

身份管理与认证

Identity management (IDM)

Authentication, Authorization, Accounting

身份管理

Single sign on (SSO)

认证协议和系统: kerberos, Radius, Diameter, Oauth

访问控制

LOGO

www.themegallery.com



1.概念： AAA



◉ Authentication, authorization, accounting



◉ 业务系统中最基本的安全服务

- 身份管理和认证是基础：证实客户的真实身份与其所声称的身份是否相符的过程。

the process of verifying a claimed identity

◉ 例：

辅导员
证明
认证



院里
审核
授权

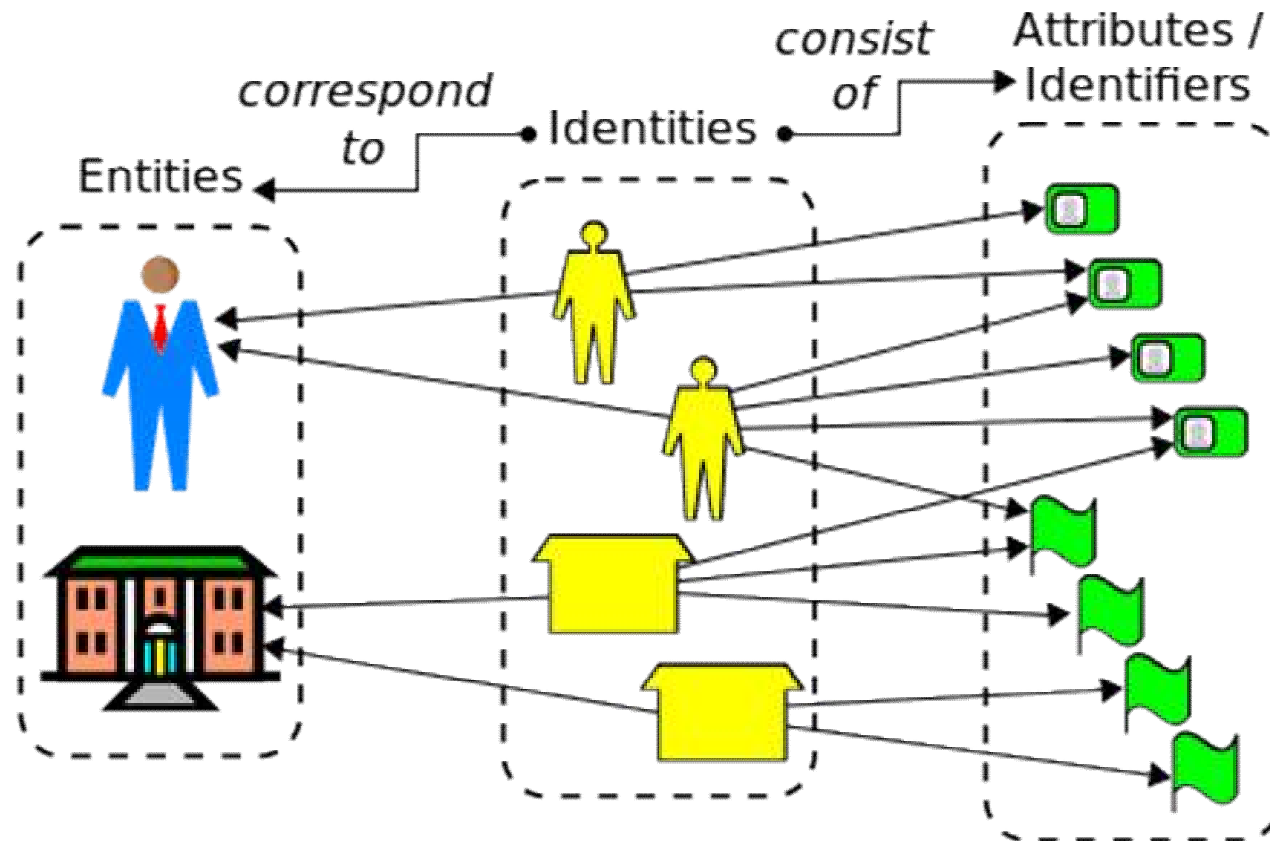


盖章 +
登记
行权
审计



IDM概念

或IAM (identity and access management)
或AIM (Access and Identity Management)



A general model of identity
Entity—identity--attributes



问题



● 网络环境中的身份管理和认证



- 举几个你熟悉的业务系统身份管理和认证实例
- 当前的使用现状你还满意吗？觉得有什么需改进的地方
- 总结
 - 需完成的功能
 - 可能面对哪些安全威胁



● 服务商的想法 vs 用户的想法

- 登录用户参与感更强、粘性更强、更可信
- 登录以后我可以记住用户数据，为其提供个性化体验，用户数据还有许多其它用处。。。
- 登录用户信息的真实性，实名制
- 注册要填表格，很多隐私信息
- 还得去收验证
- 一大堆的账号密码，记不住，不安全

从单系统的帐号管理，认证，到**IDM**
更广义的**ID**： 人，物

IDM标准发展脉络

早期IT商业目的

1999年 微软推出passport项目，与ebay合作支持电子钱包与支付



★ 2001年 Sun公司联合著名企业成立自由联盟LA，目标是对抗微软单点登录系统

★ 2002年5月 LA公布SAML1.0标准
Liberty Alliance

2005年3月 LA公布SAML2.0标准，并通过互操作测试验证

OASIS Security Assertion Markup Language (SAML)

★ 2005年5月 微软推出info card windows组件，简化繁琐的数字身份登录过程。同时支持个人支付确认功能

2005年 开源项目OpenId诞生

CT、政府逐步介入

2009年9月 ETSI成立INS工作组

2009年5月 Kantara Initiative成立

★ Merger of the **Liberty Alliance** and others

2009年9月 3GPP发布GAA 与OpenID互通 R9规范

2007年9月 3GPP发布GAA 与LA ID-WSF互通 R7规范

2007年6月 3GPP发布GAA R6规范

2007年2月 微软推出CardSpace项目，同年宣布支持OpenID

2007年 欧盟启动FP7，eIdM2010一揽子研究项目

2006年12月 ITU-T成立IdM焦点组



- **2010年4月，ITU-T 17**设立“基于全球**IP**网络的分布式名字解析系统体系架构”联络工作组（基于俄罗斯提案）。
- 2010年3月，**Open Identity Exchange OIX**成立。受OIDF和ICF共同赞助成立，OIX成为美政府可信赖架构的供应商，可以提供认证符合美国联邦标准的线上IdP

➤**IDM与未来互联网：2009年9月，ETSI** 成立“基于网络与业务的身份和访问控制”行业规范工作组，简称**INS**；

➤**IDM与云计算：2010年1月，OASIS** 成立**Identity in the Clouds** 技术委员会。（参考 **OASIS Identity in the Clouds usecase**）

➤**IDM与物联网：2010年4月，ITU-T SG 17**设立**X.discovery**研究项目（基于美国提案）；



国内外现状: 欧盟IDM的研究与部署

	欧盟FP5/FP6/FP7	EU成员eID系统调研结果
eGov	eIDM2010（法律支撑） STORK	<ul style="list-style-type: none">• eID认证系统调研结果<ul style="list-style-type: none">– 75% 的国家将PKI作为密钥认证的战略部署– 27个国家使用登录+密码系统<ul style="list-style-type: none">• 17 个国家使用简单用户名+密码登录• 8个国家使用挑战/响应系统（challenge/response）• 2个国家使用密码计算器• 技术与基础设施调研结果<ul style="list-style-type: none">– 28个国家使用或计划使用基于证书的身份。– 22个国家在eGov服务中已经实现了一定级别保证的基于证书的认证。– 23个国家在行业标准方面没有统一的考虑
隐私保护	TAS3, POCOS, PRIME, PRIMELIFE,	
隐私监测	PRISM	
生物识别与测定	TURBINE, ACTIBIO, MOBIO	
未来互联网	FIDIS www.fidis.net/home/	
电信运营商	SWIFT	



国内外现状



● 美国：类似**DNS**，建立全球身份命名标准



- www.handle.net
- 例如：目前每篇国际期刊论文都有永久标识
- 论文标识：doi:10.1155/2010/101523 （该出版社已经用上handle system）。因此用户只需要敲入
<http://dx.doi.org/10.1155/2010/101523> 就可以找到那篇文章。

● 美国在身份管理技术领域居领先地位

- 在国家层面建立身份管理的监管体系（标准+解决方案）
- 建立身份命名机制（标准+解决方案+运营）



商业应用



Microsoft account(previously known as Microsoft Passport, .NET Passport, Microsoft Passport Network, and Windows Live ID)
Microsoft account support the [OpenID](#) framework, with Windows Live ID becoming an OpenID provider

IdM参考模型



用户/终端实体

SP

依赖方

IDP

身份提供者

人

法人（机构、组织、代理机构等）

对象：

- 物理对象：终端设备、网络设备、SIM卡或者智能卡。
- 虚拟对象：网络、位置、地理空间等。

发送身份证明

业务或资源
提供商

e. g. , 中国
电信、银行、
电子商务网
站等。

业务提供商响
应身份证明

向身份提供者
请求身份资源

?

身份提供者相
应资源请求

政府或可信第
三方

信任凭证服务
标识服务
属性服务
模式/信用服务
发现服务
身份桥接服务
审计/策略执行
联盟

三方身份管理模型对应的实体



IDM model



◦ SP Service provider, Identity provider IDP



- Isolated model: SP as SP + IDP
- Centralized model: all SPs use a unique IDP
- Federation model:

Federation: the set of agreements, standards and technologies that enable a **group of SPs** to recognize user identities from other SPs in a **federated trust domain** .

Federation IDM: A system that relies on Federated identity to authenticate a user without knowing his or her password. (federate user access and allow users to login based on authenticating against one of the system participating in the federation)



2. SSO



● **Single sign-on** is a user/session authentication process that permits a user to **enter one name and password** in order to **access multiple applications**.





