Radius 原理

RADIUS（Remote Authentication Dial In User Service）协议最初是由 Livingston 公司提出的，原先的目的是为拨号用户进行认证和计费。后来经过多次改进，形成了一项通用的认证计费协议。RADIUS 认证要用到基于挑战/应答（Challenge/Response）的认证方式。

RADIUS 是一种 C/S 结构的协议，它的客户端最初就是 NAS（Net Access Server）服务器，现在任何运行 RADIUS 客户端软件的计算机都可以成为 RADIUS 的客户端。RADIUS 协议认证机制灵活，可以采用 PAP、CHAP 或者 Unix 登录认证等多种方式。RADIUS 是一种可扩展的协议，它进行的全部工作都是基于 Attribute-Length-Value 的向量进行的。

RADIUS 的基本工作原理是用户接入 NAS，NAS 向 RADIUS 服务器使用 Access-Require 数据包提交用户信息，包括用户名、密码等相关信息，其中用户密码是经过 MD5 加密的，双方使用共享密钥，这个密钥不经过网络传播；RADIUS 服务器对用户名和密码的合法性进行检验，必要时可以提出一个 Challenge，要求进一步对用户认证，也可以对 NAS 进行类似的认证；如果合法，给 NAS 返回 Access-Accept 数据包，允许用户进行下一步工作，否则返回 Access-Reject 数据包，拒绝用户访问；如果允许访问，NAS 向 RADIUS 服务器提出计费请求 Account-Require，RADIUS 服务器响应 Account-Accept，对用户的计费开始，同时用户可以进行自己的相关操作。

RADIUS 服务器和 NAS 服务器通过 UDP 协议进行通信，RADIUS 服务器的 1812 端口负责认证，1813 端口负责计费工作。采用 UDP 的基本考虑是因为 NAS 和 RADIUS 服务器大多在同一个局域网中，使用 UDP 更加快捷方便。

RADIUS 协议还规定了重传机制。如果 NAS 向某个 RADIUS 服务器提交请求没有收到返回信息，那么可以要求备份 RADIUS 服务器重传。由于有多个备份 RADIUS 服务器，因此 NAS 进行重传的时候，可以采用轮询的方法。如果备份 RADIUS 服务器的密钥和以前 RADIUS 服务器的密钥不同，则需要重新进行认证。

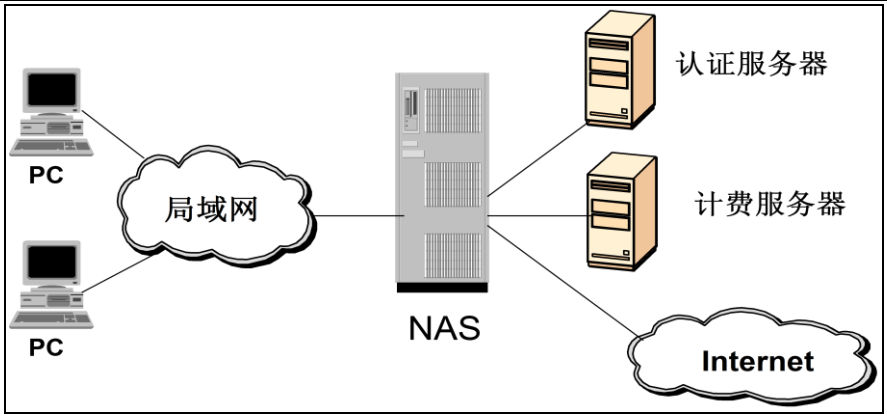
RADIUS 协议应用范围很广，包括普通电话、上网业务计费，对 VPN 的支持可以使不同的拨入服务器的用户具有不同权限。

越来越多的软件开始内置 RADIUS 协议，以方便和各种业务系统对接，如 ERP、OA、VPN 等，随着这些软件对安全性要求的日益增高，一部分软硬件开始采用双因素方式进行认证，静态密码 + 动态密码（动态令牌/短信密码）形式，那些支持 RADIUS 协议的 ERP、OA、VPN，在无需对客户端做任何修改情况下就能很好整合动态密码，增强系统身份认证安全。

基本概念

简称	英文全称	中文解释
AAA	Authentication、Authorization、Accounting	验证、授权、计费

PAP	Password Authentication Protocol	口令验证协议
CHAP	Challenge-Handshake Authentication Protocol	挑战握手认证协议
NAS	Network Access Server	网络接入服务器
RADIUS	Remote Authentication Dial In User Service	远程用户拨号认证服务
TCP	Transmission Control Protocol	传输控制协议
UDP	User Datagram Protocol	用户数据报协议
PPP	Point to Point Protocol	点对点协议



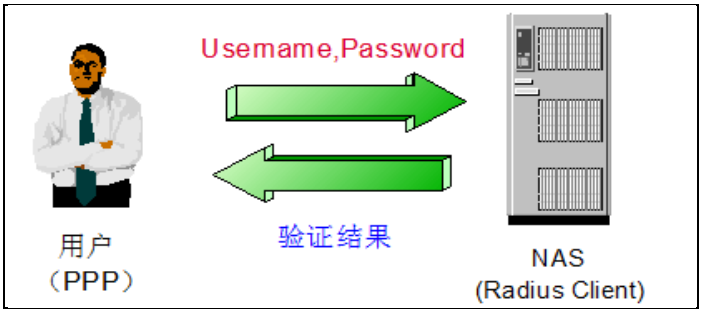
AAA 可以通过两种途径实现：

1、 在 NAS 端进行认证、授权和计费

2、 通过协议进行远程认证、授权和计费。实现了 AAA 的协议有：RADIUS、Kerberos、TACACS 等

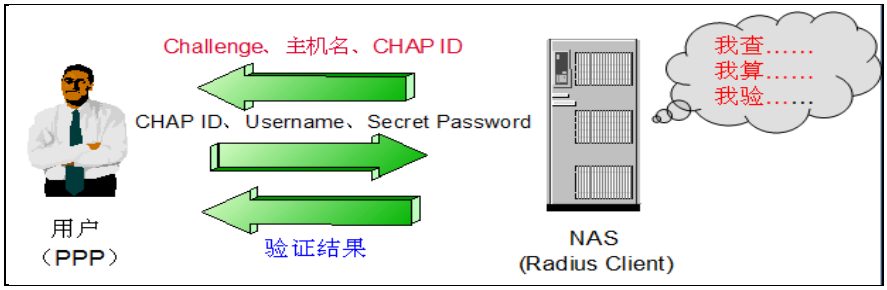
当用户想要通过某个网络(如电话网)与 NAS 建立连接从而获得访问其他网络的权利时，NAS 可以选择在 NAS 上进行本地认证计费，或把用户的信息传递给 RADIUS 服务器，由 Radius 进行认证计费；RADIUS 协议规定了 NAS 与 RADIUS 服务器之间如何传递用户信息和计费信息；RADIUS 服务器负责接收用户的连接请求，完成验证，并把传递服务给用户的配置信息返回给 NAS.

