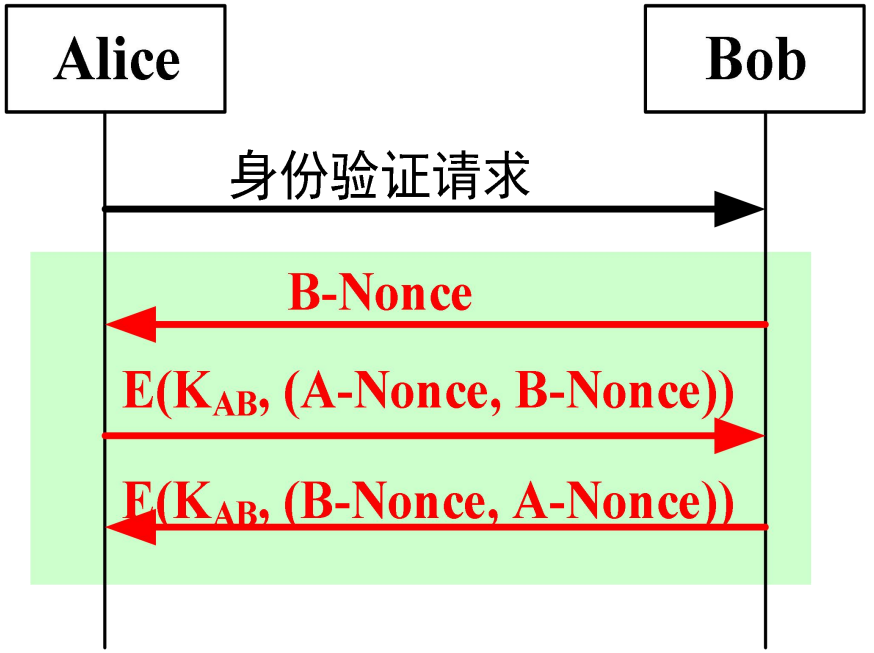


基于对称密码的挑战-应答方法

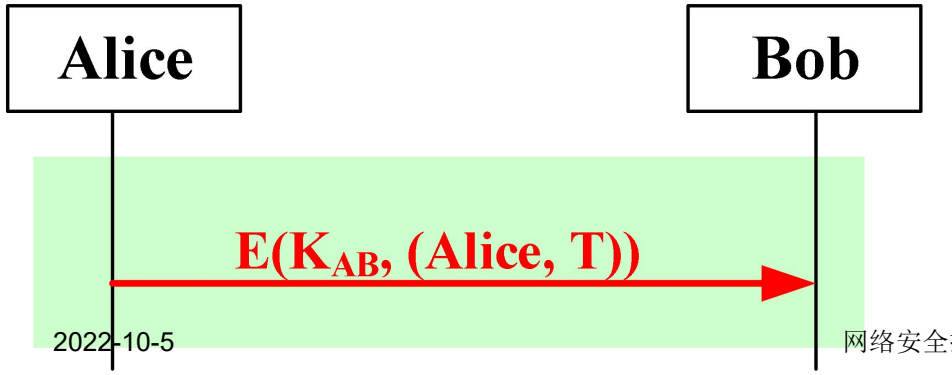
- Nonce挑战



- 双向验证

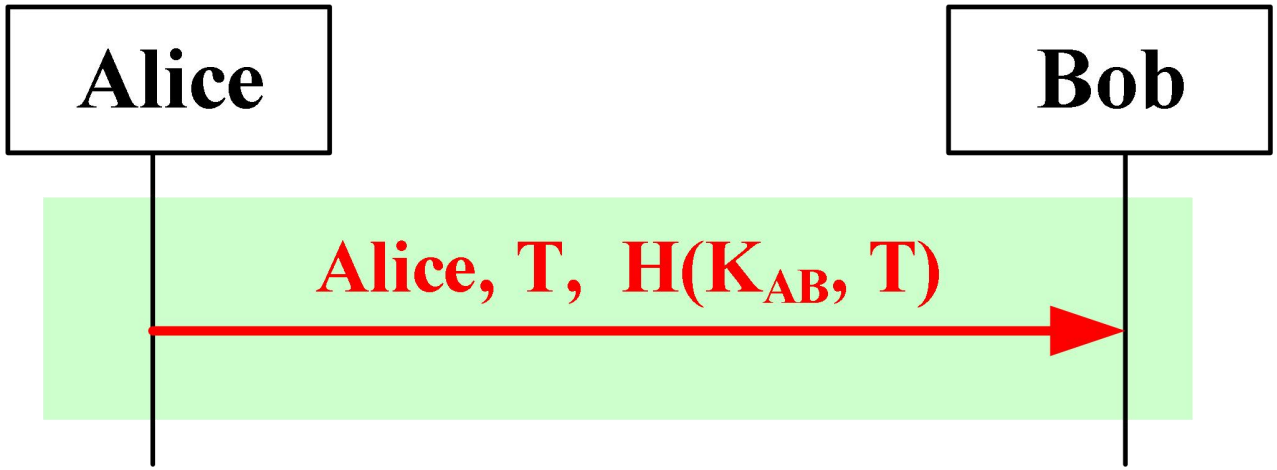


- 时间戳挑战



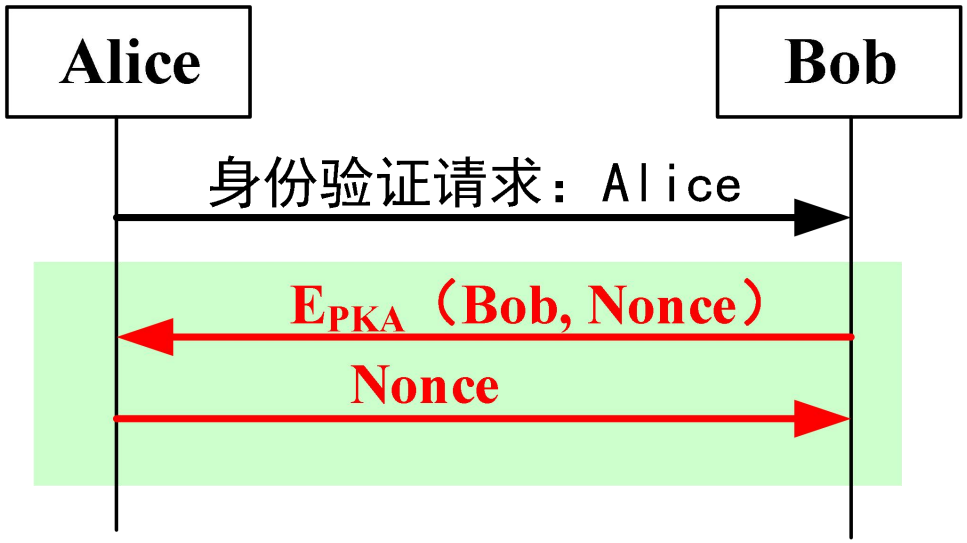
基于MAC的挑战-应答方法

- 可以保护消息的完整性，并且同时使用了一个秘密(K_{AB})
- 时间戳T作为明文传输

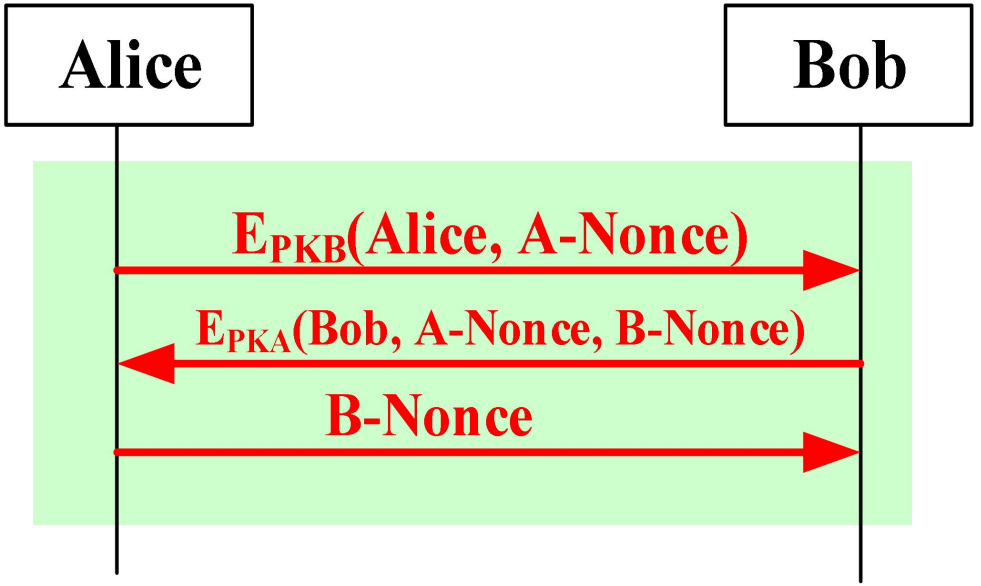


基于公钥加密的挑战-应答方法

- 单向验证

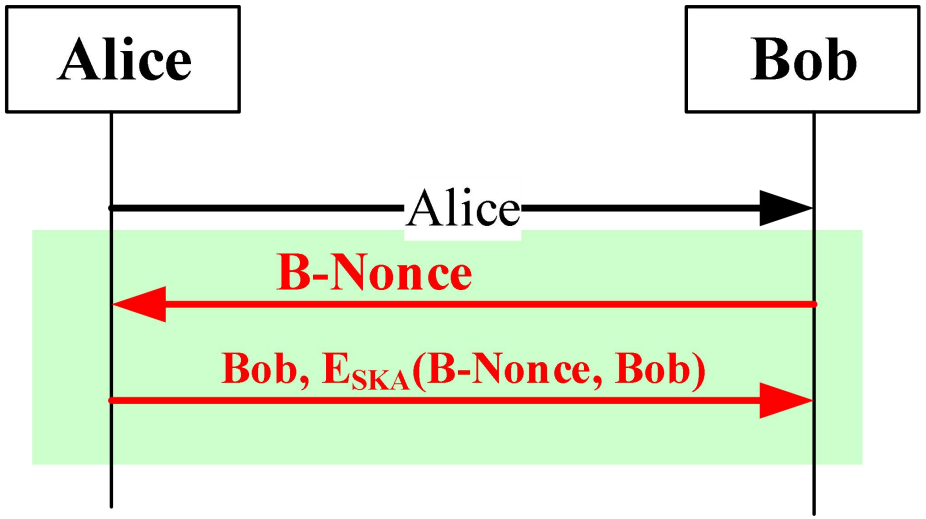


- 双向验证

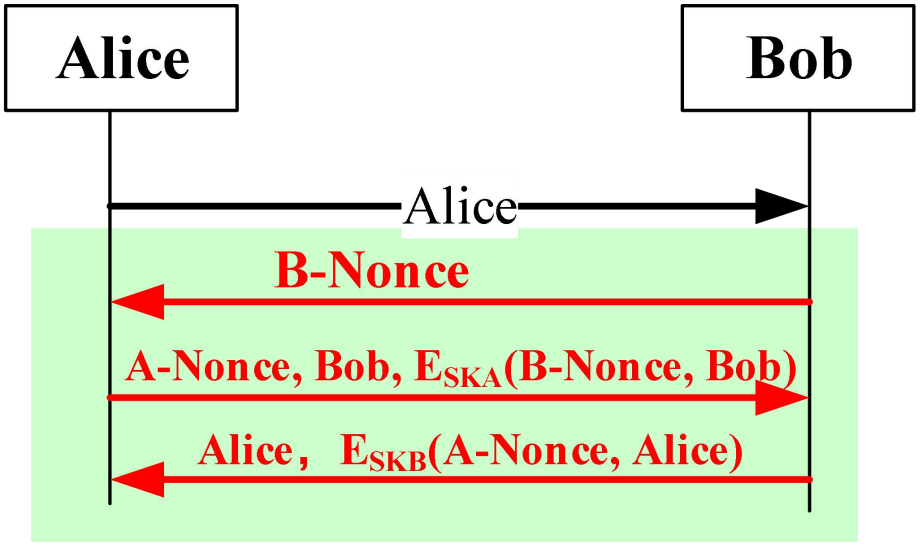


基于数字签名的挑战-应答方法

- 单向验证



- 双向验证



4.2.3 基于地址的机制

- 基于地址的机制假定声称者的可认证性是以呼叫的源地址为基础
- 前提：通信网络中的呼叫地址是可识别的
- 方法
  - 验证者对每个主体均保持一个合法呼叫地址文件
  - 验证中，验证者对呼叫地址做合法性检查或者清除源呼叫并向一个合法地址回呼