Types of SSO



- Password Synchronization (other identity certifications)
- Legacy SSO (Employee/Enterprise SSO) (eSSO , aka – Enterprise or Employee SSO)
- Web Access Management (Web SSO)
- Cross Domain (realm) SSO
- Federated SSO



Password Synchronization

- A process that coordinates passwords across multiple computers and devices and/or applications
- Each computer, device, application still authenticates but behind the scene



eSSO



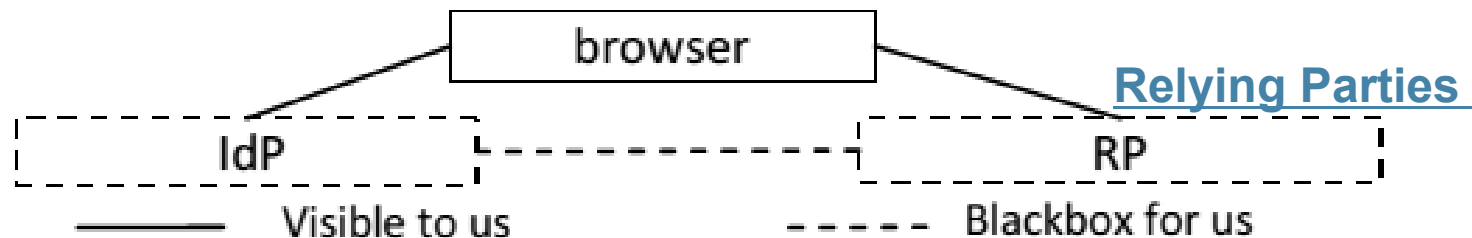
- the identity provider and service provider are in the same organization
- After primary authentication, it intercepts further login prompts and fills them for you
- Kerberos
- Yale CAS (Central Authentication Service)
<https://www.apereo.org/cas>, also support SAML based delegated authentication, allows an application to access selected additional resources on an end user's behalf without exposing a password



Basic Web SSO (WAM)



- Browser based application, Cookie
- Single sign-on to applications deployed on a single web server (domain)
- OpenID





OpenID (OID)



an open standard and decentralized protocol by the non-profit OpenID Foundation that allows users to be authenticated by certain co-operating sites (known as Relying Parties or RP) using a third party service. This eliminates the need for webmasters to provide their own ad hoc systems and allowing users to consolidate their digital identities.^[1]

- AOL, Blogger, Flickr, France Telecom, Google, Hyves, LiveJournal, Microsoft (provider name Microsoft account), Mixi, Myspace, Novell, Orange, Sears, Sun, Telecom Italia, Universal Music Group, VeriSign, WordPress, and Yahoo!. Other providers are BBC,^[3] IBM,^[4] PayPal,^[5] Steam,^[6] along with GitHub, Identi.ca, Last.fm, Linkedin, and Twitter.^[7]
- Related : Facebook Connect



Cross Domain SSO

- Multiple realms that manage user credentials.
- A user authenticated in one realm gets signed-on to an application using another realm **typically with in the same enterprise**



Federated SSO



- Extend SSO across enterprises
- Liberty Alliance, OASIS, Shibboleth, IBM/Microsoft

-Case study

-Federal Aviation Administration Requirements:

- Provide SSO to ~500,000 users
- Across 5000 airports world-wide
- >100 web and client server applications
- Multiple Directories, Departments
- Web services authentication



SAML



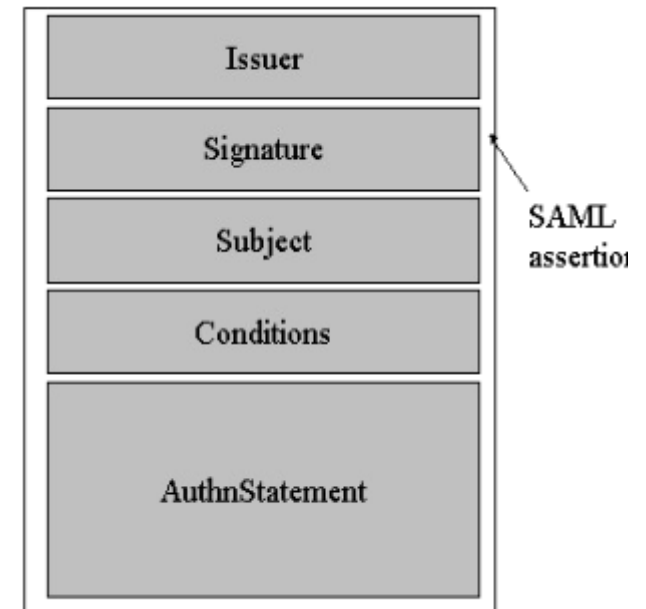
- *Security Assertion Markup Language*



- an XML-based framework for communicating user authentication, entitlement, and attribute information.

- Internet2 Shibboleth(<http://shibboleth.net/>) project, open source federated identity solutions

Shibboleth3(identity platform) support REST- and JSON based assertion in addition to XML-based SAML.





例： Liberty Alliance project,

信任方

A B C

身份提供者

用户凭证

X Y Z

1) 请求
2) 认证要求
5) 断言
6) 结论

3) 请求认证
(如, "login")

4) 可信的断言
(如, SAML)





SSO Summary



- ◉ Reduces cost



- ◉ Enhances security

- ◉ Supports compliance

- Financial Service (FFIEC directive)
- Healthcare (HIPPA)

- ◉ But....there are risks.

- Malicious user gets hold of unattended desktop
- Malicious processes/services sign on as you to services that they are not supposed to.
- http://tetraph.com/covert_redirect/



MAIN FEATURES IN EXISTING IDENTITY MANAGEMENT INITIATIVES

Initiatives	Privacy	Security	Mobility	Interoperability	Anonymity	Federation	User-centric
PRIME	•	•	•		•		•
PRIMELife	•	•			•		•
DAIDALOS		•	•		•		•
SWIFT	•		•	•	•		•
Liberty Alliance	•	•			•	•	
Kantara	•	•	•	•	•	•	
FIDIS	•	•	•				•
SAML				•	•	•	
Higgins	•	•		•	•		•
OpenID							•
Shibboleth	•					•	
STORK	•	•		•			•
PICOS	•	•	•	•			•

Jenny Torres, Michele Nogueira, and Guy Pujolle, A Survey on Identity Management for the Future Network, IEEE COMMUNICATIONS SURVEYS & TUTORIALS

Security: Signing Me onto Your Accounts through Facebook and Google: a Traffic-Guided Security Study of Commercially Deployed Single-Sign-On Web Services <http://research.microsoft.com/apps/pubs/default.aspx?id=160659>



3. 身份认证基本原理



➤ 凭证:



(1) 知道什么 (secret knowledge), **口令**, PIN码, 秘密或私钥等

(2) 拥有什么 (Token): 磁卡、令牌、密码智能卡, ID卡、门钥匙, **数字证书, 手机**, 生物识别特征: 指纹、声音、虹膜、DNA 及签名等

双因子 (two-factor) 认证机制: 你有什么 + 你知道什么
如: 用户必须同时提供卡片与卡片相应的PIN码

It consists of several steps:

- **Obtaining** the authentication information from an entity
- **Analyzing** the data
- **Determining** if the authentication information is associated with that entity



Authentication System model



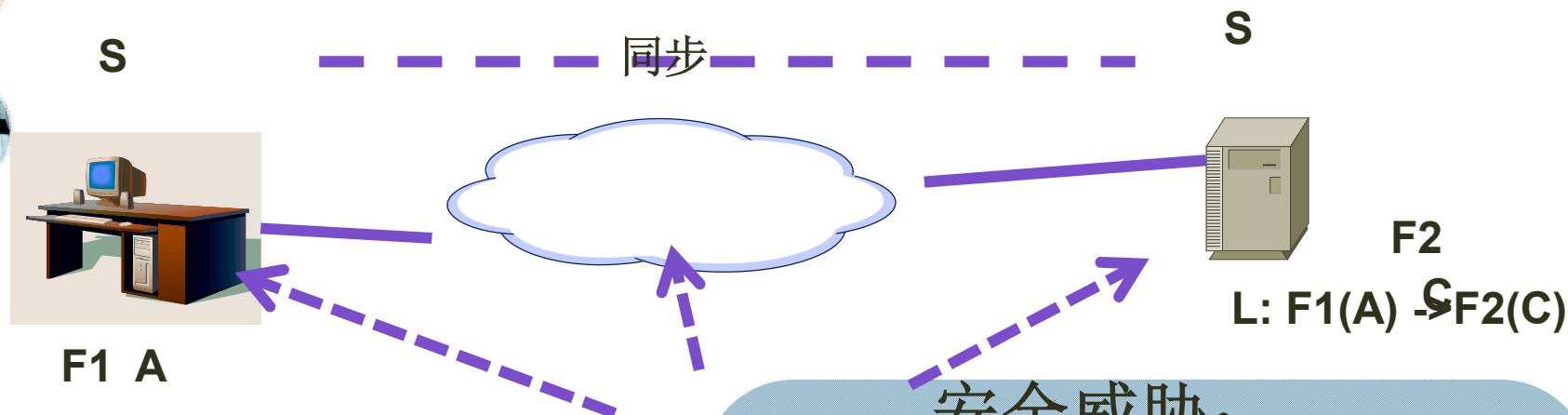
◉ (A, C, F, L, S)



- A information that proves identity
- C information used to validate authentication information (stored on authentication server)
- F complementation function $f : A \rightarrow C$
- L functions that prove identity
- S functions enabling entity to create, alter information in A or C



网络环境中的用户认证



要求

机密性，完整性，不可否认，
抗重发

用户认证信息如何安全传送
如何与网络结构和协议结合
如何满足业务要求：如移动

安全威胁：

窃听，重发，中间人
假冒，凭证：丢失、偷窃、复制
Guess, Brute force, Social engineering
Phishing
Manipulated input devices:
keyloggers



□ 令 Password system



○ **A = password** : information associated with an entity that confirms its identity.



○ **C = ?** How can passwords be protected?