本地(NAS)验证——PAP 方式：



用户以明文的形式把用户名和密码传递给 NAS，NAS 根据用户名在 NAS 端查找本地数据库，如果存在相同的用户名和密码表明验证通过，否则验证未通过。

本地(NAS)验证——CHAP 方式：



Secret Password = MD5（Chap ID + Password + challenge）

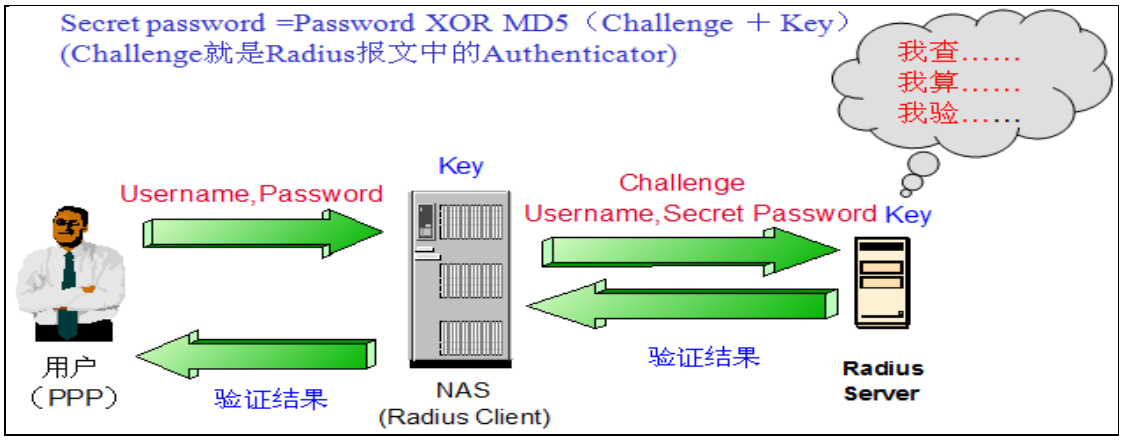
当用户请求上网时，

1、 服务器产生一个 16 字节的随机码(challenge)给用户（同时还有一个 ID 号，本地路由器的 hostname）.

2、 用户端得到这个包后使用自己独用的设备或软件对传来的各域进行加密，生成一个 Secret Password 传给 NAS。

3、 NAS 根据传来的用户名查找自己本地的数据库，得到和用户端进行加密所用的一样的密码，然后根据原来的 16 字节的随机码进行加密，将其结果与 Secret Password 作比较，如果相同表明验证成功，否则失败.

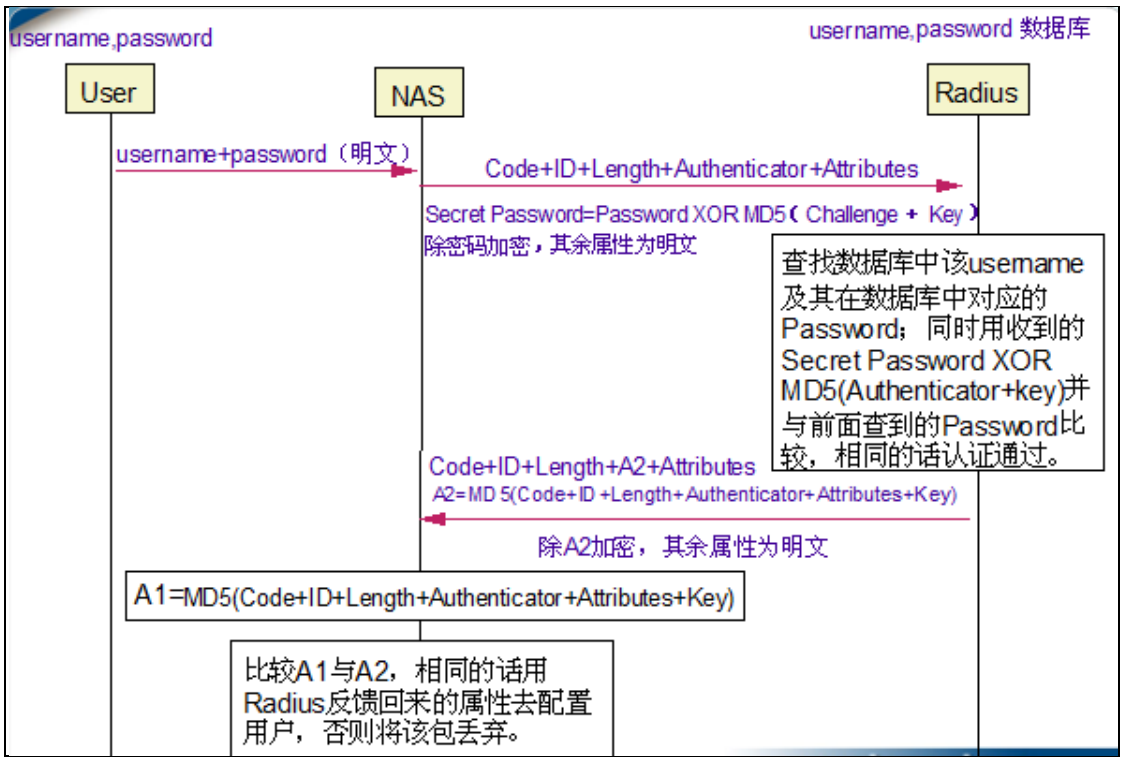
远端(Radius)验证——PAP 方式：



如果用户配置了 RADIUS 验证，其 PAP 验证过程如下：

采用 PAP 验证：

1. 用户以明文的形式把用户名和他的密码传递给 NAS。
2. NAS 把用户名和加密过的密码放到认证请求包的相应属性中传递给 RADIUS 服务器
3. 根据 RADIUS 服务器的返回结果来决定是不允许用户上网。



例 1：

1)NAS：192.168.1.16 发送 Access-Request UDP 数据包到 RADIUS Server 。

User-name : nemo

Port logging in : 3

Code = 1 (Access-Request)

ID = 0

Length = 56

Request Authenticator = {16 octet random number}

Attributes:

User-Name = "nemo"

User-Password = {16 octets of Password padded at end with nulls,
XORed with MD5(key|Request Authenticator)}

NAS-IP-Address = 192.168.1.16

NAS-Port = 3

2)RADIUS server 验证了 nemo, 并且发送了 Access-Accept UDP 数据包到接入服务器，告诉把用户 nemo 登陆到主机 192.168.1.3.

