UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

"Autenticación de usuarios en redes inalámbricas mediante el uso de un protocolo robusto para el manejo de contraseñas débiles"

T E S I S

QUE PARA OBTENER EL GRADO DE:

MAESTRO EN INGENIERÍA (COMPUTACIÓN)

P R E S E N T A:

JUAN FRANCISCO GARCÍA NAVARRO

DIRECTOR DE TESIS: DR. ENRIQUE DALTABUIT GOODAS

México, D.F. **Noviembre del 2005.**

INTRODUCCIÓN

Hoy en día, el uso de redes inalámbricas va en aumento tanto en instituciones educativas como en empresas privadas. El uso de este tipo de infraestructura requiere la implementación de mecanismos que aseguren la privacidad de la información que se transmite, así como la autenticación de forma segura de los Usuarios que tienen acceso a este tipo de redes. Los procesos involucrados para garantizar la seguridad, son la autenticación, confidencialidad e integridad.

Actualmente, el protocolo WEP (Wired Equivalent Privacy) es el mecanismo más utilizado y mayormente difundido en este tipo de redes, no obstante que han sido documentadas demasiadas vulnerabilidades en él, y es a su vez, el primero en cumplir con el estándar de seguridad 802.11 aprobado por la IEEE, y aunque WPA (Wi-Fi Protected Access) ha venido a reemplazar a WEP, todavía existen una gran cantidad de dispositivos físicos (puntos de acceso) que lo utilizan. WEP es un protocolo para garantizar la confidencialidad de la comunicación entre dos partes (normalmente un Usuario con una tarjeta de red y un Punto de Acceso ambos soportando el protocolo 802.11), el protocolo es descrito con más detalle en el capítulo primero.

El objetivo principal de este trabajo de tesis, es la implementación de un mecanismo (protocolo) de autenticación de Usuarios robusto sobre redes inalámbricas y un Servidor de autenticación basado en el estándar RADIUS que pueda garantizar :

1. Seguridad; esto es corregir las vulnerabilidades del proceso de autenticación mediante el uso de un Servidor RADIUS habitual (mismas que son documentadas en la sección 3.10), así como robustecer los mecanismos de seguridad, específicamente en b que al proceso de autenticación y control de acceso se refiere.
2. Facilidad para su implementación tanto del lado del Usuario como del Servidor de autenticación; esto es que no requiere el uso de certificados digitales, ni de una autoridad certificadora, y únicamente conlleva el uso de la máquina virtual de java.
3. Facilidad de uso para los Usuarios; el protocolo debe ser amigable con los Usuarios, esto es que los mismos no tengan la necesidad de llevar a cabo configuraciones complejas, y únicamente se limiten a proporcionar su nombre de Usuario y su respectiva contraseña.
4. Portabilidad con cualquier dispositivo físico que cumpla con el estándar RADIUS, y no limitarse a tecnología propietaria; la razón principal por la que se toma como base el estándar RADIUS es porque existe una gran cantidad de dispositivos físicos (específicamente Puntos de Acceso) que lo tienen implementado, de hecho casi todos, y prácticamente es la única forma estándar de poder interactuar con todos ellos.

Existen diversos protocolos de autenticación de Usuarios, pero la intención de haber seleccionado éste en especifico, fue para cumplir con los 5 puntos anteriores.

CAPITULO 1.- Redes Inalámbricas

1.1.- Introducción a las redes inalámbricas

Casi al mismo tiempo que aparecieron las computadoras portátiles, muchas personas tuvieron el sueño de andar por la oficina y poder conectar a Internet su computadora. En consecuencia, varios grupos empezaron a trabajar para cumplir con esta meta. El método más práctico es equipar a las computadoras de la oficina y las portátiles con transmisores y receptores de radio de onda corta que les permitan comunicarse. Este trabajo condujo rápidamente a que varias empresas empezaran a comercializar las redes inalámbricas de área local. El problema fue que no había compatibilidad entre ninguna de ellas; esta proliferación implicaba que una computadora equipada con un radio de marca X no funcionara en una estación de base de marca Y. Finalmente, la industria decidió que un estándar de red inalámbrica de área local, sería una buena idea, por lo que el comité del IEEE[1] que estandarizó las redes inalámbricas de área local, se le encargó la tarea de diseñar un estándar. El estándar resultante se le llamó 802.11[2], y la propuesta indica que tiene que trabajar en dos modos, en presencia o ausencia de una estación base también llamada Punto de Acceso.
En el primer caso toda la comunicación se hace a través de la estación base, mientras que en el segundo caso, las computadoras pueden enviarse mensajes entre sí directamente, este método es conocido como red *adhoc*.