- 潜在问题：需要维持主机与网络地址的联系，在某些环境下(移动环境)代价高、不可操作、不安全

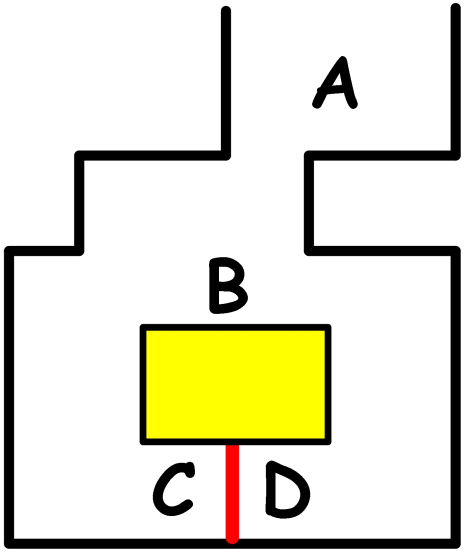
4.2.4 零知识证明

- 零知识证明（Zero-knowledge proof）是由Goldwasser等人20世纪80年代初提出的。它指的是证明者能够在不向验证者提供任何有用的信息的情况下，使验证者相信某个论断是正确的。
- 举例：A向B证明自己拥有某个房间的钥匙（假设该房间只能用钥匙打开锁，而其他任何方法都打不开）
- 方法一：A把钥匙出示给B，B用这把钥匙打开房间的锁
- 方法二：B确定该房间内有某一物体，A用自己拥有的钥匙打开房间把物体出示给B
- 第二种方法的好处：B始终不能看到钥匙的样子，从而避免了钥匙的泄露

零知识证明

- 零知识证明技术在现代密码中处于核心位置，其允许证明者向验证者证实一个论断，却不泄露任何知识
- Quisquater等提出一种形象的零知识协议例子

零知识洞穴：只有知道咒语的人才能打开C、D之间的门，证明者P要向验证者V证明他知道咒语，但不能泄露咒语



- 1）V站在A点
- 2）P走进洞穴，到达C点或D点
- 3）在P消失在洞穴中后，V走到B点
- 4）V指定P从左通道或右通道出来
- 5）P按要求出洞
- 6）P和V重复步骤1）至5）n次