Code = 2 (Access-Accept)

ID = 0 (same as in Access-Request)

Length = 38

Response Authenticator = {16-octet MD-5 checksum of the code (2),id (0), Length (38), the Request Authenticator from above, the attributes in this reply, and the shared secret}

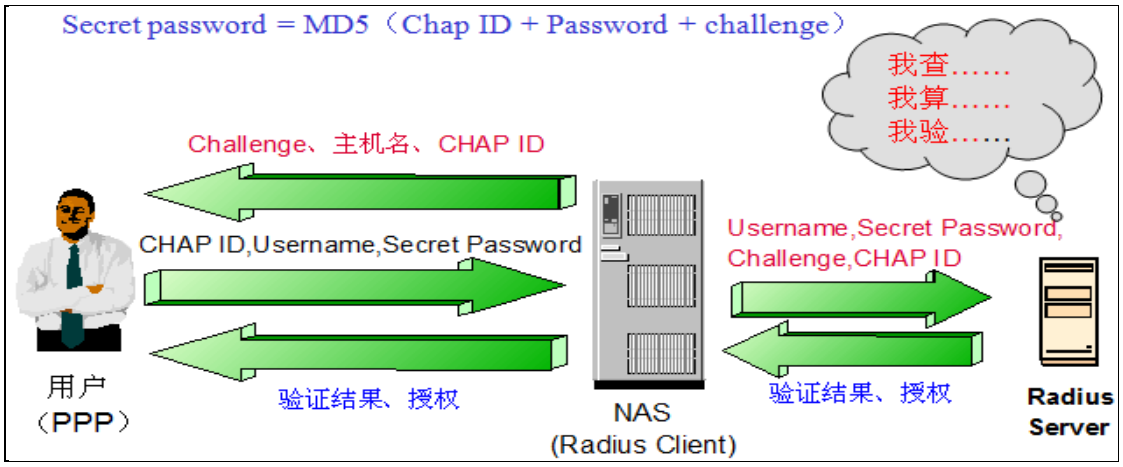
Attributes:

Service-Type = Login-User

Login-Service = Telnet

Login-Host = 192.168.1.3

远端(Radius)验证——CHAP 方式：



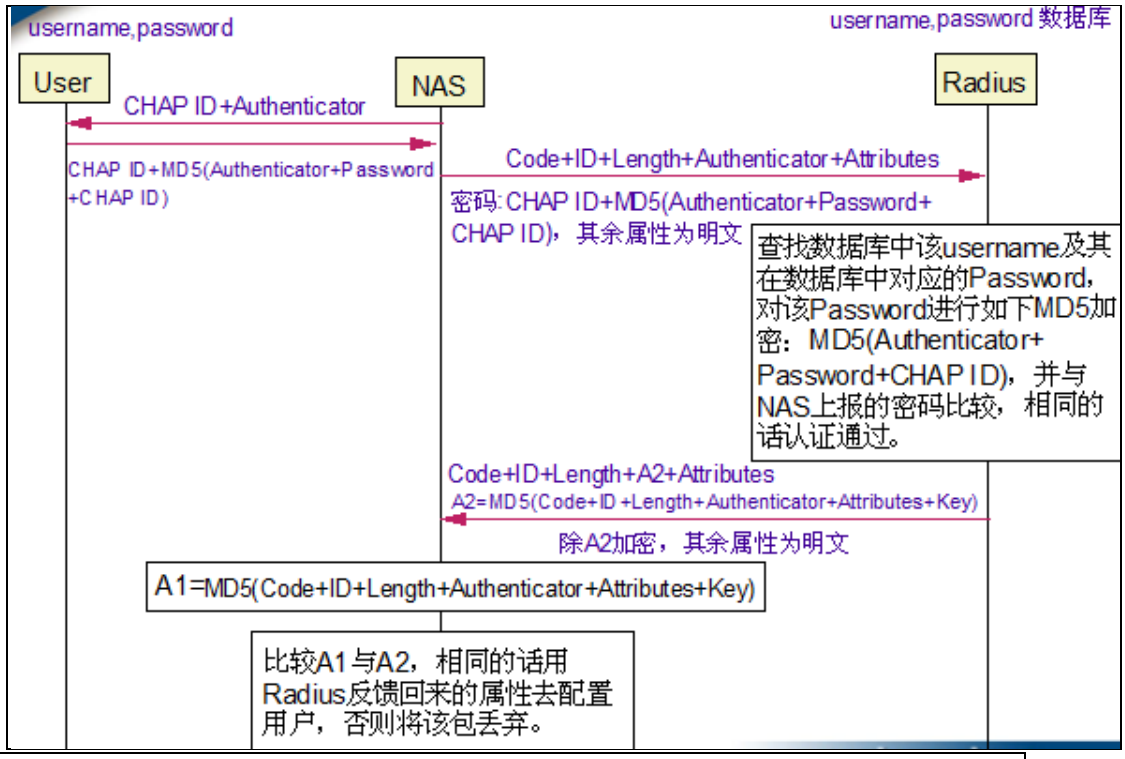
如果用户配置了 RADIUS 验证，其 CHAP 验证过程如下：

1、 当用户请求上网时，NAS 产生一个 16 字节的随机码给用户（同时还有一个 ID 号，本地路由器的 host name）。

2、 用户端得到这个包后使用自己独有的设备或软件对传来的各域进行加密，生成一个 response 传给 NAS。

3、 NAS 把传来的 CHAP ID 和 response 分别作为用户我和密码，并把原来的 16 字节随机码（challenge）传给 RADIUS 服务器。

4、 RADIUS 根据用户名在服务器端查找数据库，得到和用户端进行加密所有的一样的密码，然后根据传来的 16 的字节的随机码进行加密，将其结果与传来的 password 作比较，如果相同表明验证通过，如果不相同表明验证失败。另外，如果验证成功，RADIUS 服务器同样也要以生成一个 16 字节的随机码对用户进行询问(challenge,暂不支持)



例 2：

接入服务器 192.168.1.16 发送一个 Access-Request UDP 数据包到 RADIUS Server 。

User-name : floppy

Port logging in : 20

Protocol : PPP

接入服务器发送的数据包包含属性 Service-Type = Framed user， Framed-Protocol=PPP 暗示 RADIUS server 这个用户要使用 PPP 服务。

Code = 1 (Access-Request)

ID = 1

Length = 71

Request Authenticator = {16 octet random number also used as CHAP challenge}

Attributes:

User-Name = "floppy"