Los principales retos que tuvieron que enfrentar el comité de IEEE, fueron : encontrar una banda de frecuencia adecuada, de preferencia mundial; enfrentar el hecho de que las señales de radio tienen un rango finito; asegurarse de que se mantuviera la privacidad de los Usuarios; tomar en cuenta la vida limitada de las baterías; preocuparse por la seguridad humana (las ondas de radio son posibles causantes de cáncer); comprender las implicaciones de la movilidad y, por último, construir un sistema con suficiente ancho de banda para que fuera económicamente viable.

Cuando empezó el proceso de estandarización, Ethernet ya había llegado a dominar las redes de área local, por lo que el comité decidió hacer que el 802.11 fuera compatible con Ethernet sobre la capa de enlace de datos. En particular, se podría enviar un paquete IP sobre la red inalámbrica de área local, del mismo modo en el que una computadora conectada mediante cable enviaba un paquete IP a través de Ethernet; no obstante, existen algunas diferencias inherentes con Ethernet en las capas física y de enlace de datos, y tuvieron que manejarse mediante el estándar.

Primero, una computadora Ethernet siempre escucha el medio antes de transmitir; solo si el medio está inactivo la computadora puede empezar a transmitir; esta idea no funciona igual en las redes inalámbricas de área local, esto se debe a que puede darse el caso en el que la computadora A está transmitiendo a la computadora B, pero el alcance del radio del transmisor de A es muy corto como para encontrar a la computadora C; si C desea transmitir a B entonces puede escuchar el medio antes

[1] IEEE son las siglas de Institute of Electrical and Electronics Engineers, una autoridad profesional técnica sin fines de lucro, con más de 360, 000 miembros individuales en aproximadamente 176 países. Mas información referente puede encontrarse en [4].
[2] El estándar 802.11 puede ser consultado electrónicamente en [5] y [18]

de empezar, pero el hecho de que no escuche nada no quiere decir que su transmisión tendrá éxito, este fue un problema el cual tuvo que ser resuelto por el protocolo 802.11.

La figura 1.1 muestra la descripción del problema:

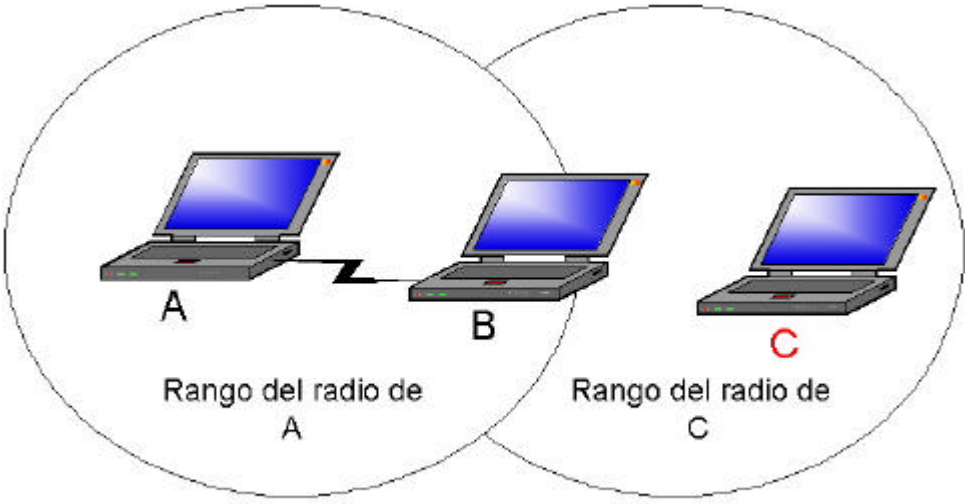


Figura 1.1

El segundo problema por resolver, fue que algunos objetos sólidos pueden reflejar una señal de radio, por lo que esta podría ser recibida múltiples veces (a través de varias rutas); esta interferencia se conoce como desvanecimiento por múltiples trayectorias.
Otro problema, es que una gran cantidad de software no toma en cuenta la movilidad, por ejemplo, muchos procesadores de texto tienen una lista de impresoras de entre las cuales los Usuarios pueden elegir para imprimir un archivo. Cuando la computadora en la que se ejecuta el procesador de texto se coloca en un nuevo entorno, la lista interna de impresoras ya no es útil.