- Plaintext (sniffing, steal)
- Encrypted (steal, replay)
- One-way hash (dictionary attack, replay)

○ **F: Algorithms**

- Examples: challenge-response, one-time password

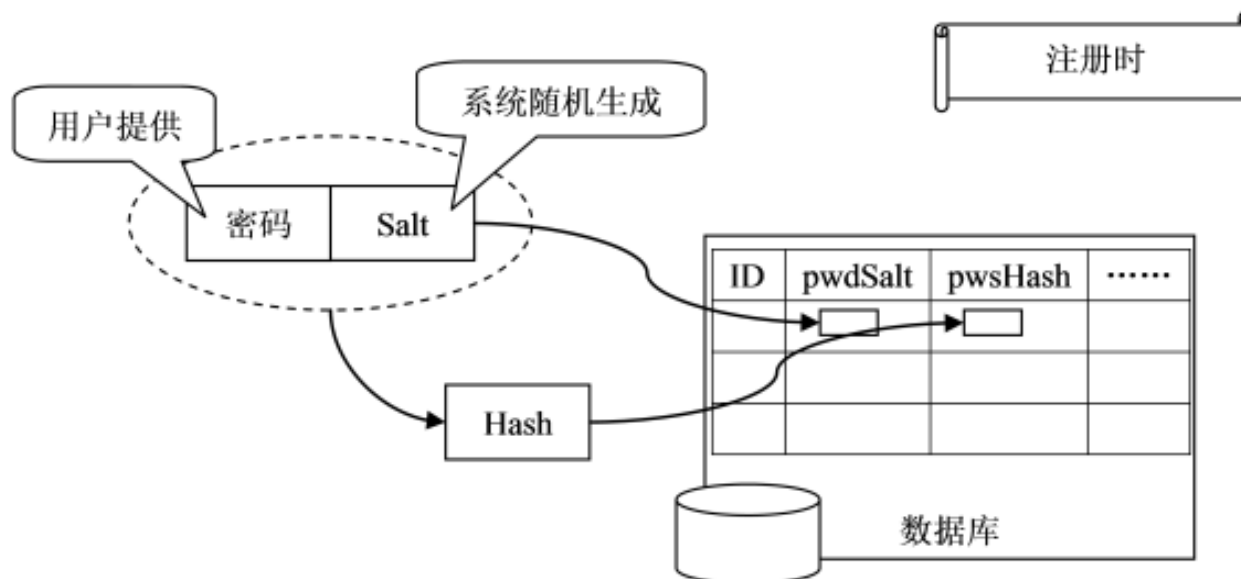
○ **L : equal**

字典式攻击和salt



MALLORY在业余时间编制了1000000个最常用的口令表，他用单向函数对所有1000000个口令进行运算，并将结果存储起来。现在MALLORY偷出口令文件，将它与自己的可能的口令文件进行比较，再观察哪个能匹配。这就是字典式攻击。

Salt是使这种攻击更困难的一种。**Salt**是一随机字符串，它与口令连接在一起，再用单向函数对其运算，然后将**salt**值和单向函数运算的结果存入主机数据库中。如果可能的**salt**值的数目足够大的话，它实际上就消除了对常用口令采用的字典式攻击。





一次性口令（One Time Password）



每次登录过程中传送的口令都不相同，以提高登录过程安全性，并可对付重放攻击。



一次性口令的特点：

- ◆ 概念简单，易于使用
 - ◆ 基于一个被记忆的密码，不需要任何附加的硬件
- ◆ 算法安全
 - ◆ 不需要存储诸如密钥、口令等敏感信息

需要： 种子（**Seed**）、迭代（**Iteration**）及同步



S/Key One-time password



- h one-way hash function (MD5 or SHA-1, for example)

- User chooses initial seed k

- The key generator calculates:

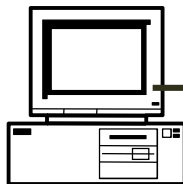
$$h(k) = k_1, h(k_1) = k_2, \dots, h(k_{n-1}) = k_n$$

- Passwords are in reverse order:

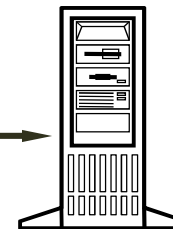
$$p_1 = k_n, p_2 = k_{n-1}, \dots, p_{n-1} = k_2, p_n = k_1$$

$$h(P_A), h^2(P_A), h^3(P_A) \dots, h^n(P_A)$$

User A



$$h^{n-i+1}(P_A), i$$



Server B



一次性口令产生和验证过程

- ① 用户输入登录名和相关身份信息ID。
- ② 如果系统接受用户的访问，则给用户传送一次性口令建立所使用的单向函数 f 及一次性密码 k ，这种传送通常采用加密方式。
- ③ 用户选择“种子”密钥 x ，并计算第一次访问系统的口令 $z=f^n(x)$ 。向第一次正式访问系统所传送的数据为 (k, z) 。
- ④ 系统核对 k ，若正确，则将 $(ID, f^n(x))$ 保存。
- ⑤ 当用户第二次访问系统时，将 $(ID, f^{n-1}(x))$ 送系统。系统计算 $f(f^{n-1}(x))$ ，将其与存储的数据对照，如果一致，则接受用户的访问，并将 $(ID, f^{n-1}(x))$ 保存。
- ⑥ 当用户第三次访问系统时，将 $(ID, f^{n-2}(x))$ 送系统。系统计算 $f(f^{n-2}(x))$ ，将其与存储的数据对照，如果一致，则接受用户的访问，并保存新计算的数据。
- ⑦ 当用户每一次想要登录时，函数相乘的次数只需 1。



基于共享密钥的认证

● (A,C,F,L,S) A= key

● 基于共享密钥的常见认证协议有：

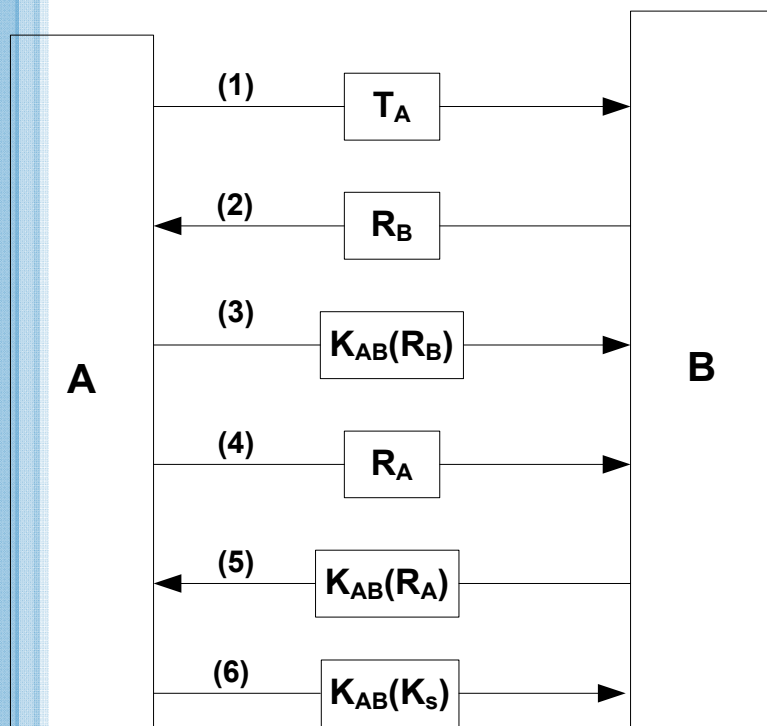
- 无可信第三方的质询—回应协议（使用HASH，如CHAP）
- 有可信第三方，使用密钥分发中心的认证协议
- Needham-Schroeder认证协议（多路质询—回应协议）(P41) (如 kerberos)
- Otway-Rees认证协议



质询—回应(challenge-response)协议



还记得无中心，有共享密钥的
密钥交互流程吗？



抗重发
验证了双方都知道秘密

- (1) A向B发送一个消息 T_A ，表示想和B通话。
- (2) B无法判断这个消息是来自A还是其他人，因此B回应一个质询 R_B 。 R_B 是一个随机数。
- (3) A用与B共享的密钥 K_{AB} 加密 R_B ，得到密文 $K_{AB}(R_B)$ ，再发送给B；B收到密文 $K_{AB}(R_B)$ ，用自己同样拥有的 K_{AB} 解密，对比结果，如果相同就确认了A的身份。此时B已完成了对A的单向认证。
- (4) A同样需要确定B的身份，于是发送一个质询 R_A 给B。 R_A 也是一个随机数。
- (5) B用与A共享的密钥 K_{AB} 加密 R_A ，得到密文 $K_{AB}(R_A)$ ，再发送给A；A收到密文 $K_{AB}(R_A)$ ，用自己同样拥有的 K_{AB} 解密，对比结果，如果相同就确认了B的身份，完成了双向认证。
- (6) A确认B的身份之后，选取一个会话密钥 K_S ，并且用 K_{AB} 加密之后发送给B。



Needham-Schroeder认证协议

- (1) 产生一个大的随机数 R_A 作为临时值，向KDC发送消息 $M(R_A, A, B)$;
- (2) KDC产生一个会话密钥 K_S ，再用B的密钥 K_B 加密 (A, K_S) ，作为下轮A发给B的Ticket $K_B(A, K_S)$ ，然后再用A的密钥 K_A 加密 $(R_A, B, K_S, K_B(A, K_S))$ ，发送给A;
- (3) A用自己的密钥 K_A 解密密文，获取 K_S 和 $K_B(A, K_S)$ ；然后产生一个新的随机数 R_{A2} ，用KDC发过来的 K_S 加密 R_{A2} ，将票据 $K_B(A, K_S)$ 和 $K_S(R_{A2})$ 发给B;
- (4) B接收到消息用自己的密钥 K_B 解密密文 $K_B(A, K_S)$ 得到 K_S ，再用 K_S 解密密文 $K_S(R_{A2})$ 得到 R_{A2} ；然后用 K_S 加密 $(R_{A2}-1)$ 并产生随机数 R_B ，再发回给A。
- (5) A收到消息后确认了B的身份，再向B发送 $K_S(R_B-1)$ 。B收到消息后也可以确认A的身份，也确认了双方都有 K_S 。

共享秘密: K_A 和 K_B

(1) $A \rightarrow KDC : A || B ||$

R_A

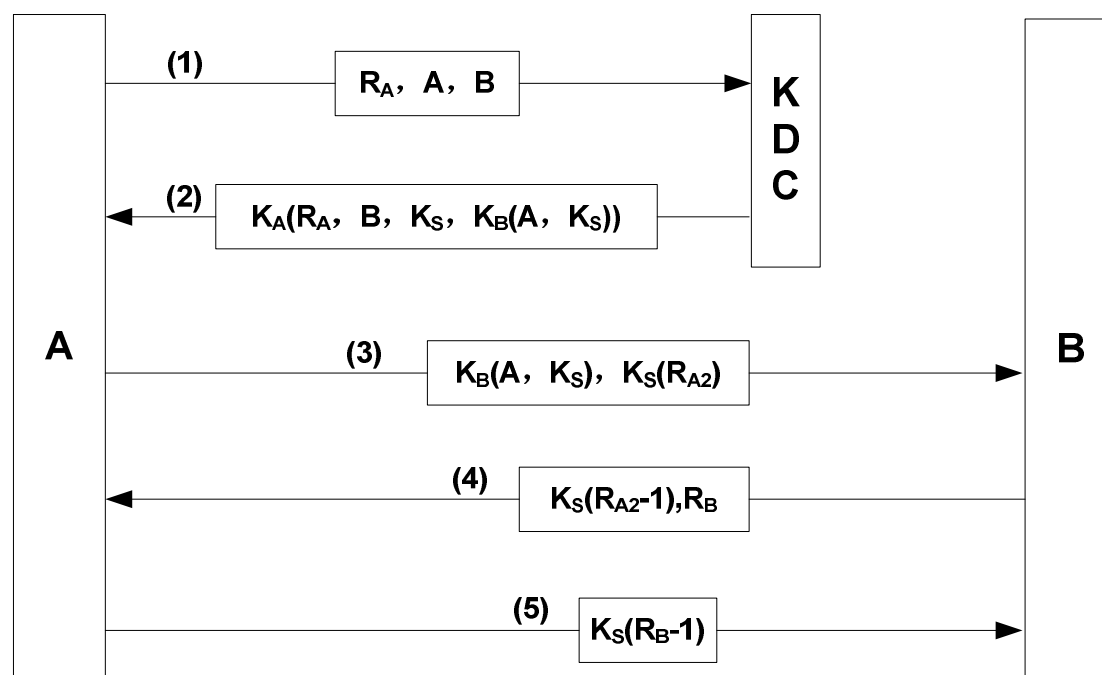
(2) $KDC \rightarrow A : EK_A$

$[R_A || B || K_S || EK_B [K_S || A]]$

(3) $A \rightarrow B : EK_B [K_S || A]$

(4) $B \rightarrow A : EK_S [R_B]$

(5) $A \rightarrow B : EK_S [R_B-1]$





Needham-Schroeder认证协议

问题:

- 记住 K_S , 重发 $K_B (A, K_S)$ 和 $K_S (R_{A2})$, 可假冒A
- 解决: 加时间戳 (kerberos) (课外练习)
- A如何对B进行认证 (课外练习)
- K_A 泄密后?