CHAP-Password = {1 octet CHAP ID followed by 16 octet CHAP response}

NAS-IP-Address = 192.168.1.16

NAS-Port = 20

Service-Type = Framed-User

Framed-Protocol = PPP

RADIUS server 验证 floppy, 并发送 Access-Accept UDP 数据包到 NAS 告诉 NAS 可以允许 PPP 服务并从它的动态地址池中分配一个网络地址给用户。

Code = 2 (Access-Accept)

ID = 1 (same as in Access-Request)

Length = 56

Response Authenticator = {16-octet MD-5 checksum of the code (2),id (1), Length (56), the Request Authenticator from above, the attributes in this reply, and the key}

Attributes:

Service-Type = Framed-User

Framed-Protocol = PPP

Framed-IP-Address = 255.255.255.254

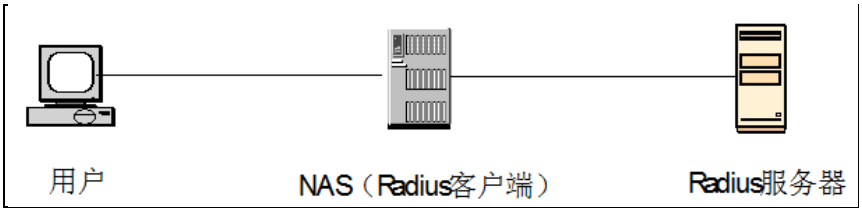
Framed-Routing = None

Framed-Compression = 1 (VJ TCP/IP Header Compression)

Framed-MTU = 1500

RADIUS 协议

Radius 服务分客户端和服务器端



通常对 Radius 协议的服务端口号是 1645（认证）、1646（计费）或 1812（认证）、1813（计费）。

下图是 radius 协议包结构：

```
Packet :
     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
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |     Code      |  Identifier   |            Length             |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |                                                               |
    |                         Authenticator                         |
    |                                                               |
    |                                                               |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |  Attributes ...
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-

    Attribute :
     0                   1                   2
     0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
    |     Type      |    Length     |  Value ...
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
```

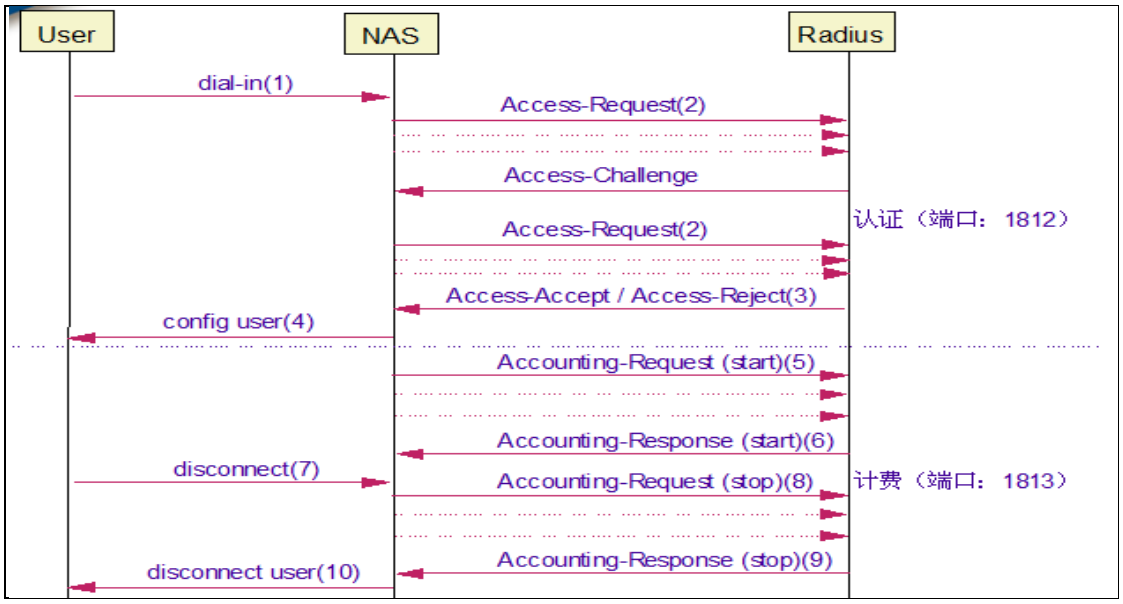
各个域的解释:

Num	域名	长度	描述
1	Code(包类型)	1 字节	指示 Radius 包的类型
2	Identifier (包标识)	1 字节	用于匹配请求包和响应包，同一组请求包和响应包的 Identifier 应相同。
3	Length(包长度)	2 字节	整个包的长度
4	Authenticator(验证字)	16 字节	用于对包进行签名
5	Attributes		属性

1） Code，包类型：

类型名	描述
1	Access-Request——认证请求过程
2	Access-Accept——认证接受过程
3	Access-Reject——认证拒绝过程
4	Accounting-Request——计费请求过程
5	Accounting-Response——计费响应过程

Radius 认证计费过程：



1、用户拨入后（1），所拨入的设备（比如 NAS）将拨入用户的用户信息（比如用户名、口令、所占用的端口等等）打包向 RADIUS 服务器发送（2）。

2、如果该用户是一个合法的用户，那么 Radius 告诉 NAS 该用户可以上网，同时传回该用户的配置参数（3）；否则，Radius 反馈 NAS 该用户非法的信息（3）。

3、如果该用户合法，MAS 就根据从 RADIUS 服务器传回的配置参数配置用户（4）。如果用户非法，NAS 反馈给用户出错信息并断开该用户连接（4）。

4、如果用户可以访问网络，RADIUS 客户要向 RADIUS 服务器发送一个记费请求包表明对该用户已经开始记费（5），RADIUS 服务器收到并成功记录该请求包后要给予响应（6）。

5、当用户断开连接时（连接也可以由接入服务器断开）（7），RADIUS 客户向 RADIUS 服务器发送一个记费停止请求包，其中包含用户上网所使用网络资源的统计信息（上网时长、进/出的字节/包数等）（8），RADIUS 服务器收到并成功记录该请求包后要给予响应（9）。

> 1、RADIUS 的通信是用“请求 - 响应”方式进行的，即：客户发送一个请求包，服务器收到包后给予响应。
>
> 2、RADIUS 协议采用的是 UDP 协议，数据包可能会在网络上丢失，如果客户没有收到响应，那么可以重新发送该请求包。多次发送之后如果仍然收不到响应，RADIUS 客户可以向备用的 RADIUS 服务器发送请求包。

2） Identifier，包标识：

用以匹配请求包和响应包，该字段的取值范围为 0～255

协议规定：

1、 在任何时间，发送给同一个 Radius 服务器的不同包的 Identifier 域不能相同，如果出现相同的情况，Radius 将认为后一个包是前一个包的拷贝而不对其进行处理