El cuarto problema se presenta cuando una computadora portátil se mueve lejos de la estación base (Punto de Acceso) que está usando, y dentro del rango de una estación base diferente, se requiere algún tipo de manejo. La solución al problema fue incorporar una red prevista de múltiples celdas, cada una con su propia estación base, pero con todas las estaciones base conectadas por Ethernet; desde afuera todo el sistema se vería como una Ethernet sola, la conexión entre el sistema 802.11 y el mundo exterior se conoce como portal.

La figura 1.2 ilustra una red con múltiples celdas.

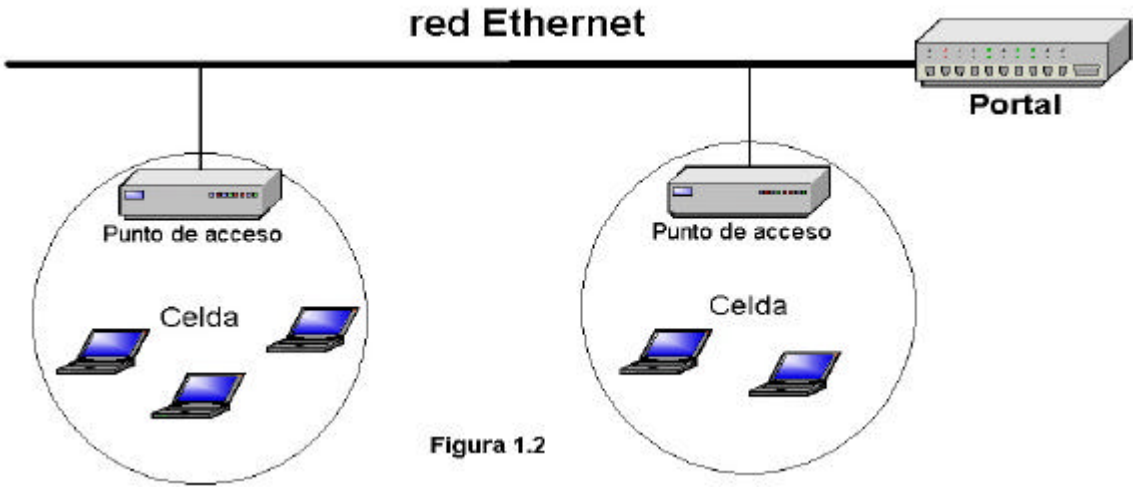


Figura 1.2

Después de algún trabajo, el comité se presentó en 1997 con un estándar que se dirigía a cubrir estos problemas, la red inalámbrica de área local descrita en esa ocasión, se ejecutaba a 1 o 2 Mbps; casi de inmediato la gente comenzó a quejarse de que era demasiado lenta, de manera que empezaron a trabajar en estándares más rápidos. En 1999, concluyeron con dos nuevos estándares, el 802.11a el cual utiliza una banda de frecuencia más ancha y se ejecuta a velocidades de hasta 54 Mbps; y por otro lado, el estándar 802.11b, el cual utiliza la misma frecuencia que el 802.11, pero se vale de una técnica de modulación diferente para alcanzar 11 Mbps.
No obstante con estos dos estándares, el comité 802 ha creado una nueva variante, el 802.11g, que utiliza la técnica de modulación del 802.11a pero la banda de frecuencia del 802.11b.

1.2.- El estándar 802.1X

El estándar 802.1x, define un mecanismo para el control de acceso a redes basado en puertos, que hace uso de las características físicas de la infraestructura IEEE 802 LAN. Provee algunos dispositivos de autenticación y autorización incluidos a un puerto de la red que tiene características de una conexión punto a punto, y previene el acceso a tal puerto en casos en los que la autorización y autenticación fallan. La especificación 802.1x incluye un número de características definidas específicamente para soportar el uso de control de acceso al puerto en las redes IEEE 802.11 LAN's (WLAN). Eso incluye la habilidad para un Punto de Acceso WLAN para distribuir u obtener información

La especificación 802.1x incluye un número de características definidas específicamente para soportar el uso de control de acceso basado en puertos en las redes LAN (WLAN) que cumplan con el estándar IEEE 802.11, esto incluye la capacidad para un Punto de Acceso de distribuir u obtener información de las estaciones asociadas por medio de mensajes del tipo EAPOL.