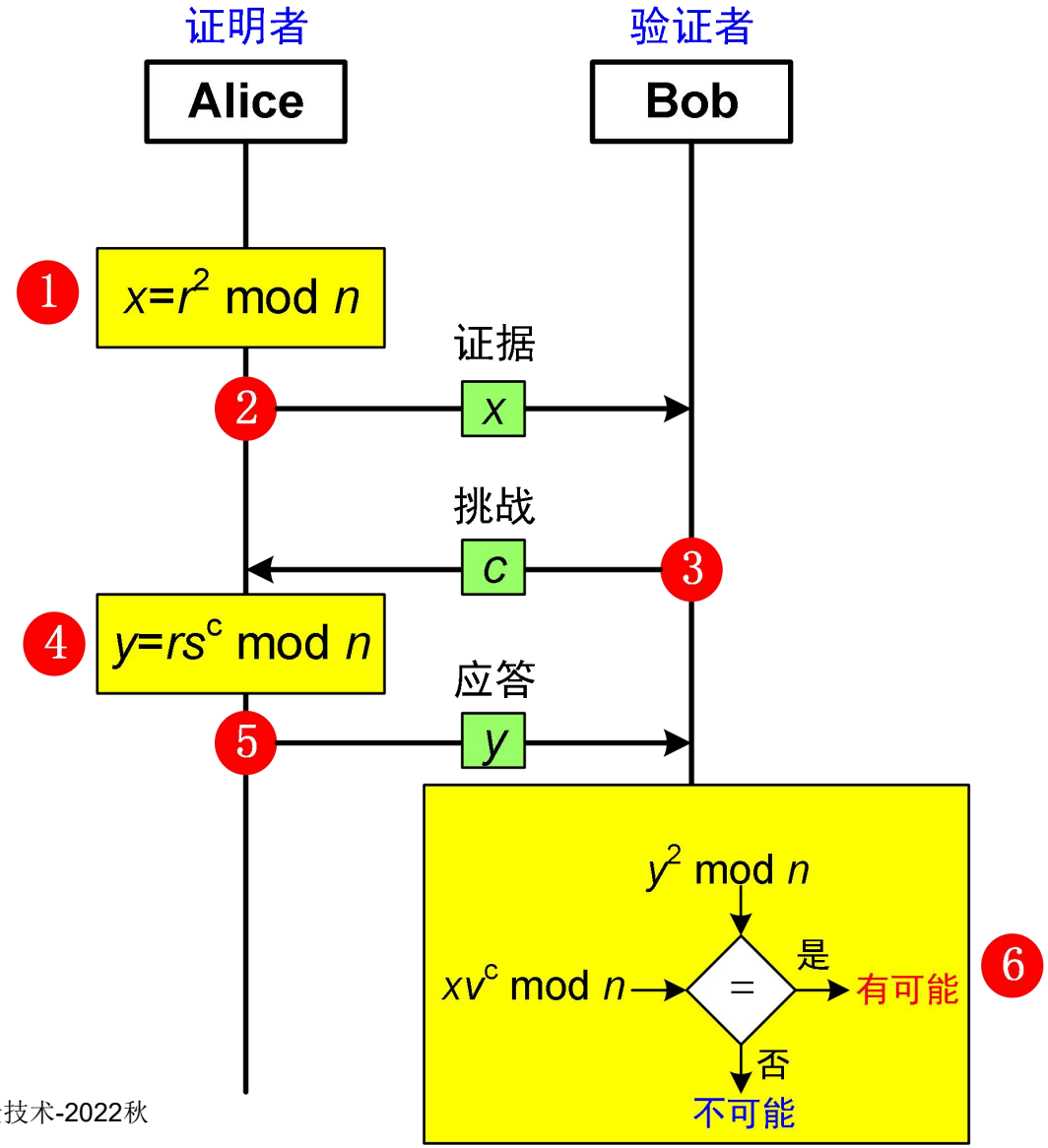
- 分析：如果P知道咒语，则100%成功，否则，$1/2^n$

基于零知识证明的身份认证

- 应用：一个安全的身份识别协议至少要满足以下条件
  - 证明者Alice能够向验证者Bob证明他的确是Alice
  - 在证明者Alice向验证者Bob证明他的身份后，验证者Bob没有获得任何有用的信息，Bob不能模仿Alice向第三方证明他是Alice

基于零知识证明的身份认证

- **Fiat-Shamir协议**
  - s Alice的私钥(1到n之间随机选择的一个秘密数)
  - *v* Alice的公钥 $v=s^2 \bmod n$
  - *r* 随机数
  - *n*是公开的



作业1：请说明如果Alice正确预测到c=1, 她应该发送什么样的x和y就能通过验证？如果c=0呢？

4.2.5 基于个人特征的机制

- 相当于应用了一个特殊的口令，只是这些口令与传统口令不一致，不能由别人使用
- 相关技术
  - 指纹识别
  - 声音识别
  - 手迹识别
  - 视网膜扫描
  - 手形
  - 步态等

生物特征识别

- 当前身份认证主要技术
  - 标志物认证（如证件、卡和钥匙等）
  - 标识知识认证（如用户口令）
- 存在的缺陷
  - 标志物：容易丢失和被伪造
  - 标识知识：容易遗忘和记错
- 克服传统身份认证的缺陷的方法：生物特征识别技术

生物特征识别

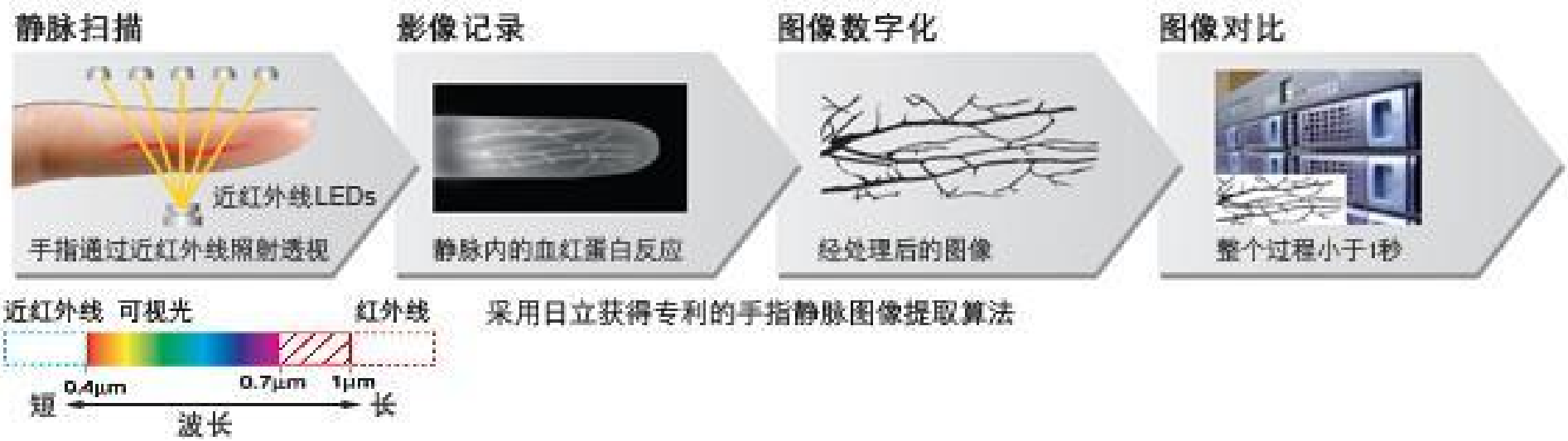
- 生物特征识别：将人类固有的行为特征或者生理特征通过计算机进行收集整理，用来作为个人身份鉴别。
- 优点：不易丢失、窃取、共享、遗忘，且信息量足够不易破译。
- 缺点
  - 生物特征有限，丢失后无法及时更新，导致永久丢失

- 分类
  - 基于人的行为模式：步态、基键动力学、手写签名等
  - 基于人的生理特征

生物、特征	优点	缺点	安全等级	采集方式	可变	拒识率 FRR	误识率 ERR	特征性质
声音	方便	噪音	中	非接触	是			内部
人脸	易采集	光线影响	中	非接触	是			外部
指纹	广泛	磨损、油污	高	接触	是			外部
虹膜	精度高	光线、眼镜	很高	非接触	否			内部
指静脉	活体识别	无	很高	非接触	未证实			内部