数字时间戳 (Digital Time-Stamp)

- 数字时间戳服务 (DTS) 是网上安全服务项目，由专门的机构提供。
- 时间戳是一个经过加密后形成的凭证文档，包括：需加时间戳的文件的摘要、DTS收到文件的日期和时间、DTS的数字签名
- 时间戳产生过程：
- 用户将需加时间戳的文件用HASH编码加密形成摘要，并将其发送到DTS；DTS在加入了收到日期和时间信息后再对该文件加密和数字签名，然后返回用户

Otway-Rees认证协议



(1) **A**产生一消息，包括用和**KDC**共享的密钥**Ka**加密的一个索引号**R**、**A**的名字、**B**的名字和一随机数**Ra**。



(2) **B**用**A**消息中的加密部分构造一条新消息。包括用和**KDC**共享的密钥**Kb**加密的一个索引号**R**、**A**的名字、**B**的名字和一新随机数**Rb**。

(3) **KDC**检查明文**R**和两个加密部分中的索引号**R**是否相同，如果相同，就认为从**B**来的消息是有效的（认证了**A, B**）。**KDC**产生一个会话密钥**Ks**用**Kb**和**Ka**分别加密后传送给**B**，每条消息都包含**KDC**接收到的随机数。

(4) **B**把用**A**的密钥加密的消息连同索引号**R**一起传给**A**。

KDC对A, B认证

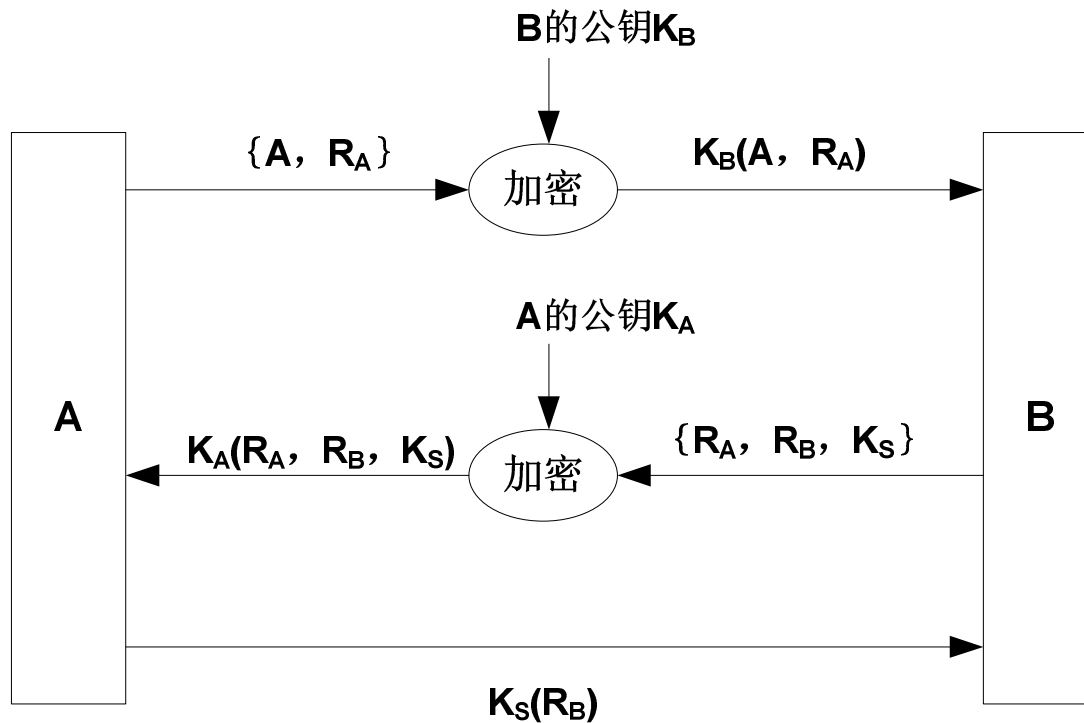
(1) $A \rightarrow B: A \parallel B \parallel R \parallel EK_a [A \parallel B \parallel R \parallel Ra]$

(2) $B \rightarrow KDC: R \parallel A \parallel B \parallel EK_a [A \parallel B \parallel R \parallel Ra] \parallel EK_b [A \parallel B \parallel R \parallel Rb]$

(3) $KDC \rightarrow B: R \parallel EK_b [Rb \parallel Ks] \parallel EK_a [Ra \parallel Ks]$

(4) $B \rightarrow A: R \parallel EK_a [Ra \parallel Ks]$

基于公钥的认证（无可信第三方的）



(1) A首先生成质询信息 R_A ， R_A 是一个随机数；接着A用B的公钥 K_B 加密会话信息 $\{A, R_A\}$ ，然后发给B。

(2) B用自己的私钥解出 $\{A, R_A\}$ ，再生成质询信息 R_B 和会话密钥 K_S ，接着B用A的公钥 K_A 加密会话信息 $\{R_A, R_B, K_S\}$ ，然后发给A。

(3) A用自己的私钥解出 $\{R_A, R_B, K_S\}$ ，核对 R_A 无误后，用 K_S 加密 R_B ，然后发给B。B收到后用 K_S 解出 R_B ，核对无误后完成双向认证。