El incremento en el uso de redes del tipo LAN 802.1x en espacios públicos, ha obligado a los proveedores de este servicio a controlar el acceso a estas redes a los Usuarios permitidos únicamente. Antes de la llegada de 802.1x, si un Usuario era capaz de conectarse a un puerto utilizado por el estándar 802, entonces dicho Usuario tenia completo acceso a la red, y este problema se magnificó con

el rápido crecimiento de las redes WLAN; no obstante, hoy en día cualquier Usuario dentro del rango físico de un Punto de Acceso WLAN puede intentar utilizar los recursos de la red. Debido a esta situación, es necesaria la implementación de un control de acceso a la red basado en puertos, dado que el puerto es el punto de unión entre la red y el Usuario, es el lugar lógico en donde el control de acceso toma lugar, es también un punto ideal para aplicar el filtrado de paquetes y protocolos; de esta forma, controlando los puertos de unión, se puede lograr establecer mecanismos de control de acceso eficaces.

El control de acceso 802.1X basado en puertos, tiene el efecto de crear dos puntos distintos de acceso para proceso de autenticación; un punto de acceso permite el intercambio de marcos entre el sistema y otros sistemas de la red LAN, este puerto no controlado, permite únicamente el intercambio de mensajes de autenticación del tipo EAP. El otro puerto controlado, permite el intercambio de marcos pero únicamente si el puerto es autorizado a hacerlo. Cuando un *host* se conecta al puerto de la red LAN en un *switch* 802.1X, la autenticidad del *host* es determinada por el puerto del *switch* de acuerdo al protocolo especificado por 802.1X, antes de que los servicios ofrecidos por el *switch* sean puestos disponibles en tal puerto. Una vez que el proceso de autenticación ha completado (durante el cual únicamente los marcos EAPOL son permitidos para ser intercambiados), si y solo si únicamente cuando la autenticación haya sido exitosa entonces el puerto es abierto para permitir todo el tráfico. Es claro que 802.1X fue desarrollado para direccionar redes punto-a-punto, en otras palabras, debe existir una relación uno-a-uno entre el Usuario y el Punto de Acceso, análogamente como en una red alambrada, un Usuario esta directamente conectado a un *switch*.

La figura 1.3 ilustra el esquema de los puertos controlado y no controlado en el Punto de Acceso.

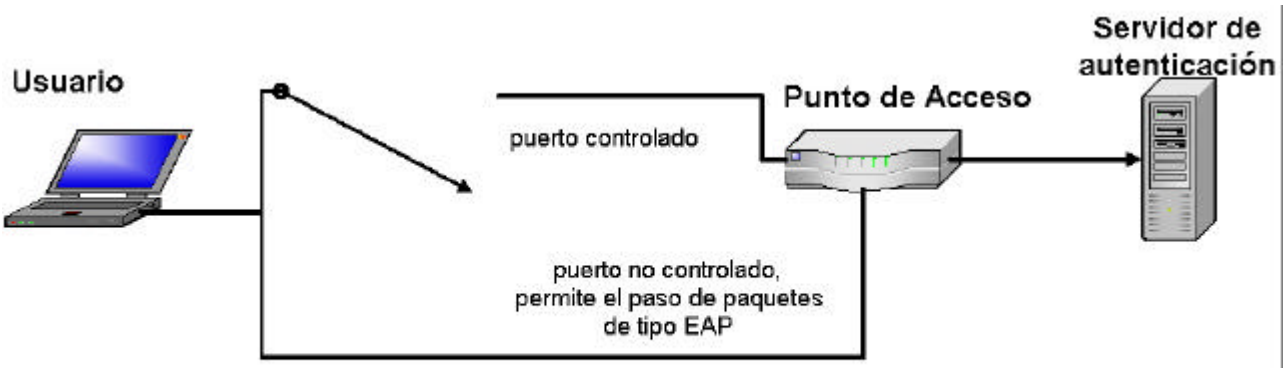


Figura 1.3.- uso de los puertos en 802.1X

802.1X en las redes inalámbricas 802.11

La especificación 802.1X incluye dos características principales diseñadas específicamente para soportar el uso de Control de Acceso basado en puertos en la redes LAN IEEE 802.11.