2、 Radius 针对某个请求包的响应包应该与该请求包在 Identifier 上相匹配（相同）。

3） Length，包长度：

整个包长度,包括 Code,Identifier,Length,Authenticator,Attributes 域的长度。

4） Authenticator，验证字：

该验证字分为两种：

1、请求验证字---Request Authenticator

用在请求报文中,必须为全局唯一的随机值。

2、响应验证字---Response Authenticator

用在响应报文中，用于鉴别响应报文的合法性。

响应验证字＝MD5(Code+ID+Length+请求验证字+Attributes+Key)

RADIUS 主要特征

Radius 是一种流行的 AAA 协议，同时其采用的是 UDP 协议传输模式。

1）为什么使用 UDP？

1、 NAS 和 RADIUS 服务器之间传递的一般是几十至上百个字节长度的数据，用户可以容忍几秒到十几秒的验证等待时间。当处理大量用户时服务器端采用多线程，UDP 简化了服务器端的实现过程。

2、 TCP 是必须成功建立连接后才能进行数据传输的，这种方式在有大量用户使用的情况下实时性不好。

3、当向主用服务器发送请求失败后，还要必须向备用的服务器发送请求。于是 RADIUS 要有重传机制和备用服务器机制，它所采用的定时，TCP 不能很好的满足。

2）C/S 结构

RADIUS 采用客户/服务器（Client/Server）结构：

1、RADIUS 的客户端通常运行于接入服务器（NAS）上，RADIUS 服务器通常运行于一台工作站上，一个 RADIUS 服务器可以同时支持多个 RADIUS 客户（NAS）。

2、RADIUS 的服务器上存放着大量的信息，接入服务器（NAS）无须保存这些信息，而是通过 RADUIS 协议对这些信息进行访问。这些信息的集中统一的保存，使得管理更加方便，而且更加安全。

3、RADIUS 服务器可以作为一个代理，以客户的身份同其他的 RADIUS 服务器或者其他类型的验证服务器进行通信。用户的漫游通常就是通过 RADIUS 代理实现的。

3）网络安全

RADIUS 协议的加密是使用 MD5 加密算法进行的，在 RADIUS 的客户端（NAS）和服务器端（Radius Server）保存了一个密钥（key），RADIUS 协议利用这个密钥使用 MD5 算法对 RADIUS 中的数据进行加密处理。密钥不会在网络上传送(两端使用一个都知晓的密钥，从而不需要再从网络上传输告之)。RADIUS 的加密主要体现在两方面：

1、包加密：

在 RADIUS 包中，有 16 字节的验证字（authenticator）用于对包进行签名，收到 RADIUS 包的一方要查看该签名的正确性。如果包的签名不正确，那么该包将被丢弃，对包进行签名时使用的也是 MD5 算法（利用密钥），没有密钥的人是不能构造出该签名的。

2、口令加密：

在认证用户时，用户的口令不会在网上明文传送，而是使用了 MD5 算法对口令进行加密。没有密钥的人是无法正确加密口令的，也无法正确地对加密过的口令进行解密。

包的签名与加密：

包的签名指的是 RADIUS 包中 16 字节的 Authenticator，我们称其为“验证字”。

·认证请求包：

RequestAuth=Authenticator，认证请求包的验证字是一个不可预测的 16 字节随机数。这个随机数将用于口令的加密。

·认证响应包

ResponseAuth = MD5(Code+ID+Length+Authenticator+Attributes+Key)

·计费请求包

RequestAcct = MD5(Code+ID+Length+16ZeroOctets+Attributes+Key)

·计费响应包

ResponseAcct = MD5(Code+ID+Length+RequestAcct+Attributes+Key)

口令的加密：

称共享密钥（key）为 Key；16 字节的认证请求验证字(Authenticator)为 Auth；将口令(Password)分割成 16 字节一段（最后一段不足 16 字节时用 0 补齐），为 p1、p2 等；加密后的口令块为 c(1)、c(2)等。下面运算中 b1、b2 为中间值：

b1 = MD5(Key + Auth)　　c(1) = p1 xor b1

b2 = MD5(Key + c(1))　　c(2) = p2 xor b2

…… …… ……

bi = MD5(Key+ c(i-1))　　c(i) = pi xor bi

那么加密后的口令为 c(1)+c(2)+...+c(i)。

上面是协议规定的算法，也有的 RADIUS 服务器为了实现起来简单，修改了上述的算法，具体的讲，b1 的算法同上，但 bi=b2=b1(i>=1)，其他运算不变。当用户的口令长度不超过 16 字节时，两种算法的结果是一样的。

4）灵活的验证机制

RADIUS 协议允许服务器支持多种验证方式，比如 PPP 的 PAP 和 CHAP、UNIX 登录以及其他认证机制。

通常的 RADIUS 服务器都支持 PAP，但有些 RADIUS 服务器不支持 CHAP，原因在于有些 RADIUS 服务器在保存用户的口令时是加密保存的。而要验证一个 CHAP 用户的合法性，必须能够获得该用户的明文口令才行。

5）扩展性

RADIUS 协议具有很好的扩展性。RADIUS 包是由包头和一定数目的属性（Attribute）构成的。新属性的增加不会影响到现有协议的实现。

通常的 NAS 厂家在生产 NAS 时，还同时开发与之配套的 RADIUS 服务器。为了提供一些功能，常常要定义一些非标准的（RFC 上没有定义过的）属性。关于各个厂家有那些扩展的属性，一般可以从相应的 RADIUS 服务器的字典（dictionary）文件中找到(tinyRadius 的字典在：/org/tinyradius/dictionary/default_dictionary)。

计费保护：

由于计费是个用户敏感的问题 ，我们提供的计费信息要求准确外更要求完备，不能丢失话单，而 Radius 计费服务器的计费很容易受通讯不畅或计费服务器工作超负荷等意外因素的影响而丢失话单 ， 为此我们提供了本地计费的保护能力，即话单从 NAS 发送出去以后如果没有收到计费服务器的确认信息我们就认为计费失败 ，并将该话单（ 不论是实时计费中间话单还是离线话单 ）送到本地话单池中。