公钥私钥对的可信问题



数字证书X.509

还记得证书的格式吗？
如何用证书进行身份认证呢？

原理：

$$A \rightarrow B : M, E_{SK_A}(H(M)), CertA$$

说明： B 使用证书，通过验证 A 对 M 的签名，从而验证 A 的身份。

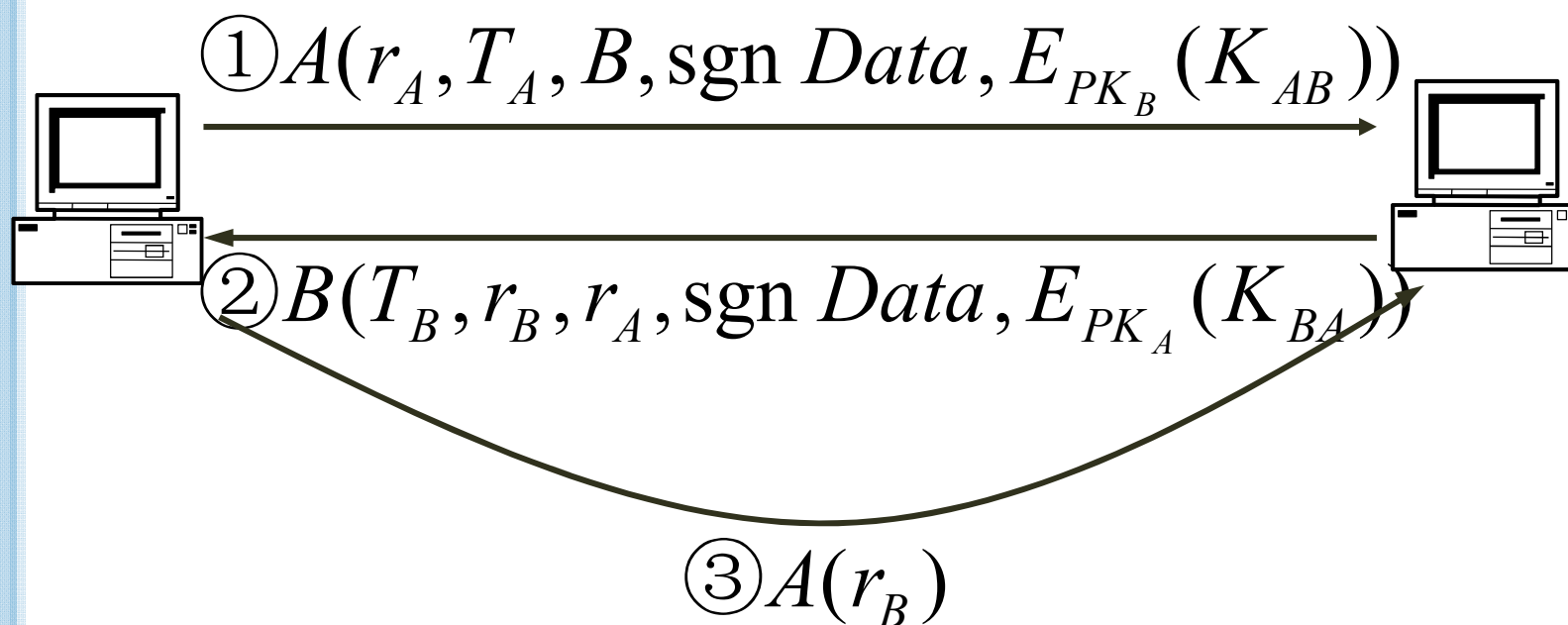


数字证书X.509



只有 (1) 为单向认证
只有 (1) (2) 为双向认证
三向认证
由PKI系统验证证书的有效性

常同时传送证书**CERT**



(3) 利用随机数而不是时间戳实现抗重发



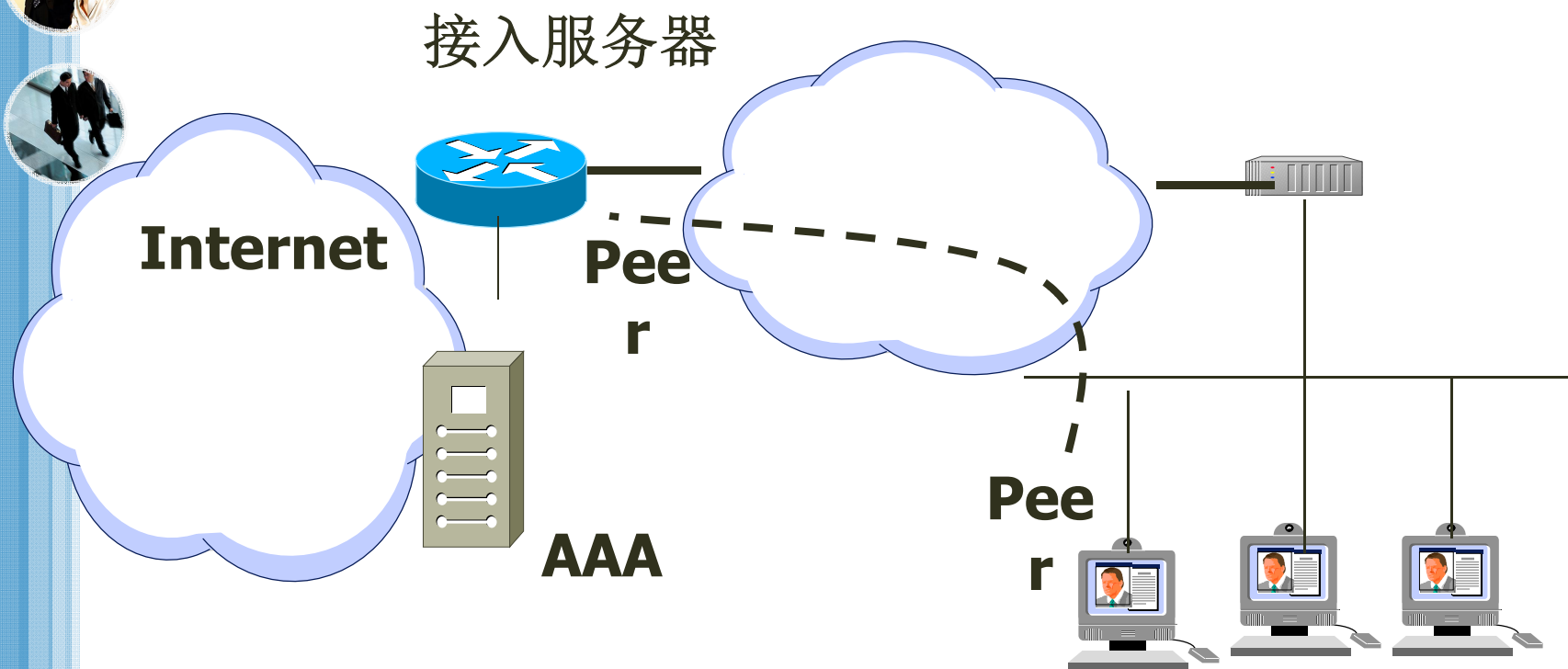
实际系统



- ◉ 宽带接入中 **PAP, CHAP, EAP**, 端口认证
- ◉ 电信网中的**RADIUS, DIAMETER**
- ◉ **Kerberos**
- ◉ **Web 2.0** 中的**Oauth**
- ◉ **Skype**
- ◉ 更多: **IPSEC, TLS**等协议中的认证及密钥交互



实例：拨号接入用户的认证：PPP



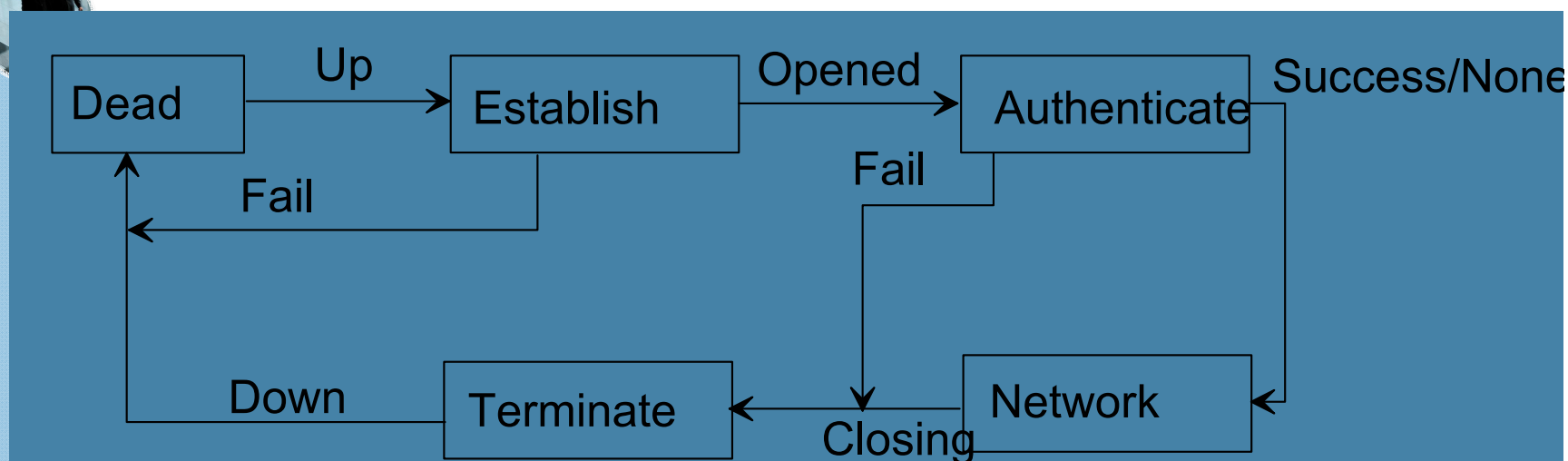
拨号接入的结构、过程



PPP



PPP状态转移图



建立在**PPP**上的口令验证协议（**PAP**、**SPAP**、**CHAP**、**MPPE**和**EAP**）

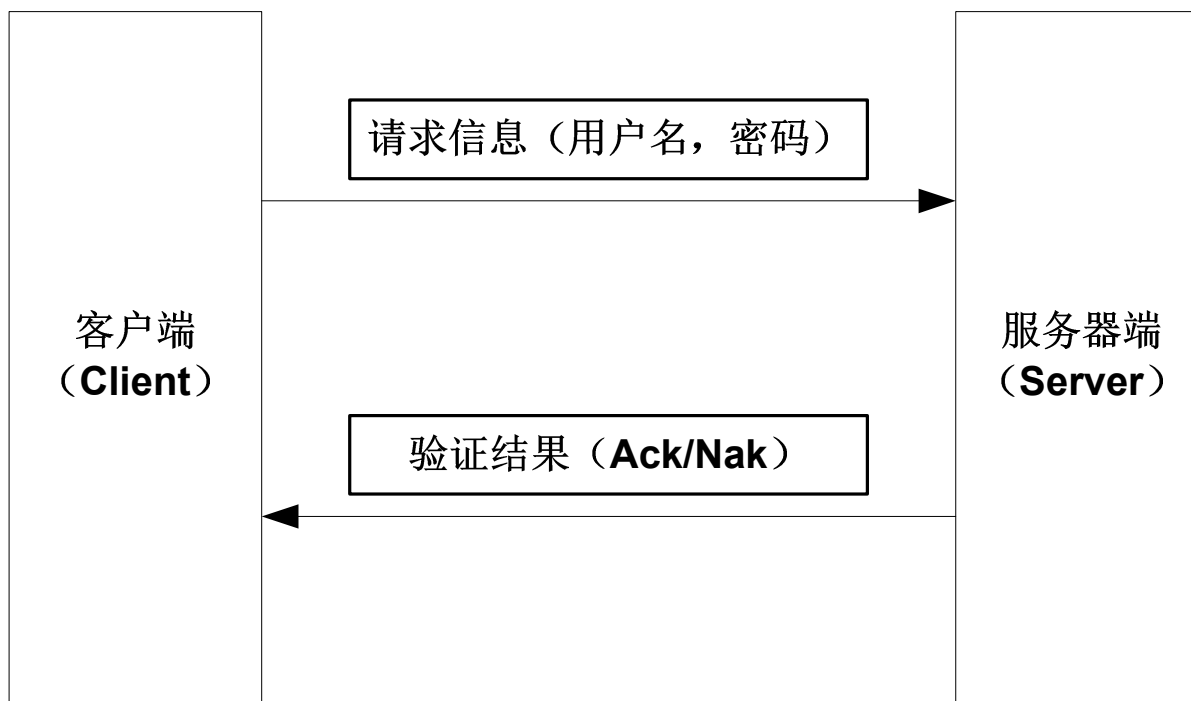
1. 口令验证协议（**PAP Password Authentication Protocol**）
 2. 挑战-握手验证协议（**CHAP**）
 3. 微软挑战-握手验证协议（**MS-CHAP**）



PAP身份认证的方式



- PAP不是一个强壮的认证协议，它利用双向握手确认呼叫方的合法性，但是口令以明文的形式在链路转送，并且它不能防止重放或重复尝试攻击。PAP允许在远端节点控制身份认证的频率和时间。
- PAP协议的身份认证是两次握手验证过程。





CHAP



- **Challenge Handshake Authentication Protocol**
- 对PAP进行了改进，不再直接通过链路发送明文口令
- CHAP采用是三次握手验证，服务器端存有客户的明文口令，所以服务器可以重复客户端进行的运算，将结果与用户返回的口令进行对照。
- CHAP为每一次验证任意生成一个挑战字符串来防止受到重发攻击（replay attack）。在整个连接过程中，CHAP将不定时地向客户端重复发送挑战口令，从而避免第3方冒充远程客户进行攻击。

ID仍明文传，所以Brute force

CAPTCHA

以太网接入：基于端口的认证

IEEE 802.1x (Port based network access control protocol)

EAP客户端

非授权端口 (只允许EAPOL包经过)
授权端口,

EAP设备端

NAS



EAPOL (EAP over LAN)



EAP <-> **RADIUS**

IP网

AAA

RADIUS : RFC 2865~2869



RADIUS

○ Remote Access Dial In User Services) RFC2138 (更新RFC2865) , RFC2866 RADIUS Accounting

- 管理远程用户验证和授权的常用方法，是一种基于UDP协议的轻量级（lightweight）协议。
- 允许用户信息集中管理
- 适用于以PPP为基础的接入, 如...
- 功能弱：授权功能几乎没有，计费功能很差，计费开始_结束(只计时间, 不能计流量)