- Puertos lógicos .- Es la capacidad para hacer uso de la dirección MAC de la estación y el Punto de Acceso como la dirección de destino en el intercambio de paquetes del protocolo EAPOL.
- Administración de llaves.- Es la capacidad de un Punto de Acceso de intercambiar información de llaves con las estaciones incluidas, esto se logra mediante el mensaje EAPOL-Key que va después de una autenticación exitosa.

Proceso de Asociación.- En una red LAN 802.11, las estaciones no están físicamente conectadas a la red, por el contrario múltiples estaciones comparten el mismo medio de acceso a la red (el aire). Un caso especial de acceso a un medio compartido existe en las redes inalámbricas LAN IEEE 802.11, en las cuales una estación (tarjeta de red) debe formar una asociación con un Punto de Acceso para poder hacer uso de la red LAN. El protocolo que establece la asociación, permite a la estación y al Punto de Acceso conocer las direcciones MAC de cada uno de ellos respectivamente. Esto efectivamente crea un puerto lógico que la estación puede usar para comunicarse con el Punto de Acceso. Cuando algún Punto de Acceso es configurado para utilizar Autenticación Abierta, entonces el Usuario esta permitido a asociarse con el Punto de Acceso antes de que las llaves WEP derivadas dinámicamente estén disponibles, una vez que la asociación ha sido establecida, el Usuario puede autenticarse usando EAP.

El proceso de asociación entre un Usuario (Tarjeta de Red Inalámbrica) y un Punto de Acceso, se establece para:

1) Describir el entorno de trabajo arquitectónico dentro del cual la autenticación y la restantes acciones tomarán lugar
2) Definir los principios de operación de los mecanismos de control de acceso
3) Definir los diferentes niveles de control de acceso que son soportados, y el comportamiento del puerto con respecto a la transmisión y recepción de los marcos en cada nivel de control de acceso.
4) Establecer los requerimientos para un protocolo entre el dispositivo que requiere que la autenticación tome lugar (el Punto de Acceso) y el Usuario
5) Establecer los requerimientos para un protocolo de comunicación entre el autenticador y el Servidor de autenticación
6) Establecer los mecanismos y procedimientos que soporta el control de acceso a la red, a través del uso de protocolos de autenticación y autorización
7) Especificar la decodificación de los datagramas utilizados en el protocolo de autenticación y autorización
8) Establecer los requerimientos para administración de control de acceso basado en puertos, identificando los objetos administrados y definiendo las operaciones de administración
9) Especificar la forma en que las operaciones de administración son hechas disponibles a un administrador remoto, usando el protocolo y la descripción de arquitectura provista por el SNMP (*Simple Network Management Protocol*).

Una vez que la estación se ha asociado con el Punto de Acceso, entonces puede comenzar a intercambiar mensajes EAP con el Servidor de Autenticación para autorizar el puerto controlado.

Antes de que el puerto lógico haya sido autorizado, únicamente se permite el intercambio de paquetes EAP.

La figura 1.4 detalla el intercambio de paquetes EAP durante el proceso de asociación.