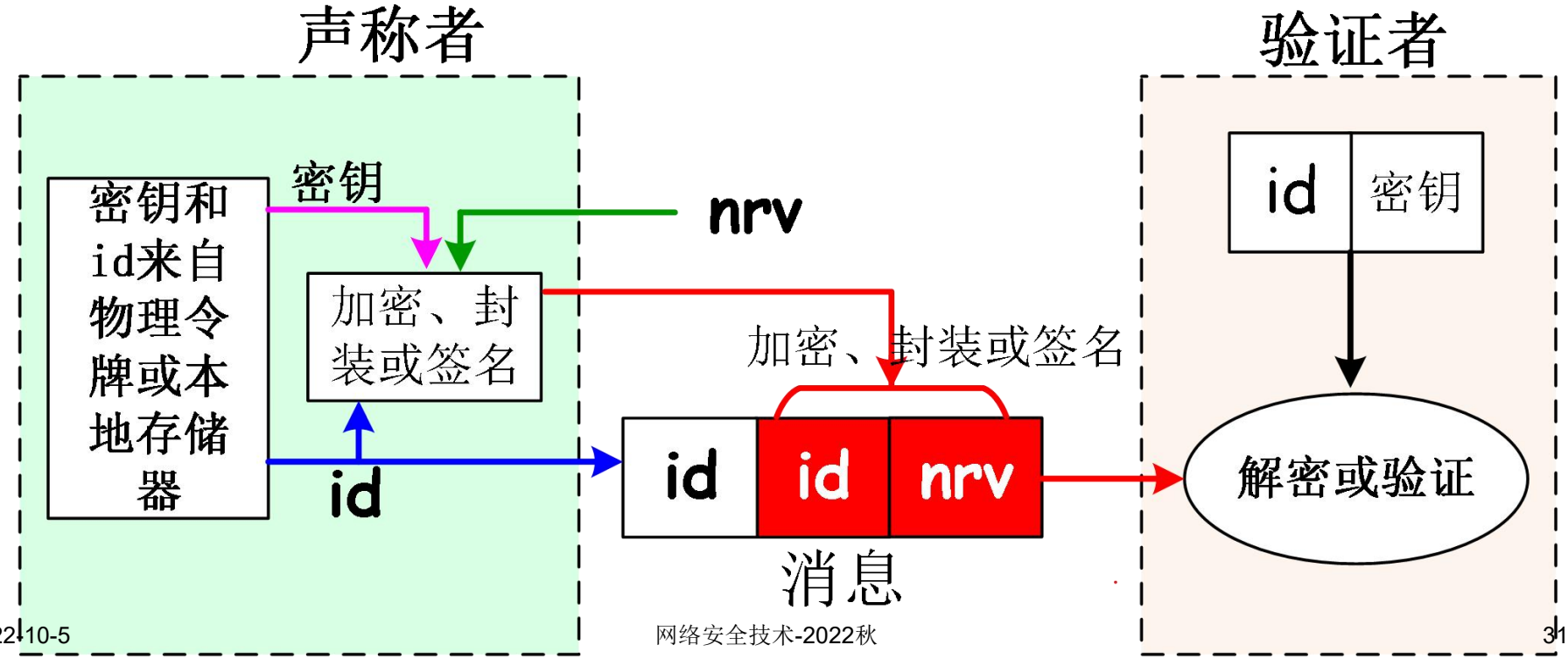
指静脉

- 原理
  - 静脉位置浅，生物组织光学窗口
  - CCD, 红外LED



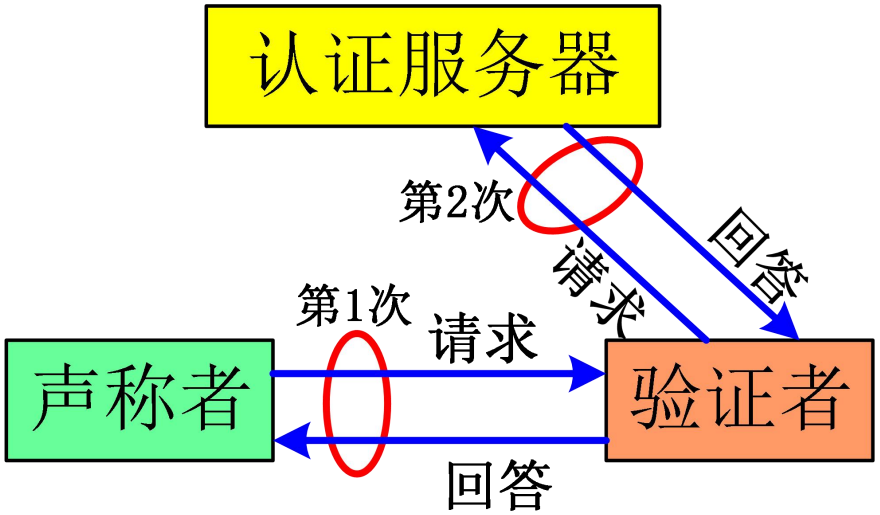
4.3 基于密码的认证机制

- 基本原理：基于声称者知道某一秘密密钥这一事实，使验证者信服声称是声称者的声称。
- 对称密码技术和公钥密码技术都可以使用



4.3.1 在线认证服务器

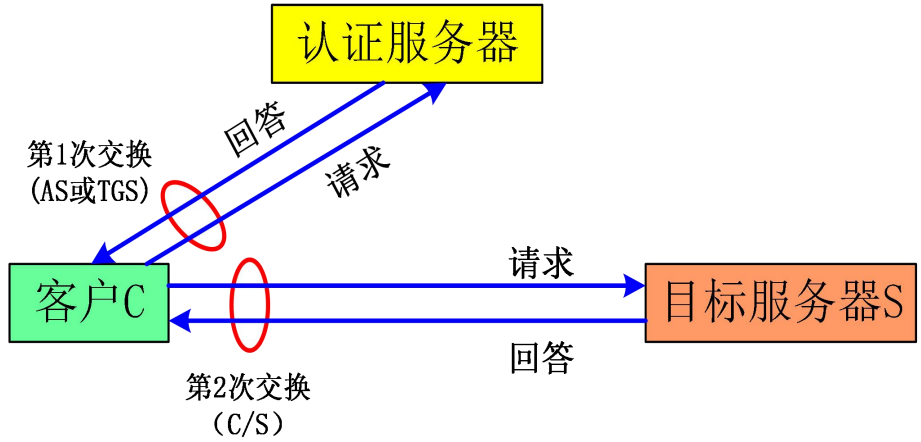
- 大型网络中，用户两两之间利用对称密码认证技术不实用，会导致**密钥爆炸**。
- 解决方法：**在线认证服务器**（仍然利用**对称密码技术**，每个用户或主机与认证服务器之间共享密钥）
- 方法1
  - 声称者用自己的密钥加密或封装一个消息发送给验证者
  - 验证者将该消息发送给认证服务器进行验证



  - 说明：验证者与认证服务器之间利用**共享密钥进行安全通信**，认证服务器利用与声称者之间的**共享密钥对声称者认证**

在线认证服务器

- 方法2 (典型实例：kerberos)
  - 声称者与认证服务器进行信息交换获得一个许可证（票据ticket）
  - 声称者将许可证发送给验证者
  - 说明
    - 第1次交换：使用声称者和认证服务器之间的共享密钥进行保护
    - 第2次交换：声称者出示的许可证只有知道认证服务器和验证者的共享密钥的实体才能接收



4.3.2 离线认证服务器

- 在使用公钥密码技术的认证机制中不需要在线认证服务器
- 验证者利用离线服务器需要获得主机的合格公钥和证书撤销列表
- 离线认证服务器与在线认证服务器比，优点
  - 没有建立呼叫与访问在线认证服务器的响应延迟
  - 服务器通信量少
  - 脱离服务器实体仍能继续认证
  - 服务器不需要可信的实现

4.3.3 个人认证

- 密码技术不适合在个人认证中直接应用
- 个人认证一般采用：口令机制与基于密码技术机制相结合的方法
- 基本方法
  - 人使用口令向器件认证自己
  - 器件使用密码技术向最终的验证者验证自己
- 两种情况
  - 从口令推导密钥：在身份-口令对和密码系统的秘密密钥之间建议一一映射
  - 使用智能卡：用于认证目的的智能卡的使用与拥有者输入的PIN有关

4.4 设计认证协议注意的问题

- 设计认证协议
  - 需要考虑各种攻击
  - 需要考虑针对认证协议的各种分析方法
- 需要考虑的问题
  - 主动攻击
  - 认证的连续性

主动攻击

- 重放攻击
  - 针对同一验证者的重放：在认证消息中包含一个非重复值
  - 针对不同验证者的重放：在受保护的消息中包含目标验证者的标识符。
- 反射攻击
- 中间人攻击(Man-in-the-MiddleAttack，“MITM”)

认证过程与密钥建立过程进行组合（机密信息加密、带外认证、地址异常等）

- 竞争攻击：利用认证服务器在处理同时发来的两个以上的认证请求时有漏洞
- 伪装成login程序的攻击

认证的连续性

- 认证的连续性是指在整个通信过程中可以持续对通信中的一方进行认证
- 维持认证的方法：同时使用连接完整性服务，将认证与密钥（用于连接完整性）交换功能结合使用

认证协议的安全性分析

- 非形式化分析方法
  - 又称攻击检验方法，利用已知的各种攻击方法对协议进行攻击
  - 不能全面客观地分析
- 形式化分析分析（略）
  - 前提假设：完美加密、协议参与实体、入侵者的知识与能力
  - 基本方法
    - BAN（逻辑分析法）、模型检测方法、通用形式分析法等

4.5 认证协议

- Kerberos
- X.509协议

Kerberos

- Kerberos是分布式不可信环境下的应用层认证服务，1988年由MIT开发
- Kerberos 既是鉴别协议，同时也是 KDC，它已经变得很普及，现在是互联网建议标准。
- Kerberos要解决的问题

假设在一个开放的分布式环境中，工作站的用户希望访问分布在网络各处的服务器上的服务。希望服务器能够将访问权限限制在授权用户范围内，并且能够认证服务请求。(注:工作站无法准确判断它的终端用户以及请求的服务是否合法）

Kerberos

- 分布式场景面临的主要威胁
  - **身份假冒**：工作站上的用户可以冒充另一用户操作（冒充另一个用户）
  - **地址假冒**：用户可能改变一个工作站的网络地址，从该机上发送伪造的请求（冒充另一台工作站）
  - **重放**：用户可能监听信息或者使用重放攻击，从而获得服务或者破坏正常操作

非授权用户可能会获得他没有被授权得到的服务和数据

  - **服务器假冒**:用户登录一个非法服务器并使用上面的服务，造成自己的机密信息泄露

Kerberos

- Kerberos采用集中的可信第三方认证服务器实现用户对服务器的认证和服务器对用户的认证
- 常用的两个版本：V4仍然被采用，V5修正了v4的安全缺陷，成为Internet标准草案RFC4120
- 名称来源于希腊神话，Kerberos是地狱入口的守护者，通常有三个头
- 设计者的设计初衷是要用Kerberos的三个组件来守卫网络之门
  - 身份验证服务器(AS)
  - 票据许可服务器(TGS)
  - Kerberos数据库

Kerberos

- Kerberos 只用于客户与服务器之间的鉴别，而不用于人对人的鉴别。
- Kerberos 要求所有使用 Kerberos 的主机必须在时钟上进行“松散的”同步
- 所谓“松散的”同步是要求所有主机的时钟误差不能太大，例如，不能超过 5 分钟的数量级。

Why?