RADIUS 协议属性

认证报文的常用属性

属性值	属性名称	意义
1	User-Name	用户名
2	User-Password	用户密码
3	Chap-Password	Chap 认证方式中的用户密码
4	Nas-IP-Address	NAS 的 ID 地址
5	Nas-Port	用户接入端口号
6	Service-Type	服务类型
7	Framed-Protocol	协议类型
8	Framed-IP-Address	为用户提供的 IP 地址
9	Framed-IP-NetMask	地址掩码
10	Framed-Routing	为路由器用户设置的路由方式
11	Filter-Id	过滤表的名称
12	Framed-MTU	为用户配置的最大传输单元
13	Framed-Compression	该连接使用压缩协议
14	Login-IP-Host	对 login 用户提供的可连接主机的 ip 地址
15	Login-Service	对 login 用户可提供的服务
16	Login-TCP-Port	TCP 服务端口
18	Reply-Message	认证服务器返回用户的信息
24	State	认证服务器发送 challenge 包时传送的需在接下来的认证报文中回应的字符串（与 Acess-Challenge 相关的属性）
25	Class	认证通过时认证服务器返回的字符串信息，要求在该用户的计费报文中送给计费服务器
26	Vendor-Specific	可扩展属性
27	Session-Timeout	在认证通过报文或 Challenge 报文中，通知 NAS 该用户可用的会话时长 （时长预付费）
28	Idle-Timeout	允许用户空闲在线的最大时长
32	NAS-Identifier	标识 NAS 的字符串
33	Proxy-State	NAS 通过代理服务器转发认证报文时服务器添加在报文中的属性
40	Acct-Status-Type	计费请求报文的类型
41	Acct-Delay-Time	Radius 客户端发送计费报文耗费的时间
42	Acct-Input-Octets	输入字节数

43	Acct-Output-Octets	输出字节数
44	Acct-Session-Id	计费会话标识
45	Acct-Authentic	在计费包中标识用户认证通过的方式
46	Acct-Session-Time	用户在线时长
47	Acct-Input-Packets	输入包数
48	Acct-Output-Packets	输出包数
49	Acct-Terminate-Case	用户下线原因
50	Acct-Multi_Session-Id	相关计费会话标识
51	Acct-Link-Count	生成计费记录时多连接会话的会话个数
60	Chap-Challenge	可以代替认证字字段传送 challenge 的属性
61	Nas-Port-Type	接入端口的类型
62	Port-Limit	服务器限制 NAS 为用户开放的端口数

TinyRadius

快速上手：

1）TinyRadius 简介

TingRadius 是一个简单、小巧又灵活的 java Radius 库，它能够发送和接收各种类型的 Radius 协议包。同时它遵循 LGPL(GNU Lesser General Public License，GNU 宽通用公共许可证) 条款。

2）使用 TinyRadius 可以做什么？

1、 在您的 java 程序中，tinyRadius 可以发送与接收 Radius 协议包，包括：Access-Request, Access-Accept, Access-Reject, Access-Challenge, Accounting-Request, Accounting-Response and others
2、 用来作为认证讲求消息的 PAP 和 CHAP 协议
3、 可以添加从字典文件中读取的任意 Radius 属性
4、 发送和接收带用商家特有属性的 Radius 协议包

3）使用 TinyRadius 不能做或不应做什么？

1、 TinyRadius 不能为您搭建一个完成的 Radius 服务器（如果需要请使用 FreeRadius 或 JRadius）
2、 不写 java 代码你是无法直接去连接你的用户数据库的（这个库只能作为你应用程序的插件使用而不是一个独立的服务器）。

TinyRadius 中已经配备了简单的小应用程序来为你展示如何去建立 Radius 服务器和 Radius 客户端。

4）运行 TinyRadius 请注意

1、 TinyRadius 已经在 JDK1.1 到 JDK1.6 上测试并工作的很好，但我建议你使用 JDK1.4 以上版本
2、 你需要使用 Apache 的 Commons Loggin 去编译运行 TinyRadius，在发布包中已经包含了。

实例 1：简易的认证

如果你不需要设置特殊的属性值，你可以使用 RadiusClient 中的 authenticate()方法：

```
RadiusClient rc = new RadiusClient(host, sharedSecret);
if (rc.authenticate(userName, password)) {
    ...
```

实例 2：发送带有多属性的认证请求

1、 创建一个 RadiusClient 类，传入主机名和你想要连接的服务器的共享密钥，你可以用这个类的方法设置其他的属性（如 post numbers）

```
RadiusClient rc = new RadiusClient(host, shared);
```

2、创建一个认证请求的 Radius 报文，在构造方法中传入用户的名字和密码，User-Name 属性将直接被加到对象上，而 User-Password 属性(PAP)或是 CHAP-Password 和 CHAP-Challenge 属性将在打包时生成因为请求认证报文是需要加密的密码。

```
AccessRequest ar = new AccessRequest(user, pass);
ar.setAuthProtocol(AccessRequest.AUTH_CHAP); // or AUTH_PAP
```

3、设置更多的属性。注意 TinyRadius 从给定类型名解析属性类型，而且它将 ip 地址和常量名转换为正确的值。也请注意 Vendor-Specific 的子属性 WISPr-Location 是如何被设置的。

```
ar.addAttribute("NAS-Identifier", "this.is.my.nas-identifier.de");
ar.addAttribute("NAS-IP-Address", "192.168.0.100");
ar.addAttribute("Service-Type", "Login-User");
ar.addAttribute("WISPr-Location-ID", "ger,de.sample-location");
```

4、发送和接收响应

```
RadiusPacket response = rc.authenticate(ar);
if (response.getPacketType() == RadiusPacket.ACCESS_ACCEPT) {
    ..
```

实例 3：如何去实现一个 Radius 服务器

你需要继承 org.tinyradius.util.RadiusServer。为以下的方法提供一个实现：

```
String getSharedSecret(InetAddress client);
```

这个方法会检查是否允许客户机与 Radius 服务器通信。如果允许，它应该返回一个共享密钥来保护与客户机的通信。

```
String getUserPassword(String userName);
```

上面的方法返回了给定用户的密码，如果你不能访问密码(在 CHAP 是这样的情况)或者你需要更好的控制（例如你想设置响应数据报的属性），你必须重写如下的方法：

```
RadiusPacket accessRequestReceived(AccessRequest request, InetAddress client);
RadiusPacket accountingRequestReceived(AccountingRequest request, InetAddress client);
```

重写这两个方法可以更好的控制计费请求和认证请求报文的处理，只要返回一个 Radius 报文作为一个回应，或者如果请求被忽视就返回 null。

实现了你的 Radius 服务器类后，你要以使用 start()和 stop()方法分别开户和结束服务了。对于启动方法，你要传入是否要监听认证端口和计费端口。这个方法会开启新的线程。

```
RadiusServer server = new MyRadiusServer();
server.start(true, true);
server.stop();
```

实例 4：如何实现一个 Radius 代理服务器

你需要继承 org.tinyradius.proxy.RadiusProxy 类，除了实现此类的实现方法外，你还必需提供以下方法的实现。

```
RadiusEndpoint getProxyServer(RadiusPacket packet, RadiusEndpoint client);
```