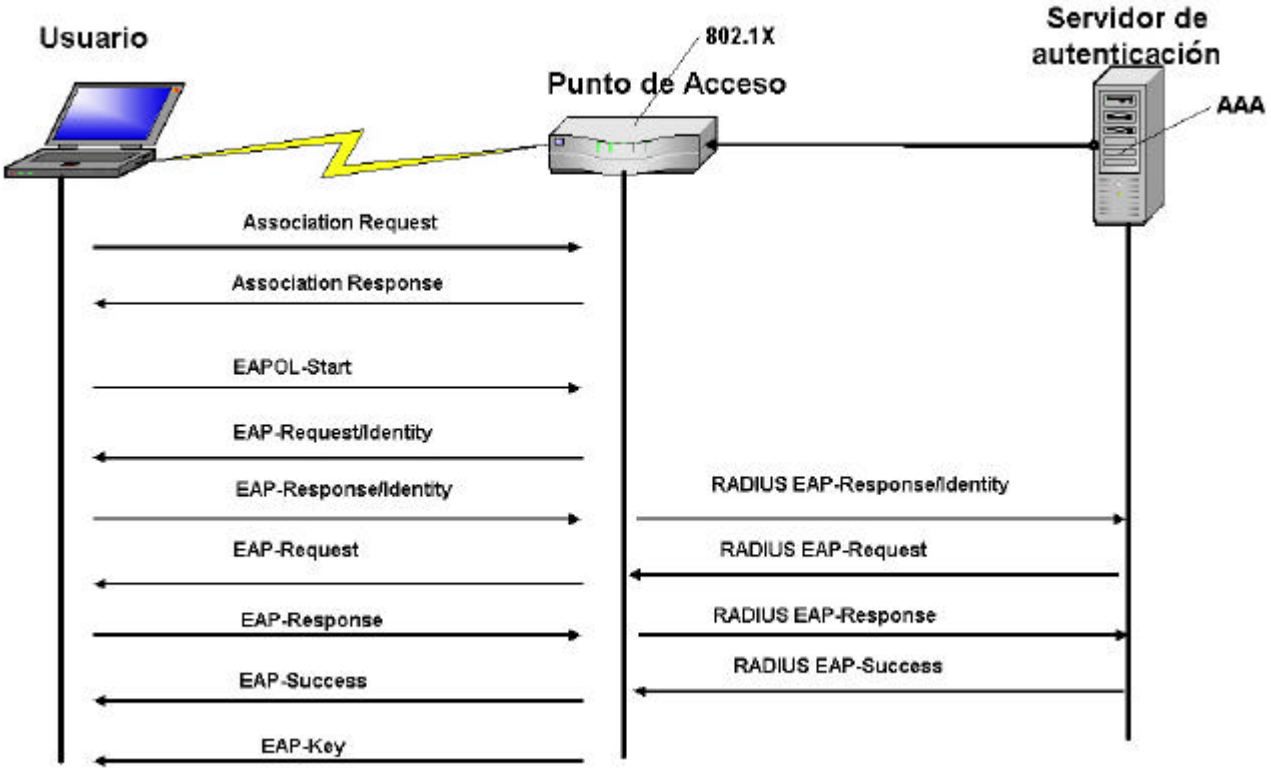


Figura 1.4 Proceso de Asociación

En la figura 1.4 se puede notar que el diálogo EAP entre la estación y el Punto de Acceso es del tipo EAPOL, mientras que la comunicación entre el Punto de Acceso y el Servidor de Autenticación es del tipo RADIUS EAP, lo cual crea una comunicación EAP entre el Servidor de Autenticación y el Usuario.

La definición oficial del protocolo EAP puede ser encontrada en el RFC 2284, la referencia es la [19]

802.11.- Es una familia de especificaciones desarrolladas por el consorcio IEEE para tecnología inalámbrica, aplica a redes inalámbricas de área local (LAN por sus siglas en inglés), y proveen transmisión de 1 o 2 Mbps en un ancho de banda de 2.4 GHz, usando ya sea usando Frecuency Hopping Spread Spectrum (FHSS) ó Direct Sequence Spread Spectrum (DSSS).

802.11a.- Es una extensión del protocolo 802.11 que aplica a redes de área local inalámbricas, y provee hasta 54 Mbps en una banda de 5 GHz . Este protocolo utiliza un esquema de codificación denominado Orthogonal Frequency División Multiplexing (OFDM) en lugar FHSS o DSSS.

802.11b.- Es una extensión del protocolo 802.11 que aplica para redes inalámbricas y provee 11 Mbps de transmisión en una banda de 2.4 GHz, este protocolo utiliza únicamente DSSS. 802.11b fue ratificado como estándar en 1999, permitiendo funcionalidad inalámbrica comparable a Ethernet.
El estándar 802.11b define la capa física y la capa de control de acceso al medio (MAC) para una red de área local sin cables. La capa física utiliza DSSS, que codifica cada bit en un patrón de bit llamado código *chipping*. Como todos utilizan el mismo código, DSSS no es un protocolo de acceso múltiple; es decir, no intenta coordinar el acceso al canal desde múltiples *hosts*, sino que es un mecanismo de la capa física que propaga la energía en una señal sobre un rango de frecuencia amplio, mejorando por ello la capacidad del receptor para recuperar los bits originales transmitidos.

802.11a vs. 802.11b.- Por supuesto, el rendimiento superior del protocolo 802.11a, ofrece excelente soporte para aplicaciones las cuales requieran un ancho de banda bastante amplio, pero la operación de una frecuencia más alta, implica un rango mucho mas corto de señal, en un promedio de 5 a 1, esto mientras un protocolo 802.11a cubre un rango de 60 pies, el 802.11b puede abarcar hasta 300 pies.
Los diferentes tipos de radiofrecuencia y modulación entre 802.11a y 802.11b no permite la interoperación entre ellos, esto es que un Cliente con una tarjeta 802.11a no podrá ser capaz de conectarse a un Punto de Acceso 802.11b.