Kerberos

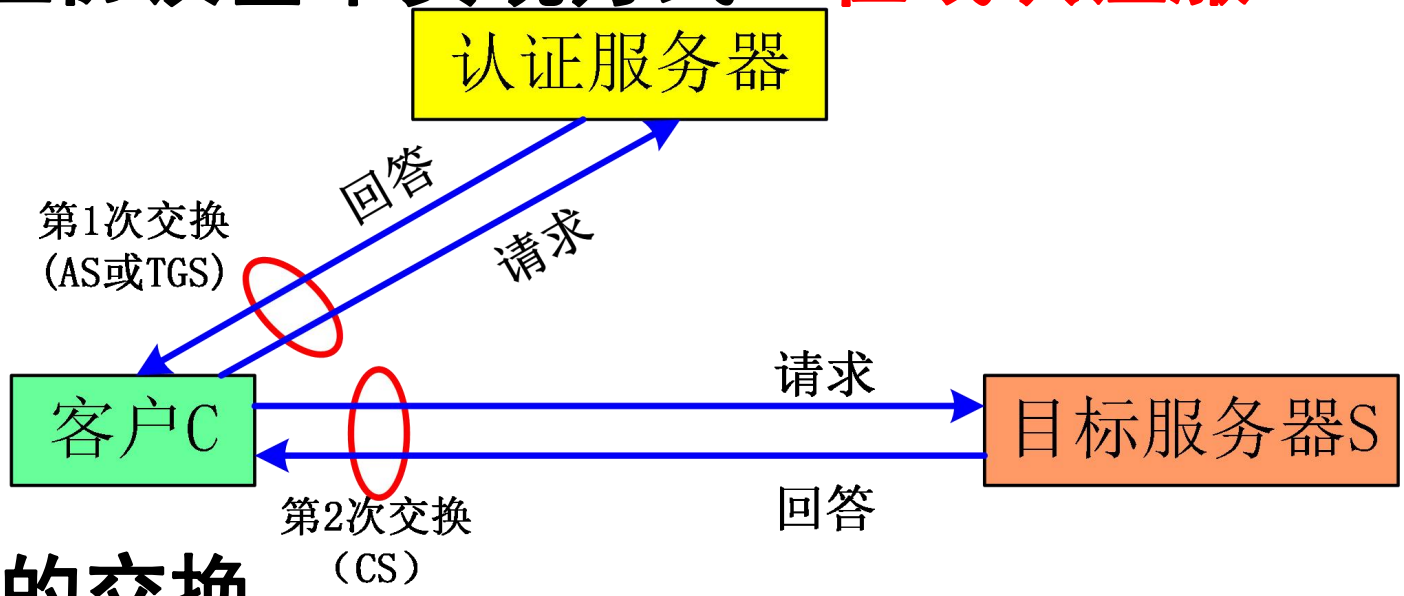
- 基本思想

  能正确对信息进行解密的用户就是合法用户

- 特点
  - 分布式认证协议
  - 使用票据进行认证
  - 避免在网络上传输密钥
  - 基于可信第三方
  - Kerberos使用对称加密方案进行加密

Kerberos 认证协议

- Kerberos 认证协议基本实现方式：在线认证服务器



- 两种不同类型的交换
  - 身份认证服务(AS，Authentication Server)交换
  - 客户/服务器(CS)认证交换

一个简单的认证过程

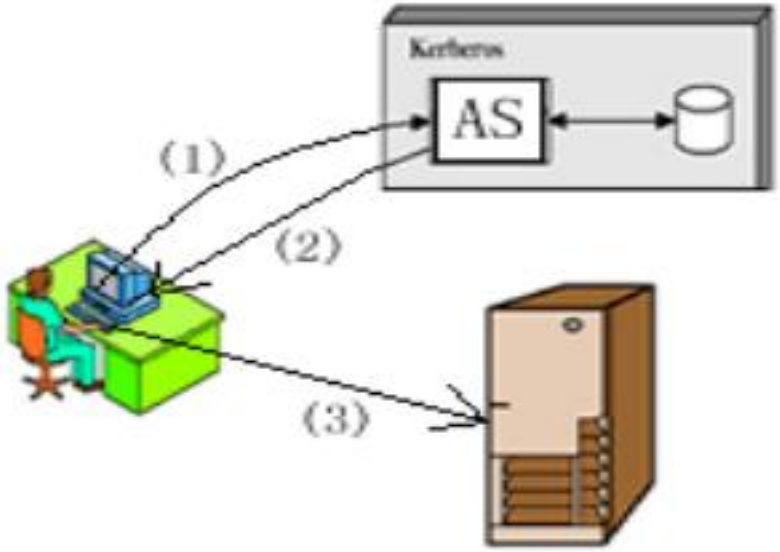
认证过程

(1) C → AS : IDc || Pc || IDv

(2) AS → C : Ticket

(3) C → V : IDc || Ticket

$Ticket = E(K_v, [IDc \| ADc \| IDv])$



字符说明

C ＝客户端；AS＝认证服务器

V ＝服务；IDc＝用户名称

IDv＝服务器名称；||＝级联

ADc＝客户端的IP地址

Pc＝客户端上用户密码（口令）

Kv＝AS与服务器V共享的加密密钥

一个简单的认证过程

- 安全性分析
  - 身份假冒
  - 地址假冒
- 存在的问题
  - 口令泄露：用明文传递的用户口令会被窃听
- 解决办法：不在线路上传输口令，而是基于能正确对信息进行解密的用户就是合法用户

一个改进后的简单的认证过程

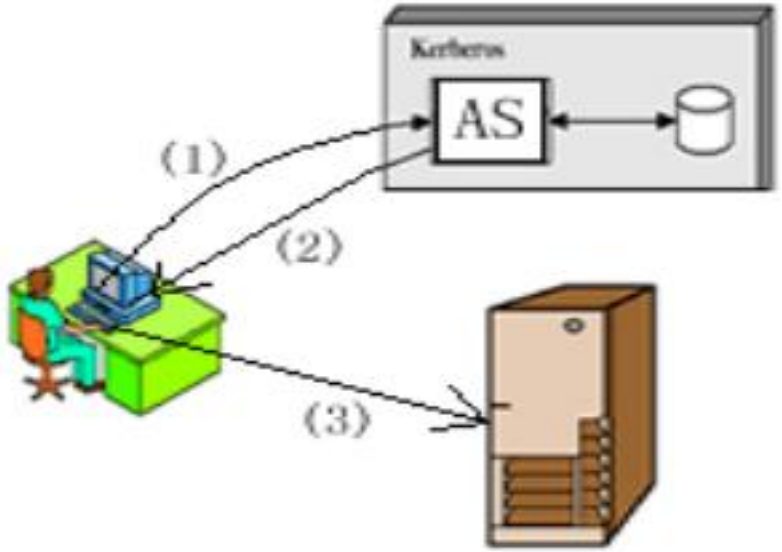
认证过程

(1) C → AS : IDc || IDv

(2) AS → C : E(Kc, Ticket)

(3) C → V : IDc || Ticket

Ticket = $E(K_v, [IDc \| ADc \| IDv])$



字符说明

|| ＝级联

C ＝客户端

AS＝认证服务器

V ＝服务（服务器）

IDc＝用户名称

IDv＝服务器名称

ADc＝客户端的IP地址

Kc＝AS与客户端C共享的加密密钥，由用户口令Pc产生

Kv＝AS与服务器V共享的加密密钥

一个改进后的简单认证过程

- 安全能力分析
  - 假冒用户身份
  - 假冒工作站地址
  - 窃听口令
- 存在的问题
  - 重复身份认证：用户每次访问一种新服务就需要进行一次身份认证，导致多次身份认证，频繁使用Kc
  - 票无限期使用
- 解决办法
  - 引入票据授权服务器(TGS)
  - 票中增加产生时间(时间戳)和有效期信息