使用提供的客户端（包含客户端的 IP 地址，端口号和共享密钥），你必须决定给定的 Radius 报文是否应被转发或者是 TinyRadius 服务器自己处理报文

如果你返回 null，报文会像通常一样处理，否则，报文会被代理（添加一个 Proxy-State 属性）发给 Radius 服务器。

作为一个完整的例子，请查看以下三个类：TestProxy, TestServer 和 TestClient，可以建立只含有 TinyRadius 的 Radius 代理，只使用”localhost”

TinyRadius 包中自带的实现

搭建服务器：TestServer

```
/**
 * $Id: TestServer.java,v 1.6 2006/02/17 18:14:54 wuttke Exp $
 * Created on 08.04.2005
 * @author Matthias Wuttke
 * @version $Revision: 1.6 $
 */
package org.tinyradius.test;
```

```
import java.io.IOException;
import java.net.InetSocketAddress;
import org.tinyradius.packet.AccessRequest;
import org.tinyradius.packet.RadiusPacket;
import org.tinyradius.util.RadiusException;
import org.tinyradius.util.RadiusServer;

/**
 * Test server which terminates after 30 s.
 * Knows only the client "localhost" with secret "testing123" and
 * the user "mw" with the password "test".
 */
public class TestServer {
	public static void main(String[] args) throws IOException, Exception {
		RadiusServer server = new RadiusServer() {
			// Authorize localhost/testing123
			public String getSharedSecret(InetSocketAddress client) {
				if (client.getAddress().getHostAddress().equals("127.0.0.1"))
					return "testing123";
				else
					return null;
			}
			// Authenticate mw
			public String getUserPassword(String userName) {
				if (userName.equals("mw"))
					return "test";
				else
					return null;
			}
			// Adds an attribute to the Access-Accept packet
			public RadiusPacket accessRequestReceived(AccessRequest accessRequest,
InetSocketAddress client)
			throws RadiusException {
				System.out.println("Received Access-Request:\n" + accessRequest);
				RadiusPacket packet = super.accessRequestReceived(accessRequest, client);
				if (packet.getPacketType() == RadiusPacket.ACCESS_ACCEPT)
					packet.addAttribute("Reply-Message", "Welcome " +
accessRequest.getUserName() + "!");
				if (packet == null)
					System.out.println("Ignore packet.");
				else
					System.out.println("Answer:\n" + packet);
				return packet;
			}
```

```
            };
            if (args.length >= 1)
                server.setAuthPort(Integer.parseInt(args[0]));
            if (args.length >= 2)
                server.setAcctPort(Integer.parseInt(args[1]));
            server.start(true, true);
            System.out.println("Server started.");
            Thread.sleep(1000*60*30);
            System.out.println("Stop server");
            server.stop();
        }
}
```

以上的类实现了一个简单的 Radius 服务器，该服务器会在 30 分钟后停止服务。只会应答本机的客户端，即 ip 为 127.0.0.1,共享密钥为 testing123,用户名为 mw，密码为 test

重写的第一个方法 getSharedSecret()判断如果客户端的地址为 127.0.0.1 的话就返回共享密钥 testing123，即允许通信，否则返回 null

重写的第二个方法 getUserPassword()判断如果用户名是 mw 就返回密码 test，这是模拟的数据库中查询到该用户的情况，如果数据库中没有该用户，就返回 null。

重写的第三个方法 accessRequestReceived(),首先打印了请求包的内容。然后调用父类中的实现，验证密码的正确与否来决定返回接受包还是拒绝包，该代码如下：

```
/**
     * Constructs an answer for an Access-Request packet. Either this
     * method or isUserAuthenticated should be overriden.
     * @param accessRequest Radius request packet
     * @param client address of Radius client
     * @return response packet or null if no packet shall be sent
     * @exception RadiusException malformed request packet; if this
     * exception is thrown, no answer will be sent
     */
    public RadiusPacket accessRequestReceived(AccessRequest accessRequest, InetSocketAddress
client)
    throws RadiusException {
        String plaintext = getUserPassword(accessRequest.getUserName());
        int type = RadiusPacket.ACCESS_REJECT;
        if (plaintext != null && accessRequest.verifyPassword(plaintext))
            type = RadiusPacket.ACCESS_ACCEPT;

        RadiusPacket answer = new RadiusPacket(type, accessRequest.getPacketIdentifier());
        copyProxyState(accessRequest, answer);
        return answer;
    }
```

之后就是根据返回的包来打印一些提示信息，当报文不为空时打印响应包的内容。最后在主方法中接收输入参数（是否改变默认的监听端口），并开启服务，30 分钟后关闭服务。接下来看一下客户端的代码：TestClient

```
package com.higinet.tinyradius;
/**
 * $Id: TestClient.java,v 1.4 2006/02/17 18:14:54 wuttke Exp $
 * Created on 08.04.2005
 * @author Matthias Wuttke
 * @version $Revision: 1.4 $
 */
import org.tinyradius.packet.AccessRequest;
import org.tinyradius.packet.AccountingRequest;
import org.tinyradius.packet.RadiusPacket;
import org.tinyradius.util.RadiusClient;
/**
 * Simple Radius command-line client.
 */
public class TestClient {

	/**
	 * Radius command line client. <br/>
	 * Usage: TestClient <i>hostName sharedSecret userName password</i>
	 * @param args	arguments
	 * @throws Exception
	 */
	public static void main(String[] args) throws Exception {
		if (args.length != 4) {
			System.out.println("Usage: TestClient hostName sharedSecret userName
password");
			System.exit(1);
		}

		String host = args[0];
		String shared = args[1];
		String user = args[2];
		String pass = args[3];

		RadiusClient rc = new RadiusClient(host, shared);

		// 1. Send Access-Request
		AccessRequest ar = new AccessRequest(user, pass);
		ar.setAuthProtocol(AccessRequest.AUTH_PAP); // or AUTH_CHAP
		ar.addAttribute("NAS-Identifier", "this.is.my.nas-identifier.de");
		ar.addAttribute("NAS-IP-Address", "192.168.0.100");
		ar.addAttribute("Service-Type", "Login-User");
		ar.addAttribute("WISPr-Redirection-URL", "http://www.sourceforge.net/");
		ar.addAttribute("WISPr-Location-ID", "net.sourceforge.ap1");
```

```
            System.out.println("Packet before it is sent\n" + ar + "\n");
            RadiusPacket response = rc.authenticate(ar);
            System.out.println("Packet after it was sent\n" + ar + "\n");
            System.out.println("Response\n" + response + "\n");

            // 2. Send Accounting-Request
            AccountingRequest acc = new AccountingRequest("mw",
            AccountingRequest.ACCT_STATUS_TYPE_START);
            acc.addAttribute("Acct-Session-Id", "1234567890");
            acc.addAttribute("NAS-Identifier", "this.is.my.nas-identifier.de");
            acc.addAttribute("NAS-Port", "0");

            System.out.println(acc + "\n");
            response = rc.account(acc);
            System.out.println("Response: " + response);
            rc.close();
    }
}
```