RADIUS



- RADIUS服务器可以被放置在Internet网络的任何地方为客户NAS提供验证。
- RADIUS服务器可以提供代理服务将验证请求转发到远端的RADIUS服务器。
 - 例如，ISP之间相互合作，通过使用RADIUS代理服务实现漫游用户在全球各地使用本地ISP提供的拨号服务连接Internet和VPN。如果ISP发现用户名不是本地注册用户，就会使用RADIUS代理将接入请求转发给用户的注册网络。这样企业在掌握授权权利的前提下，有效的使用ISP的网络基础设施，使企业的网络费用开支实现最小化。



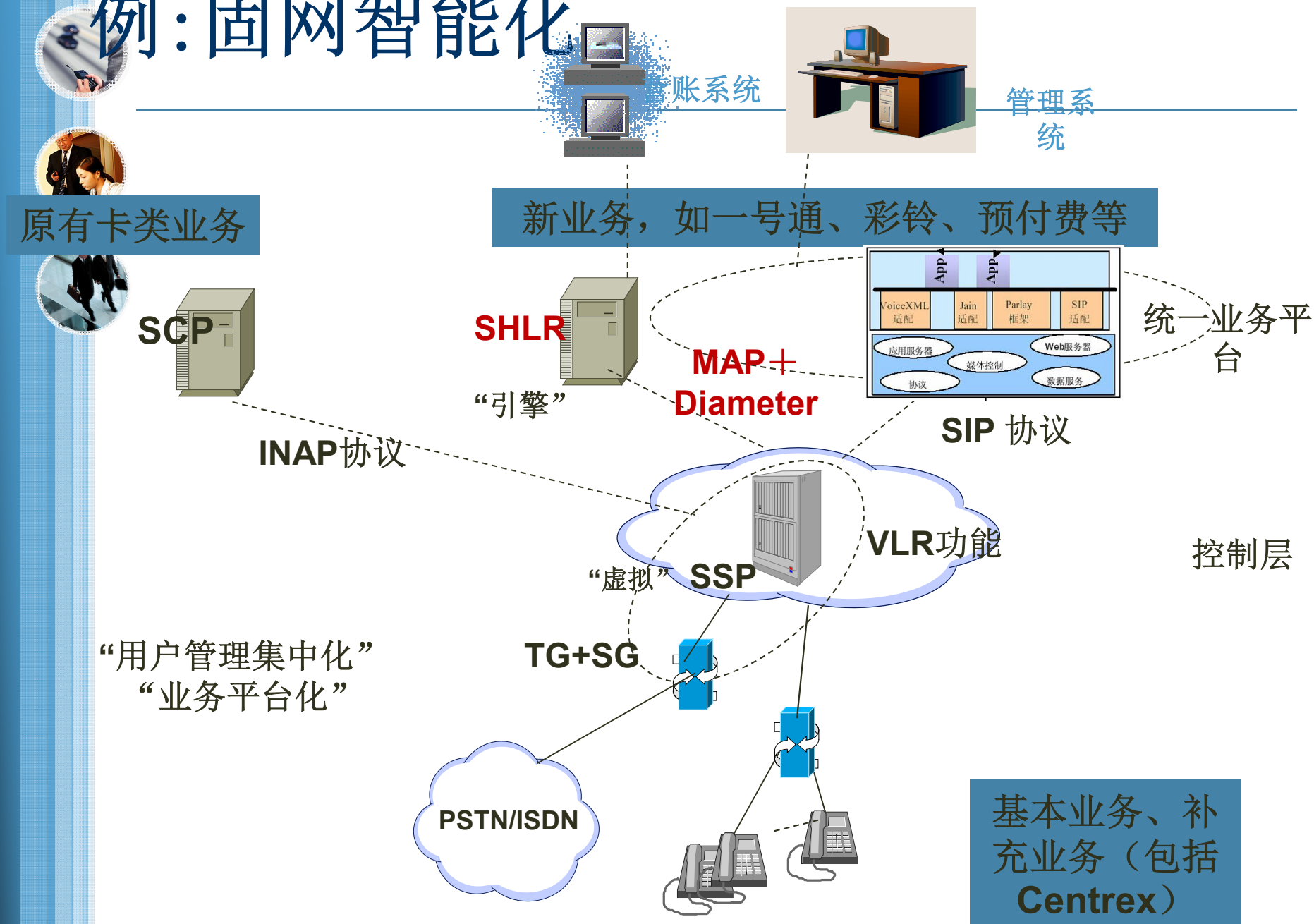
◉ Diameter (1998年,sun提出, IETF AAAWG)

- Diameter Base Protocol RFC3588 sept, 2003

◉ Diameter的一些特性

- failover mechanism
- Transmission-level security : IPSEC, TLS
- 可靠传送: TCP SCTP
- **server-initiated messages**
- 客户机/服务器的能力协商
- Peer discovery and configuration
- Handle accounting and billing
- 更好地适用于各种应用
-

例：固网智能化





Kerberos



- RFC 4120 The Kerberos Network Authentication Service (V5)“
- RFC 4121 The Kerberos Version 5 Generic Security Service Application Program Interface (GSS-API) Mechanism: Version 2.”
- IETF Kerberos working group
- Authentication, Authorization, Accounting
- Stable release 22 Dec 2010 - krb5-1.9
- Website <http://web.mit.edu/kerberos/>
- Kerberos uses port 88 by default.

为网络通信提供可信第三方服务的**面向开放系统**的认证机制



问题:

- 口令次数问题, 每次访问服务器均需输入口令。
- 众多的应用

可能的解决办法:

- 票据重用:
- 引入票据许可服务器TGS (ticket-granting server)

(2) 完整的Kerberos验证协议



Ticket Granting Server

Kerberos 用户的名字, 密钥, 截止信息(记录的有效时间, 通常为几年)等

Kerberos Database

完成principle的认证, 并生成会话密钥
存放一个Kerberos数据库的只读的副本

Authentication Server

KDBM Server

接受客户端的请求
对数据库进行操作

Workstation

用户程序: 登录 Kerberos, 改变 Kerberos 密码, 显示和破坏 Kerberos 标签 (ticket) 等工作。

Kerberos Key Distribution Service



principle



Kerberos 两种证书



- 票据ticket和认证符authenticator。这两种证书都要加密，但加密的密钥不同。
- Ticket用来安全地在AS和TGS之间传递用户的身份，同时保证使用ticket的用户必须是ticket中指定的用户。
Ticket的组成： C/S的标识，client的地址，时间戳，生存时间，会话密钥五部分组成。Ticket一旦生成，在life指定的时间内可以被client多次使用来申请同一个server的服务。
- Authenticator：提供信息与ticket中的信息进行比较，一起保证发出ticket的用户就是ticket中指定的用户。认证符有下列部分组成：client的名字，client的地址，记录当前时间的时间戳。**authenticator**只能在一次服务请求中使用，每当client向server申请服务时，必须重新生成Authenticator。



Kerberos验证标准： 详



常用术语的简写：



- IDc、IDv、IDtgs 分别为C、V、TGS的身份
- ADc: 用户的网络地址
- TSi: 第i个时戳
- Lifetime: 第I个有效期限
- Pc: C上的用户口令
- **Kc**: C和AS的共享密钥
- **Kv**: V和TGS的共享密钥
- **Ktgs**: TGS 和AS的共享密钥
- Kc,tgs: C与TGS的共享密钥
- Kc,v: C与V的共享密钥（会话密钥）

C: 客户机

V: 服务器

TGS: ticket-granting
server

AS: 认证服务器



$$Ticket_{TGS} = E_{K_{TGS}} (K_{C,TGS}, ID_C, AD_C, ID_{TGS}, TS_2, LT_2)$$

$$Au_C = E_{K_{C,TGS}} (ID_C, AD_C, TS_3)$$

Kerberos

客户C

$$\textcircled{1} ID_C, ID_{TGS}, TS_1$$

AS

$$\textcircled{2} E_{K_C} (K_{C,TGS}, ID_{TGS}, TS_2, LT_2, Ticket_{TGS})$$

$$\textcircled{5} Ticket_V, Au_V$$

$$\textcircled{6} E_{K_{C,V}} (TS_5 + 1)$$

$$\textcircled{3} ID_V, Ticket_{TGS}, Au_C$$

应用服务器V

$$\textcircled{4} E_{K_{C,TGS}} (K_{C,V}, ID_V, TS_4, Ticket_V)$$

TGS

$$Au_V = E_{K_{C,V}} (ID_C, AD_C, TS_5)$$

$$Ticket_V = E_{K_V} (K_{C,V}, ID_C, AD_C, ID_V, TS_4, LT_4)$$



工作过程



用户口令PC: 由用户和AS共享, AS将PC保存在数据库中。



1. $C \rightarrow AS : ID_C || ID_{tgs} || TS_1$

2. $AS \rightarrow C :$

$E_{K_c}[K_c, tgs || ID_{tgs} || TS_2 || Lifetime_2 || Ticket_{tgs}]$

$Ticket_{tgs} = E_{K_{tgs}}[K_c, tgs || ID_C || AD_C || ID_{tgs} || TS_2 || Lifetime_2]$

说明:

1. 不输入C的口令, 就不能解开来自AS的信息
2. TS_1 时戳用来防止重放攻击;
3. K_c 由用户口令导出 (用户机器收到AS回应后, 要求用户输入密码, 将密码转化为DES密钥 K_c);
4. K_c, tgs 是C和TGS间的会话密钥。



3. $C \rightarrow TGS : ID_v || Ticket_{tgs} || Authenticator_c$
 $Authenticator_c = E_{K_{c, tgs}}[ID_c || AD_c || TS_3]$

说明：

TGS拥有 K_{tgs} ，可以解密 $Ticket_{tgs}$ ，然后使用从
 $Ticket_{tgs}$ 得到的 $K_{c, tgs}$ 来解密 $Authenticator_c$