一个更安全的认证过程

- 三种不同类型的交换
  - 身份认证服务(AS，Authentication Server)交换
  - 票据认可服务(TGS，Ticket-Granting Server)交换
  - 客户/服务器(CS)认证交换

一个更安全的认证过程

- 每次用户登录执行一次

(1) C → AS : $ID_c \| ID_{tgs}$

(2) AS → C : $E(K_c, Ticket_{tgs})$

- 每种类型的服务执行一次

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

(4) TGS → C : $Ticket_V$

- 每个服务会话执行一次

(5) C → V : $ID_c \| Ticket_V$

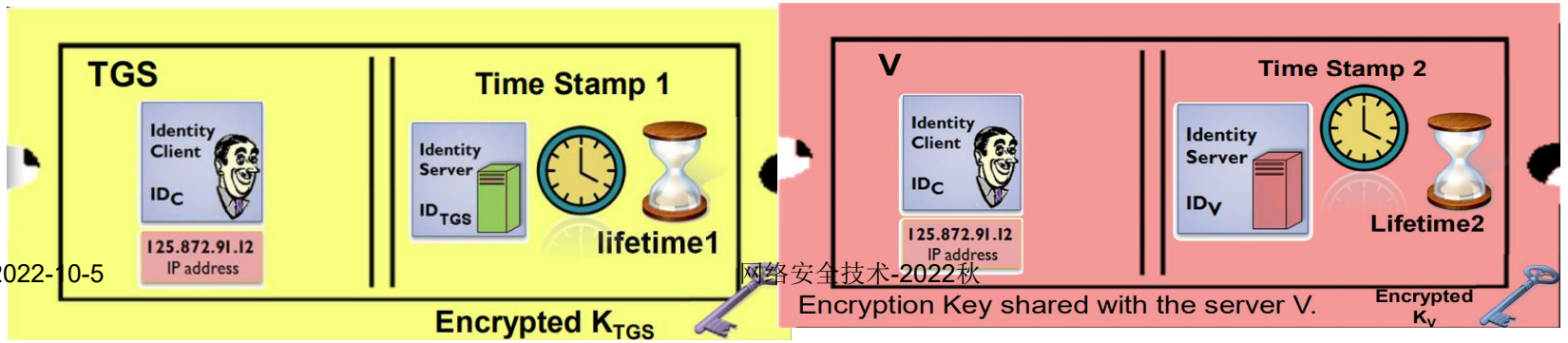
票据

(1) $Ticket_{tgs}$：票据授权票据

(2) $Ticket_V$：服务授权票据

$Ticket_{tgs} = E(K_{tgs}, [ID_c \| AD_c \| ID_{tgs} \| TS_1 \| Lifetime_1])$

$Ticket_V = E(K_v, [ID_c \| AD_c \| ID_v \| TS_2 \| Lifetime_2])$

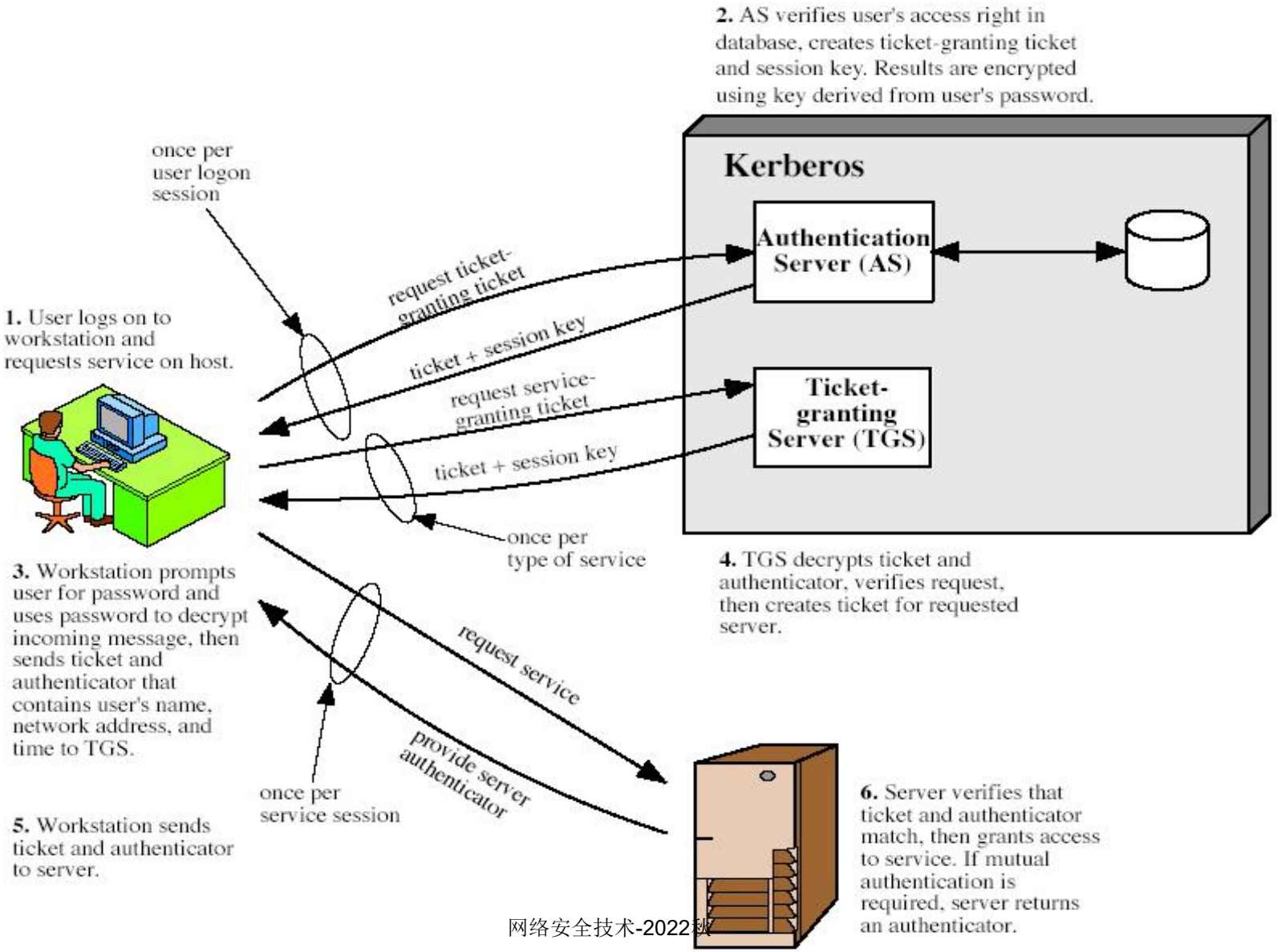


[LoQM]