1、 首先，该类提醒你输入客户端的主机名，共享密钥，用户名与密码。用主机名和共享密钥来创建一个 radius 客户端。

2、 之后构造认证请求报文。设置认证协议是 PAP，也可以是 CHAP。相关的属性名可以参考 RADIUS 协议属性表。

3、 打印发送认证请求之前的报文；发送认证请求；打印发送后的报文，发现多了一个属性: User-Password: 0x7fb44fe71f2aa497c3035c0a1fa15350，这就是在打包时加密生成的；生成的过程在 `encodePapPassword()` 方法中，该方法是将密码加密。然后在 `encodeRequestAttributes()` 方法中将密码属性放进去。在服务器端接收的时候还需要将密码解密：`decodePapPassword()`，解密为纯文本形式。
打印返回的报文：报文类型变是 Access-Accept，多了 Reply-Message: Welcome mw!；再往后是计费请求报文，我们这里不看它。

4、 记住关闭客户端。

如果使用 PAP 方式的话，可以取得用户密码的明文形式

AccessRequest.getUserPassword();

可能会遇到的问题

1、 If RADIUS User password is more than 16 bytes, then user authentication fails，如果用户的密码超过 16 位的话密码的加密将失败，从而导致认证失败，原因在于类：org.tinyradius.packet.AccessRequest 的 `encodePapPassword()` 方法

```
/**
* This method encodes the plaintext user password according to RFC 2865.
* @param userPass the password to encrypt
* @param sharedSecret shared secret
```

```
* @return the byte array containing the encrypted password
*/
   private byte[] encodePapPassword(final byte[] userPass, byte[]
sharedSecret) {
 // the password must be a multiple of 16 bytes and less than or equal
// to 128 bytes. If it isn't a multiple of 16 bytes fill it out with zeroes
// to make it a multiple of 16 bytes. If it is greater than 128 bytes
// truncate it at 128.
       byte[] userPassBytes = null;
       if (userPass.length > 128){
           userPassBytes = new byte[128];
           System.arraycopy(userPass, 0, userPassBytes, 0, 128);
       } else {
           userPassBytes = userPass;
       }

       // declare the byte array to hold the final product
       byte[] encryptedPass = null;
       if (userPassBytes.length < 128) {
           if (userPassBytes.length % 16 == 0) {
               // tt is already a multiple of 16 bytes
               encryptedPass = new byte[userPassBytes.length];
           } else {
               // make it a multiple of 16 bytes
               encryptedPass = new byte[((userPassBytes.length / 16) * 16)
+ 16];
           }
       } else {
           // the encrypted password must be between 16 and 128 bytes
           encryptedPass = new byte[128];
       }

       // copy the userPass into the encrypted pass and then fill it out
with zeroes
       System.arraycopy(userPassBytes, 0, encryptedPass, 0,
userPassBytes.length);
       for (int i = userPassBytes.length; i < encryptedPass.length; i++)
{
           encryptedPass[i] = 0;
       }
       // digest shared secret and authenticator
       MessageDigest md5 = getMd5Digest();
       byte[] lastBlock = new byte[16];
```

```
        for (int i = 0; i < encryptedPass.length; i+=16) {
            md5.reset();
            md5.update(sharedSecret);
            md5.update(i == 0 ? getAuthenticator() : lastBlock);
            byte bn[] = md5.digest();
            System.arraycopy(encryptedPass, i, lastBlock, 0, 16);
            // perform the XOR as specified by RFC 2865.
            for (int j = 0; j < 16; j++)
                encryptedPass[i + j] = (byte)(bn[j] ^ encryptedPass[i +
j]);
        }
        return encryptedPass;
    }
```

解决办法：将 `System.arraycopy(encryptedPass, i, lastBlock, 0, 16);`从 `for` 循环上面移到 `for` 下面。

```
for (int i = 0; i < encryptedPass.length; i+=16) {
      md5.reset();
      md5.update(sharedSecret);
      md5.update(i == 0 ? getAuthenticator() : lastBlock);
      byte bn[] = md5.digest();

      // perform the XOR as specified by RFC 2865.
      for (int j = 0; j < 16; j++)
            encryptedPass[i + j] = (byte)(bn[j] ^ encryptedPass[i + j]);
      System.arraycopy(encryptedPass, i, lastBlock, 0, 16);
/* Problem was with above line. This line is moved from above for loop to below for loop as as to
work correctly. */
}
```

2、 你感觉默认字典中的属性不够用，你可以定义自己的字典。

比如你想添加一个 Name 属性，你可以定义一个字典文件：my_dictionary

我们现在把这个文件放在 src 下面，所以在你添加好你之前的属性后，你可以设置你的新字典然后添加你定义的属性了：

```
InputStream source =
ClassLoader.getSystemResourceAsStream("my_dictionary");
Dictionary dictionary = DictionaryParser.parseDictionary(source);
ar.setDictionary(dictionary);
ar.addAttribute("Name","yanhua");
```

这样添加后你就可以传 Name 属性了，有点注意的是：你的新的字典要把旧字典中的属性全部拷贝过来，这是实验证明的，如果你不拷贝到你的字典中，最后默认字典中的属性将成为 Unknow-Attribute。

虽然这样可以传递了，但是你会发现在服务器端并不能取得 Name 属性，没错，你必须在服务器端也把这个文件放到 src 下面，然后在方法 accessRequestReceived 开始处写入以下的代码：

```
    //导入新的字典
InputStream source = ClassLoader.getSystemResourceAsStream("my_dictionary");
Dictionary dictionary = null ;
try {
        dictionary = DictionaryParser.parseDictionary(source);
} catch (IOException e1) {
    e1.printStackTrace();
}
accessRequest.setDictionary(dictionary);
```

现在你已经能看到你新加的属性了，但是让你恼火的是：你明明传了 Name yanhua，但是打印出来是：

Name: 0x79616e687561

很简单，你只要用下面的方式取就 OK 了：

```
byte[] nameAttr = accessRequest.getAttribute(208).getAttributeData();
String s = null ;
try {
    s = new String(nameAttr,"utf-8");
} catch (UnsupportedEncodingException e) {
    e.printStackTrace();
}
System.out.println("我取出的发来的 Name 是："+s);
```

3、此外，就是使用代理了，这个可以参看源码：TestProxy