将认证符中的数据与票据中的数据比较，以验证票据发
送者就是票据持有者。

4. $TGS \rightarrow C :$

$E_{K_{c, tgs}}[K_{c, v} || ID_v || TS_4 || Ticket_v]$

$Ticket_v =$

$E_{K_v}[K_{c, v} || ID_c || AD_c || ID_v || TS_4 || Lifetime_4]$



5. $C \rightarrow V$: $\text{Ticket}_v || \text{Authenticator}_c$

6. $V \rightarrow C$: $E_{k_{c,v}}[TS_5+1]$

其中: $\text{Ticket}_v =$

$E_{k_v}[K_{c,v} || ID_c || AD_c || ID_v || TS_4 || Lifetime_4]$

$\text{Authenticator}_c = E_{k_{c,v}}[ID_c || AD_c || TS_5]$

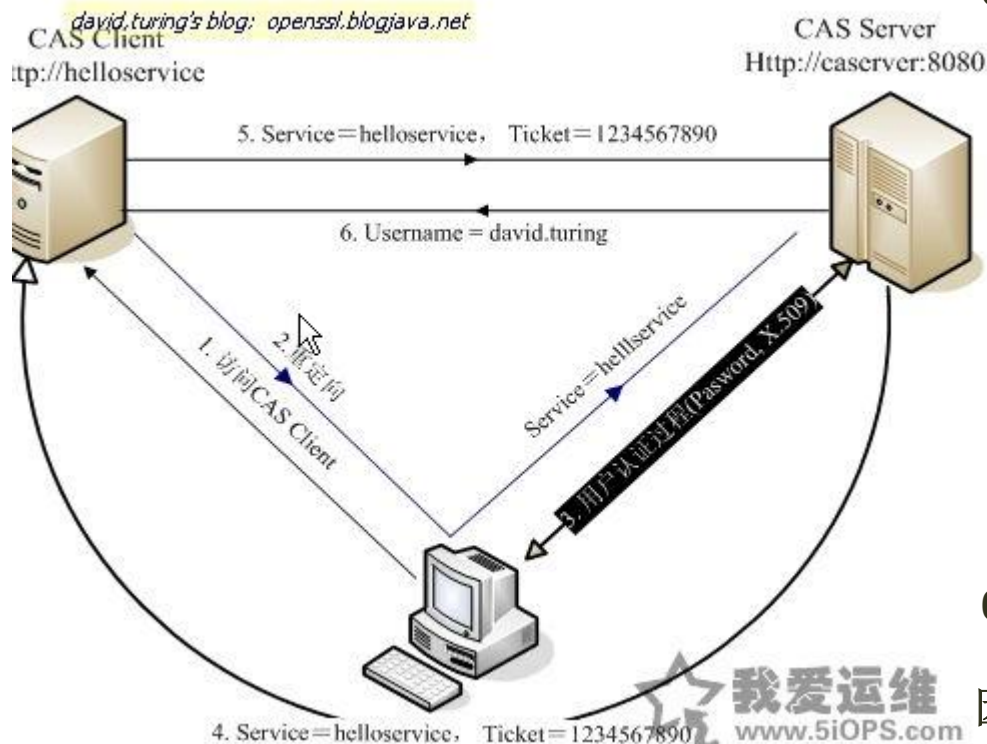


Kerberos安全性



- 时间同步:整个Kerberos的协议都依赖于时钟
- 口令安全性
- 重放攻击（Ticket的生存时间）
- 密钥的管理（认证中心保存大量的共享私钥）
- 对系统程序的破坏（如恶意篡改登录程序）

CAS



<https://www.apereo.org/projects/cas>

http://www.5iops.com/html/2012/sso_0713/181.html

1. 用户浏览器访问web应用 `http://helloservice`, CAS Client会截获用户认证请求;
2. CAS Client将用户重定向到CAS Server, 并再URL中注明用户访问的service名称, 即 `service=helloservice`;
3. CAS Server查看用户的cookie中是否有Ticket Granting Ticket(TGT):
 1. 已有TGT, 则说明用户已经登陆过, 不需要再登录, 因此直接为用户访问的Service颁发一个Service Ticket (ST);
 2. 没有TGT, 则给出用户登录界面, 让用户输入用户名和密码, 验证Credential后决定是否颁发TGT, 直到决定是否颁发ST;
4. 用户被重定向回web应用, 并再url中携带service和ST信息;
5. web应用中的CAS Client向CAS Server验证Service和ST的有效性;
6. 如果ST有效, CAS Server象Client返回用户的身份。

因此web应用本身不需要保存用户credential的信息, 所有用户认证都是在CAS Server端完成的。

Service Ticket在使用一次以后就过期, 因此CAS Server和CAS Client之间可以不用采用SSL连接。但是CAS Server给用户浏览器颁发的TGT是有有效期的(默认2小时), 所以CAS Server一般被配置成https从而保证TGT在发给用户浏览器的过程中的安全性。



OAuth产生的背景



- 随着Web服务的增多，用户希望这些服务能够协同工作来满足新的需求。没有任何一个站点可以完美地满足用户的所有需求，用户可以使用一个站点保存照片，一个存放视频，一个收发邮件等等。为了实现这种整合，一个站点需要访问另一个站点的用户资源，而这些资源经常是受保护的（家庭照片、工作文档、银行记录）。他们需要一个能进入这些站点的授权。

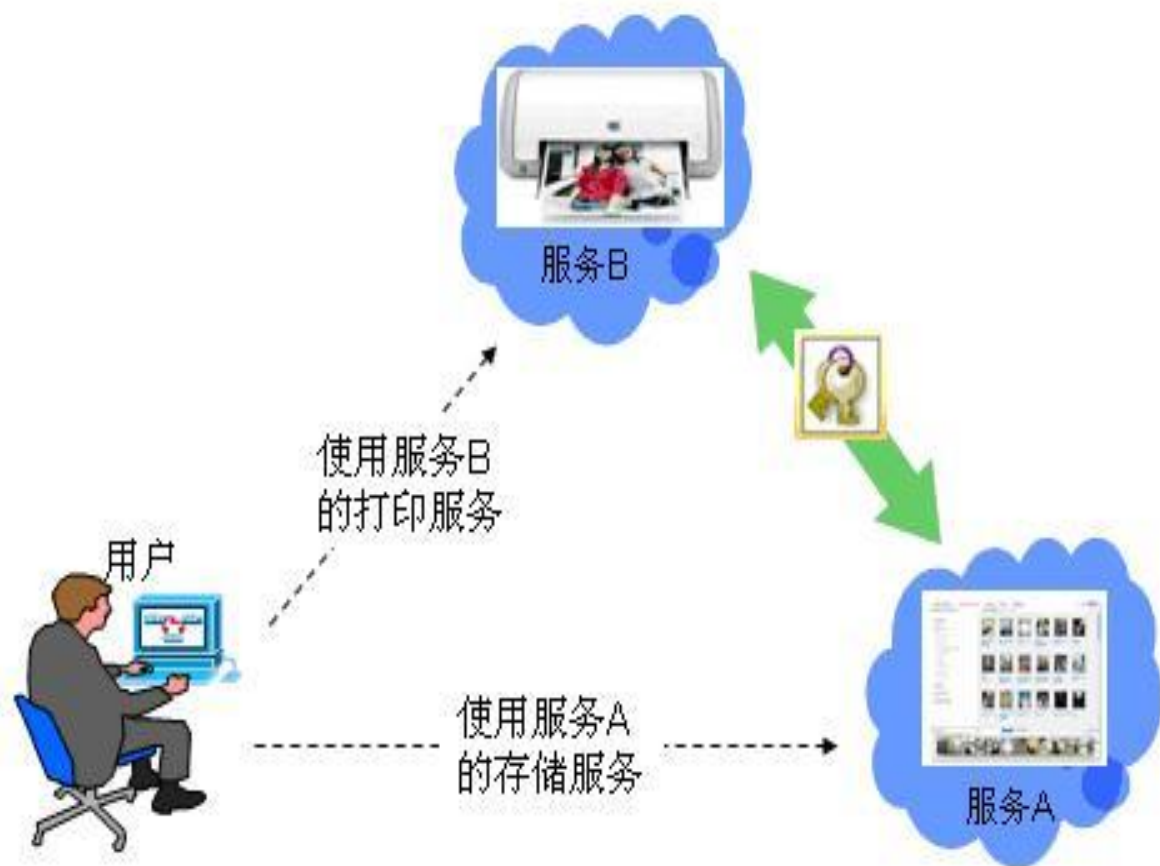
- 一个实例：Basic Auth要求Twitter应用T把用户名和口令直接附加在HTTP或HTTPS协议头中发送给Twitter API。这样，T势必要求用户在其应用中输入自己的Twitter用户名和口令，从而可以把Twitter的用户名和口令附加在HTTP(S)协议中发送给Twitter。

这样T的开发者就能知道使用了T的用户的所有用户名和密码，这样开发者就能随意使用这些Twitter账号登陆Twitter做任何操作了。比如，可以修改用户的Twitter密码，甚至直接去Twitter的Settings中删除这个帐号。这将带来潜在的安全性问题。

- API访问的授权



Oauth应用实例



方法一：用户可能先将待打印的图片从服务A上下载下来并上传到服务B上打印

用户在两家服务提供商的网站上各自注册了两个用户，假设这两个用户名各不相同，密码也各不相同。当用户要使用服务B打印存储在服务A上的图片时，用户该如何处理？

方法二：用户将在服务A上注册的用户名与密码提供给服务B，服务B使用用户的帐号再去服务A处下载待打印的图片，



○ **OAuth** 为用户提供了一种方法，可以使服务**A**在用户的许可下产生一令牌发送给服务**B**，服务**B**使用此令牌便可以访问用户在服务**A**上存储的资源。此方法避免了用户直接将服务**A** 的用户名和密码告诉服务**B**而造成用户信息泄露的问题。



OAuth是什么



- Open Authentication



- OAuth协议为用户资源的授权提供了一个安全的、开放而又简易的标准。

- 基于令牌模式的授权：允许用户提供一个令牌，而不是用户名和密码来访问他们存放在特定服务提供者的数据。每一个令牌授权一个特定的网站在特定的时段（例如，接下来的**2**小时内）内访问特定的资源（例如仅仅是某一相册中的视频）。

<http://oauth.net>



OAuth的版本



- 2007年12月4日发布了OAuth Core 1.0:
- 此版本的协议存在严重的安全漏洞: OAuth Security Advisory: 2009.1, 更详细的介绍可以参考: Explaining the OAuth Session Fixation Attack.
- 2009年6月24日发布了OAuth Core 1.0 Revision A:
- 2010年4月, OAuth 1.0协议发表为RFC 5849, 一个非正式RFC。
- 目前, OAuth2.0协议是OAuth的下一个全新版本, 与OAuth1.0并不兼容。OAuth 2.0能够同时支持“Web应用、桌面应用、移动终端、家庭设备”等等。在OAuth 2.0中, server将发行一个短有效期的access token和长生命期的refresh token
- April 2010: OAuth 2.0 <draft-ietf-oauth-v2-00.txt> published co-authored by Eran, Dick, David.