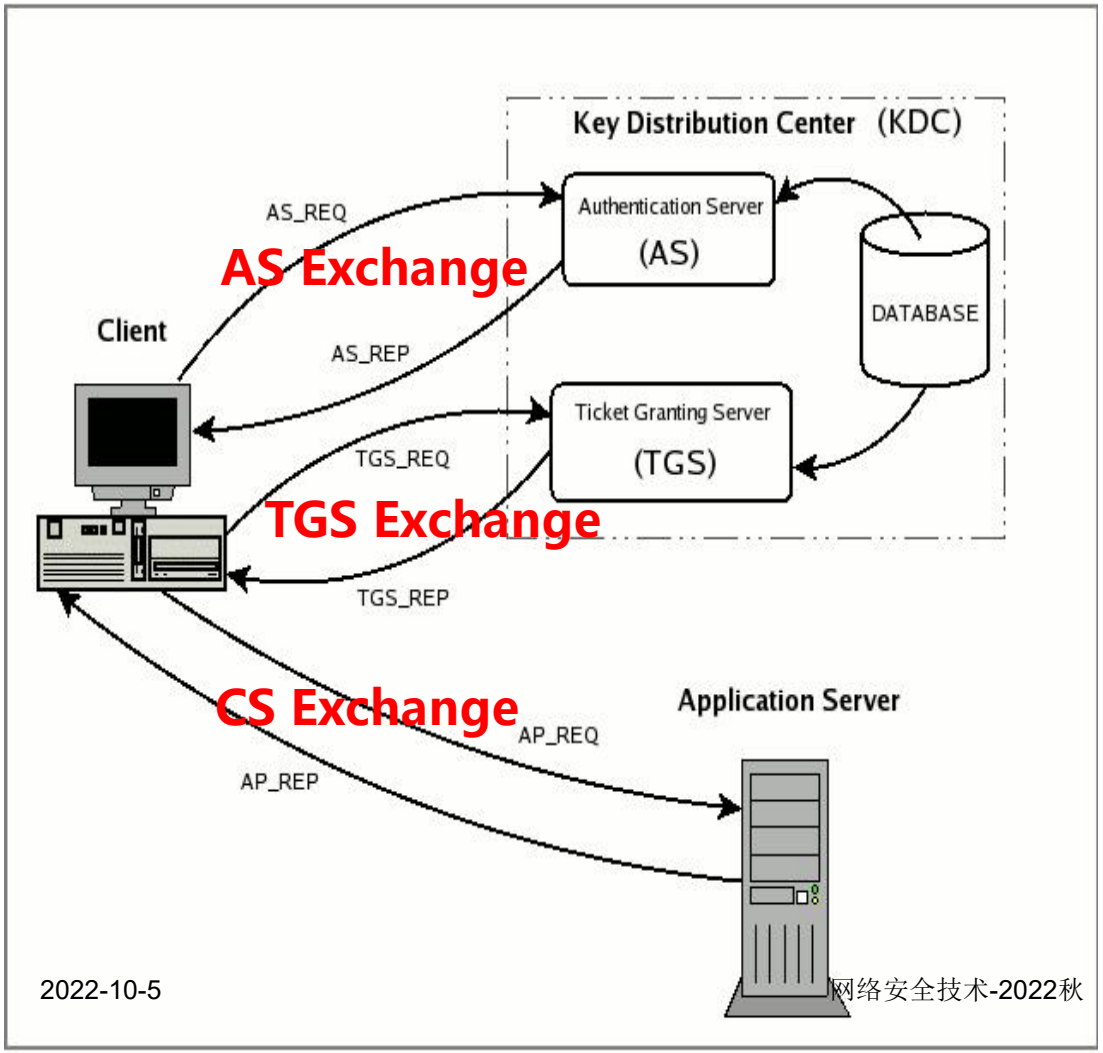
一个更安全的认证过程

- 能力分析
  - 身份假冒、地址假冒
  - 口令窃听
  - 每个用户对话只请求一次用户身份认证，保护用户主密钥
- 存在的问题
  - 重放攻击：票被重放
  - 服务器假冒攻击：即客户端未对服务器进行认证
- 解决办法：Kerberos(v4)

Kerberos(V4)


2. AS verifies user's access right in database, creates ticket-granting ticket and session key. Results are encrypted using key derived from user's password.
Kerberos
Authentication Server (AS)
Ticket-granting Server (TGS)
once per user logon session
request ticket-granting ticket
ticket + session key
request service-granting ticket
ticket + session key
once per type of service
1. User logs on to workstation and requests service on host.
3. Workstation prompts user for password and uses password to decrypt incoming message, then sends ticket and authenticator that contains user's name, network address, and time to TGS.
4. TGS decrypts ticket and authenticator, verifies request, then creates ticket for requested server.
request service
provide server authenticator
once per service session
5. Workstation sends ticket and authenticator to server.
6. Server verifies that ticket and authenticator match, then grants access to service. If mutual authentication is required, server returns an authenticator.

Kerberos认证



1. Authentication Server Request (AS_REQ)
2. Authentication Server Reply (AS_REP)
3. Ticket Granting Server Request (TGS_REQ)
4. Ticket Granting Server Replay (TGS_REP)
5. Application Request (AP_REQ)
6. Application Reply (AP_REP)

1. 用于获取票据授权票据的认证服务交换	
(1) C → AS	$ID_c \| ID_{tgs} \| TS_1$
(2) AS → C	$E(K_{c,} [K_{C,tgs} \| ID_{tgs} \| TS_2 \| Lifetime_2 \| Ticket_{tgs}])$
	$Ticket_{tgs}=E(K_{tgs,} [K_{C,tgs} \| ID_c \| AD_c \| ID_{tgs} \| TS_2 \| Lifetime_2])$

2. 用于获取服务授权票据的票据授权服务交换	
(3) C → TGS	$ID_v \| Ticket_{tgs} \| Authenticator_C$
(4) TGS → C	$E(K_{C,tgs}, [K_{C,V} \| ID_V \| TS_4 \| Ticket_V])$
	$Ticket_{tgs}=E(K_{tgs,} [K_{C,tgs} \| ID_c \| AD_c \| ID_{tgs} \| TS_2 \| Lifetime_2])$
	$Ticket_V= E(K_V, [K_{C,V} \| ID_c \| AD_c \| ID_v \| TS_4 \| Lifetime_4])$
	$Authenticator_C= E(K_{C,tgs}, [ID_c \| AD_c \| TS_3])$

3. 用于获取服务而进行的客户端/服务器认证交换	
(5) C → V	$Ticket_V \| Authenticator_C$
(6) V → C	$E(K_{C,V} , [TS_5 +1])$ （用于双向认证）
	$Ticket_V= E(K_V, [K_{C,V} \| ID_c \| AD_c \| ID_v \| TS_4 \| Lifetime_4])$
	$Authenticator_C= E(K_{C,V}, [ID_c \| AD_c \| TS_5])$

Kerberos各消息中参数作用

认证服务交换的消息

消息(1)	客户端请求票据授予票据　　$ID_C \| \| ID_{tgs} \| \| TS_1$
ID_C	由客户端告诉AS用户的身份
ID_{tgs}	告诉AS用户请求访问TGS
TS_1	使AS可以验证客户端的时钟与AS的时钟是否同步
消息(2)	AS向客户端返回票据授予票据　　$E(K_c, [K_{C,tgs} \| \| ID_{tgs} \| \| TS_2 \| \| Lifetime_2 \| \| Ticket_{tgs}])$
K_C	加密是基于用户口令的，这使得AS和客户端可以验证口令，并保护消息(2)的内容
$K_{C,tgs}$	由AS创建的用户端可以访问会话密钥的复制，它允许客户端和TGS之间可以安全交换信息，而不需要它们共享永久的密钥
ID_{tgs}	确认此票据是为TGS生成的
TS_2	通知客户端此票据的发放时间
$Lifetime_2$	通知客户端此票据的有效期
$Ticket_{tgs}$	用户端用来访问TGS的票据

Kerberos各消息中参数作用

票据授权服务交换的消息

消息 (3)	客户端请求服务授予票据 $ID_V \| \| Ticket_{tgs} \| \| Authenticator_C$
ID_V	告诉TGS用户请求访问服务器V
$Ticket_{tgs}$	向TGS确认此用户是经过AS认证过的
$Authenticator_C$	由客户端创建，用于认证票据（抗重放攻击）
消息(4)	TGS返回服务授予票据 $E(K_{C,tgs}, [K_{C,V} \| \| ID_V \| \| TS_4 \| \| Ticket_V])$
$K_{C,tgs}$	只由C与TGS共享的私密密钥，用来保护消息(4)
$K_{C,V}$	由TGS创建的用户端可以访问会话密钥的复制，它允许客户端和服务器之间可以安全交换信息，而不需要它们共享永久的密钥
ID_V	确认该票据是为V生成的
TS_4	通知客户端此票据的发放时间
$Ticket_{tgs}$	可重用，不需要用户重复输入口令
$Ticket_V$	用户端用来访问服务器V的票据
K_{tgs}	票据由只有AS和TGS知道的密钥加密，以防篡改

Kerberos各消息中参数作用

票据授权服务交换的消息(续)

$K_{C,tgs}$	TGS可以访问会话密钥的副本，用来解密认证符，认证票据
ID_C	标识此票据的合法所有者
AD_C	防止其他工作站使用票据
ID_{tgs}	使服务器确认票据本正确的解密
TS_2	通知TGS此票据的发放时间
$Lifetime_2$	票据的生存期，防止票据过期后的重放
$Authenticator_C$	由客户端创建，使TGS确认票据出示者即是被授予票据的客户端，有效期很短，（抗重放攻击）
$K_{C,tgs}$	认证符由只有客户端和TGS知道的密钥加密，防止篡改
ID_C	必须与票据的ID匹配，从而认证票据
AD_C	必须与票据的网络地址（即IP地址）匹配，从而认证票据
TS_3	通知TGS认证符的生成的时间

Kerberos各消息中参数作用

客户端与服务器认证交换的消息

消息 (5)	客户端申请服务 $Ticket_V$ || $Authenticator_C$
$Ticket_V$	使服务器确信此用户已经通过了AS的验证
$Authenticator_C$	由客户端创建，用于验证票据，防止重放攻击
消息 (6)	用户对服务器的认证（可选） $E(K_{C,V}, [TS_5+1])$
$K_{C,V}$	使客户端确认消息来自V
TS_5+1	使客户端确认这个应答不是先前应答的重放
$Ticket_V$	可重用，不需要C每次访问同一个V都向TGS申请新的票据
K_V	票据由只有TGS和V知道的密钥加密，以防篡改
$K_{C,V}$	客户端可以访问会话密钥的复制，用来解密认证符，从而认证票据
ID_C	标识票据的合法拥有者
AD_C	防止其他工作站使用票据
ID_V	使服务器确认票据被正确解密

Kerberos各消息中参数作用

客户端与服务器认证交换的消息(续)

TS_4	通知服务器V此票据的发放时间
$Lifetime_4$	票据的生存期，防止票据过期后的重放
$Authenticator_C$	由客户端创建，使V确认票据出示者即是被授予票据的客户端，有效期很短，防止重放攻击
ID_{tgs}	使服务器确认票据本正确的解密
$K_{C,V}$	认证符由只有客户端和服务器知道的密钥加密，防止篡改
ID_C	必须与票据的ID匹配，从而认证票据
AD_C	必须与票据的网络地址（即IP地址）匹配，从而认证票据
TS_5	通知服务器认证符的生成的时间

Kerberos域

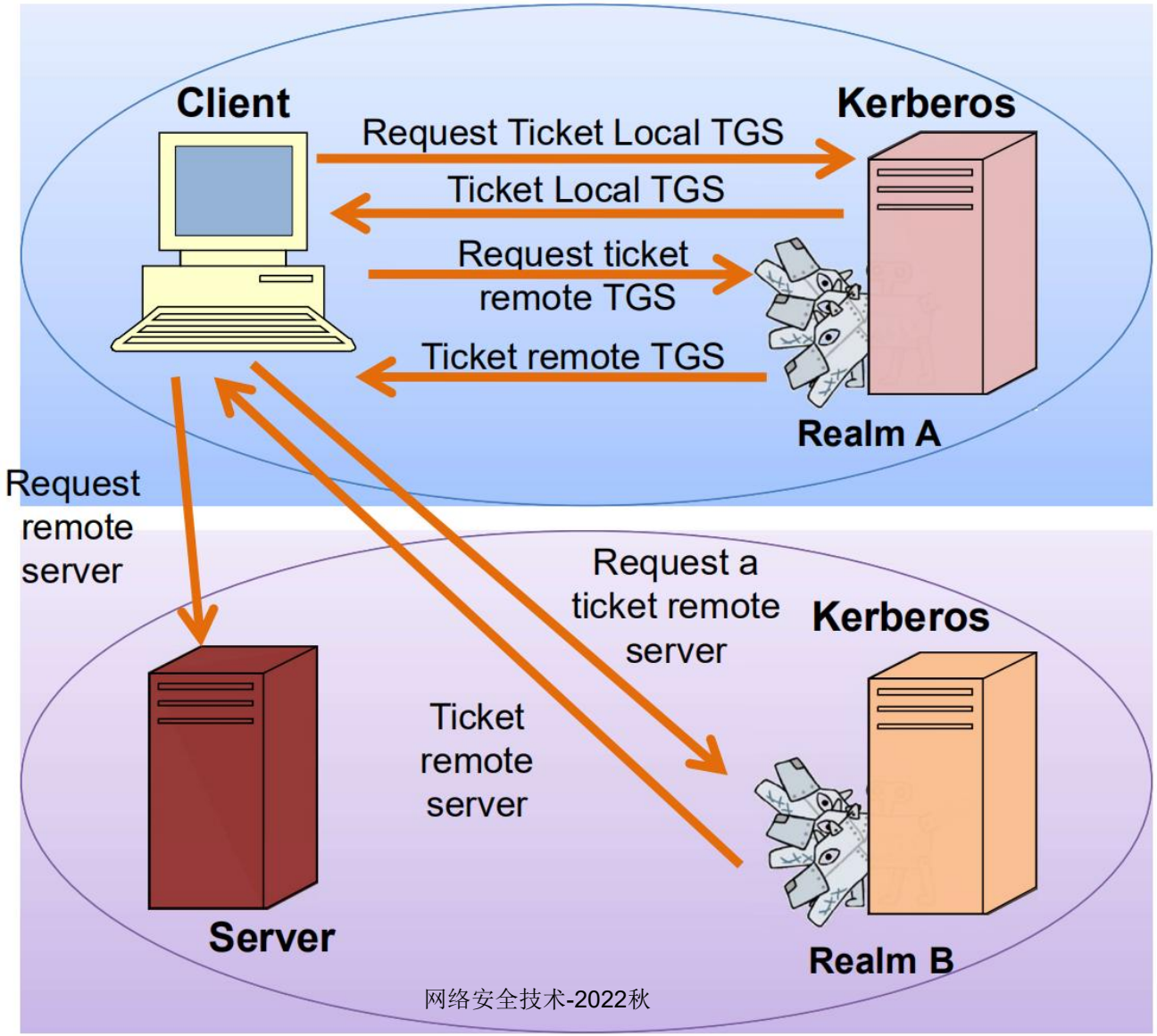
- 认证域
  - 一个Kerberos服务器
    - 用户在kerberos服务器注册，服务器保存全部用户的ID和password
    - 应用服务器在kerberos服务器注册，与每个应用服务器有一个共享密钥
  - 多个客户端Clients
  - 多个应用服务器(Application Servers)

Kerberos域

- 域间认证
  - 每个域中的Kerberos服务器与另一个域中的服务器共享一个密钥。
  - 两个Kerberos服务器相互注册

Kerberos域

- 域间认证



[LoQM]

Kerberos(V5)

- 九十年代中期提出并发展起来
- RFC1510
- 版本5改进了版本4在环境方面和技术方面的局限性
- 环境方面的局限性
  - 加密技术依赖性
  - 网络依赖性
  - 消息字节排序
  - 票据有效期
  - 认证转发
  - 不支持域间认证
- 技术方面的局限性
  - 双重加密
  - PCBC加密
  - 会话密钥
  - 口令攻击

Kerberos(V5) (略)

(1) C → AS Options ‖ ID_c ‖ $Realm_c$ ‖ ID_{tgs} ‖ $Times$ ‖ $Nonce_1$

(2) AS → C $Realm_c$ ‖ ID_C ‖ $Ticket_{tgs}$ ‖ $E(K_c, [K_{c,tgs} \| Times \| Nonce_1 \| Realm_{tgs} \| ID_{tgs}])$

$Ticket_{tgs} = E(K_{tgs}, [Flags \| K_{c,tgs} \| Realm_c \| ID_C \| AD_C \| Times])$

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

(3) C → TGS Options ‖ ID_v ‖ $Times$ ‖ ‖ $Nonce_2$ ‖ $Ticket_{tgs}$ ‖ $Authenticator_c$

(4) TGS → C $Realm_c$ ‖ ID_C ‖ $Ticket_v$ ‖ $E(K_{c,tgs}, [K_{c,v} \| Times \| Nonce_2 \| Realm_v \| ID_v])$

$Ticket_{tgs} = E(K_{tgs}, [Flags \| K_{c,tgs} \| Realm_c \| ID_C \| AD_C \| Times])$

$Ticket_v = E(K_v, [Flags \| K_{c,v} \| Realm_c \| ID_C \| AD_C \| Times])$

$Authenticator_c = E(K_{c,tgs}, [ID_C \| Realm_c \| TS_1])$

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

(5) C → V Options ‖ $Ticket_v$ ‖ $Authenticator_c$

(6) V → C $E_{K_{c,v}}$ [TS_2 ‖ Subkey ‖ Seq#]

$Ticket_v = E(K_v, [Flags \| K_{c,v} \| Realm_c \| ID_C \| AD_C \| Times])$

$Authenticator_c = E(K_{c,v}, [ID_C \| Realm_c \| TS_2 \| Subkey \| Seq\#])$

(c) Client/Server Authentication Exchange to obtain service

总结

- Kerberos
  - 开放分布式环境下的一种认证服务
  - 采用对称加密技术
  - 基于可信第三方提供的认证服务为客户端和服务器之间建立可信通信

练习题

- 基于加密技术的身份认证有多种方法，请说明如下几种身份认证方法的实现过程：
  - 基于对称密钥加密的身份认证
  - 基于数字签名的身份认证
  - 基于公钥加密的身份认证

版权声明

本PPT部分图片直接使用如下材料中的图片：

[LoQM] Lecture of Queen Mary University of London, EBU7140 Security and Authentication, week2, Oct 2020