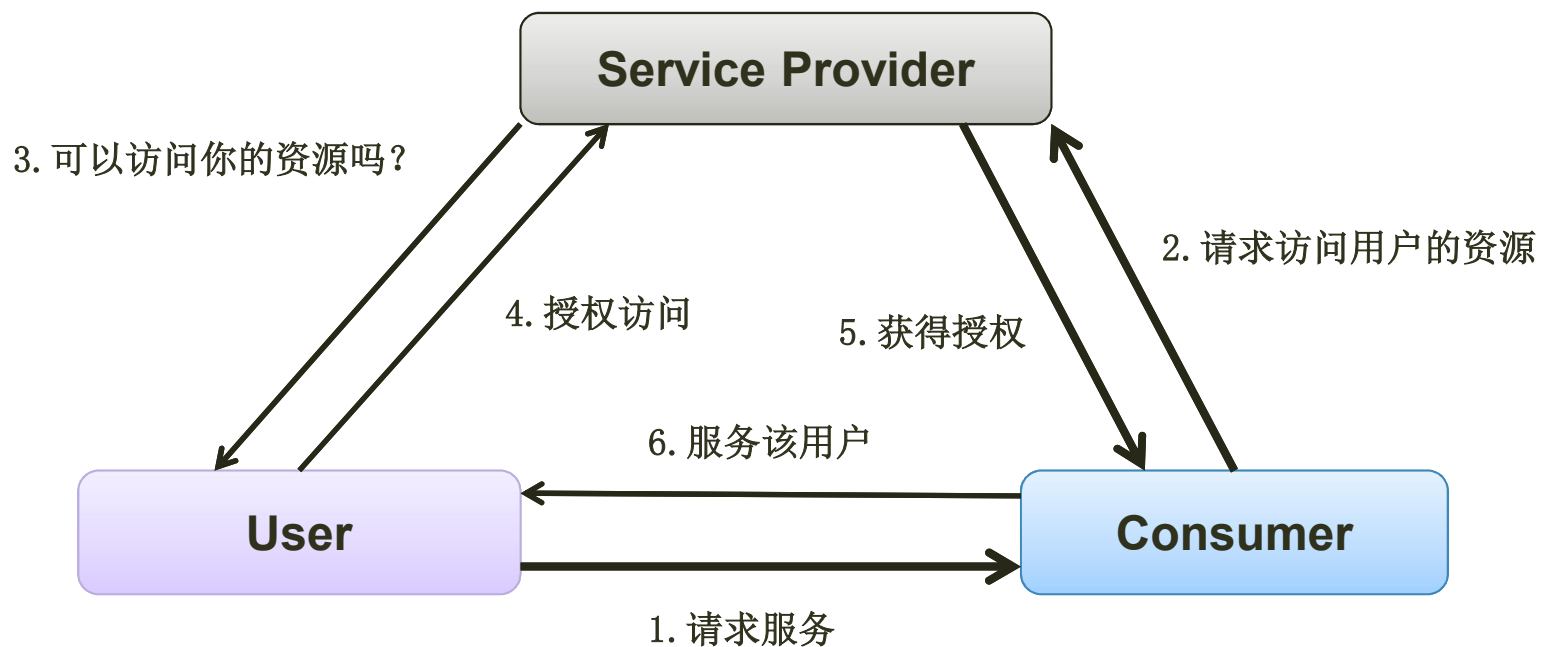
<http://oauth.net/2/>



OAuth中的三种角色



- **Service Providers**（服务提供方）：拥有需要授权才能使用的**API**的一方
- **Consumers**（应用程序方）：希望使用**API**的一方
- **Users**（最终用户）：资源的拥有者





应用例：Sina微博账号登录



用微博帐号登录



用户在你的网站上点击“用微博帐号登录”按钮，弹出Oauth授权页面



用户输入微博账号和密码，即可登录，登录后返回你指定的页面



登录成功后，你可以选择与你的网站帐号绑定，也可以直接用微博帐号实现同等功能。

先注册一个应用，得到应用专属App Key和App Secret。对应用发出的请求添加签名，向新浪微博开放平台表明应用的合法性。

因为OAuth2.0的客户端验证授权会获得用户明文密码，所以实行有限开放。

授权级别和OAuth2.0 access_token有效期对应表：

授权级别	测试	普通	中级	高级	合作
授权有效期	1天	7天	15天	30天	90天

申请条件：

应用分类属于桌面客户端、手机客户端。

应用使用人数在30000以上。

应用本身已经通过开放平台文案、广场审核，并在广场上展示超过15天。

应用本身功能与新浪微博关联紧密。

如果您的应用符合以上申请条件，可在应用控制台，接口管理标签下的授权机制选项中进行在线申请。

<http://open.weibo.com/wiki/Oauth>



你的应用

步骤

新浪微博开放平台

用户

获取request token
oauth/request_token

1

创建request token及相应的
密钥 (Secret)

将用户重定向到授权页
http://api.t.sina.com.cn/oauth/authorize?oauth_token=token

2

询问用户是否对应用授权

用户授权或者拒绝授权

如果用户同意授权，重定向至
Callback_url (您的应用)

oauth/access_token
用RequestToken向新浪微博
换取Access Token

3

创建并返回Access Token及
Secret

account/verify_credentials
获取该Access Token的信息
及对应用户的信息

4

返回该Token的信息及对应用户的信息



新浪微博 · 开放平台



你的应用

步骤

新浪微博开放平台

用户



获取request token
oauth/request_token

(1)

创建request token以及
相应密钥 (Secret)



将用户重定向到授权页
http://api.t.sina.com.cn/oauth/authorize?oauth_token=token

(2)

询问用户是否对应用授权

用户授权或者
拒绝授权

如果用户同意授权重定向
至Callback_url (您的应用)

oauth/access_token
用Request Token向新浪
微博换取Access Token

(3)

创建并返回Access Token及
Secret

account/verify_credentials
获取该Access Token的信息
及对应用户的信息

(4)

返回该Token的信息及对应用
户的信息



Oauth1.0开发



- 为了实现Oauth认证server API需要提供三个接口:

- Temporary Credential Request

<https://photos.example.net/initiate>

- Resource Owner Authorization URI:

<https://photos.example.net/authorize>

- Token Request URI:

<https://photos.example.net/token>

前面的例子, printer— Oauth client, photo----Oauth server



- (1) OAuth client, 向 OAuth server 注册, server 返回 client credentials

- Client Identifier

dpf43f3p2l4k3l03

- Client Shared-Secret:

kd94hf93k423kf44



○ (2) client向server请求临时证书 (temporary credentials), server返回证书

POST /initiate HTTP/1.1

Host: photos.example.net

Authorization: OAuth realm="Photos", 注册时的Client Identifier

oauth_consumer_key="dpf43f3p2l4k3l03",

oauth_signature_method="HMAC-SHA1",

oauth_timestamp="137131200",

oauth_nonce="wljqoS",

oauth_callback="http%3A%2F%2Fprinter.example.com%2Fready", 资源请求方

oauth_signature="74KNZJeDHnMBp0EMJ9ZHt%2FXKycU%3D" 认证


○ HTTP/1.1 200 OK

Content-Type: application/x-www-form-urlencoded 得到临时token, 表示认证通过

oauth_token=hh5s93j4hdidpola&oauth_token_secret=hdhd0244k9j7ao03&

oauth_callback_confirmed=true



- (3) 客户端 (**client**) 将用户代理 (**user-agent**) 重定向到**server**的授权界面  刚得的临时证书

- https://photos.example.net/authorize?oauth_token=hh5s93j4hdidpola

- 用户填写用户名和密码，由**server** 验证，授权成功后**server**将用户代理重定向到 (2) 中的**oauth_callback**，返回给用户代理一个verifier code，此值在下一步用来验证用户代理的唯一性。

- <http://printer.example.com/ready?> 临时证书

[oauth_token=hh5s93j4hdidpola&oauth_verifier=hfdp7dh39dks9884](#)





○ (4) 客户端 (client) 向server申请令牌证书 (token credentials) 。

POST /token HTTP/1.1

Host: photos.example.net

Authorization: OAuth realm="Photos",

oauth_consumer_key="dpf43f3p2l4k3l03",

oauth_token="hh5s93j4hdidpola",

oauth_signature_method="HMAC-SHA1",

oauth_timestamp="137131201",

oauth_nonce="walatlh",

oauth_verifier="hfdp7dh39dks9884",

oauth_signature="gKgrFCywp7rO0OXSjdot%2FIHF7IU%3D"

(3) 中

Server返回令牌证书

访问令牌，用于多次访问资源

○ HTTP/1.1 200 OK

Content-Type: application/x-www-form-urlencoded

oauth_token=nnch734d00sl2jdk&oauth_token_secret=pfkkdhi9sl3r4s00

OAuth库和资源



ActionScript/Flash

oauth-as3 <http://code.google.com/p/oauth-as3/>

A flex oauth client

<http://www.arccgis.com/home/item.html?id=ff6ffa302ad04a7194999f2ad08250d7>

C/C++

QTweetLib <http://github.com/minimoog/QTweetLib> (网址失效)

libOAuth <http://liboauth.sourceforge.net/>

Clojure

clj-oauth <http://github.com/mattrepl/clj-oauth>

.net

oauth-dot-net <http://code.google.com/p/oauth-dot-net/>

DotNetOpenAuth <http://www.dotnetopenauth.net/>

Erlang

erlang-oauth <http://github.com/tim/erlang-oauth>

Java

Scribe <http://github.com/fernandezpablo85/scribe-java>

oauth-signpost <http://code.google.com/p/oauth-signpost/>

Javascript

oauth in js <http://oauth.googlecode.com/svn/code/javascript/>

OAuth库和资源



Objective-C/Cocoa & iPhone programming

OAuthCore <http://bitbucket.org/atebits/oauthcore> (网址失效)

MPOAuthConnection <http://code.google.com/p/mpoauthconnection/>

Objective-C OAuth <http://oauth.googlecode.com/svn/code/obj-c/>

Perl

Net::OAuth <http://oauth.googlecode.com/svn/code/perl/>

PHP

tmhOAuth <http://github.com/thematttharris/tmhOAuth>

oauth-php <http://code.google.com/p/oauth-php/>

Python

python-oauth2 <http://github.com/brosner/python-oauth2> (网址失效)

Qt

qOAuth <http://github.com/ayoy/qoauth>

Ruby

OAuth ruby gem <http://oauth.rubyforge.org/> (网址失效)

Scala

DataBinder Dispatch <http://dispatch.databinder.net/About> (网址失效)



其它话题

● 实体认证的作用

- 认证分为实体认证和消息认证。
- 实体认证是对通信主体的认证，目的是识别通信方的真实身份，防止假冒，常用数字签名的方法；
- 消息认证是对通信数据的认证，目的是验证消息在传送或存储过程中是否被篡改，一般用消息摘要的方法。



其它话题



- **Passfaces:** 一个身份验证系统，让用户识别认识的人脸
- **CAPTCHA:** 由计算机来区分人和计算机