Cuando utilizar el protocolo 802.11b

- Cuando el rango de los requerimientos es significativo, por ejemplo en un área demasiado grande, en donde este protocolo reduce los costos, debido a que se requiere un número menor de puntos de acceso.
- Cuando haya pocos Usuarios compitiendo por hacer uso de los puntos de acceso, a menos que hubiera considerables necesidades de un alto rendimiento por cada Usuario.

Cuando utilizar el protocolo 802.11a

- Cuando se requiere un alto rendimiento, por ejemplo en aplicaciones que involucran video, voz o la transmisión de grandes imágenes y archivos.
- Cuando haya una considerable presencia de interferencia dentro de la banda de 2.4 GHz . El creciente uso de teléfonos celulares y dispositivos *Bluetooth* pueden saturar el espectro de radio de banda de 2.4 GHz y disminuir considerablemente el rendimiento de las redes con protocolo 802.11b; el uso de 802.11a operando en una banda de 5GHz evitaría esa interferencia.
- Lugares en los cuales existe un gran número de Usuarios, por ejemplo un laboratorio de computadoras, aeropuertos, centros de convenciones, todos ellos compitiendo por los puntos de acceso, este protocolo tiene la capacidad de manejar todas esas peticiones.

El estándar 802.1X puede ser consultado en la referencia [18]

1.3.- El protocolo WEP (Wired Equivalent Privacy)

La especificación *802.11* describe también un protocolo para el cifrado llamado *WEP*, cuyo objetivo principal es hacer que las comunicaciones en redes inalámbricas *WLAN* sean tan seguras como su contraparte *LAN* alambrada, este protocolo es descrito en detalle en [16]. *WEP* provee dos piezas

críticas a la parte de seguridad inalámbrica, autenticación y confidencialidad. *WEP* utiliza un mecanismo de llave secreta con un algoritmo de cifrado simétrico *RC4*.

Autenticación y cifrado con WEP

Cuando una estación se asocia con un Punto de Acceso, la estación debe autenticarse; primeramente, la estación y el Punto de Acceso intercambian información acerca de el tipo de autenticación que ellos llevarán a cabo. Si el tipo de autenticación es especificado como "abierto", entonces realmente no se da ningún tipo de autenticación. El Punto de Acceso y la estación se identifican uno al otro, es entonces cuando se dice que el proceso de asociación esta completo. Cuando el protocolo WEP es activado, el cifrado se realiza de la siguiente forma:

Simbología:

M :	mensaje que se quiere cifrar
c (M) :	es el resultado de aplicar una función de hash al mensaje M
IV :	es el Vector de Inicialización, 3 bytes (24 bits) generados aleatoriamente
k :	es la llave de cifrado
Å **:**	operación XOR a nivel de bits
RC4 (IV, k) :	flujo de bytes generados por el algoritmo RC4, dependiente de la llave y el vector de inicialización
C :	es texto cifrado, resultado de aplicar el algoritmo de cifrado RC4

Primero se calcula el resultado c (M), este resultado se concatena con el mensaje original para formar un texto plano de la forma: **P = { M, c (M) }** , el cual va a ser usado como entrada en la siguiente fase. Notar que c (M) y P no tienen dependencia con la llave k.

En la segunda parte, se cifra el texto plano P utilizando el algoritmo de cifrado RC4, de la siguiente forma: C = P ⊕ RC4 (IV, k); y finalmente se transmite el vector de inicialización (en texto claro), y el mensaje cifrado.

Figura 1.5.- Proceso de cifrado con WEP

Para el proceso de descifrado, simplemente se invierte el proceso de cifrado de la siguiente forma:

P' = C ⊕ RC4 (IV, k)
P' = (P ⊕ RC4 (IV, k)) ⊕ RC4 (IV, k)

$P' = P$

Acto seguido, el receptor verifica de nuevo el valor de aplicar la función de hash al texto plano descifrado, de la siguiente forma: $P' = (M', c')$ y volviendo a calcular $c (M')$, y ambos resultados deben ser iguales $\{ c (M) = c (M') \}$, esto asegura que únicamente los paquetes que tengan coincidencia en su validación serán tomados en cuenta.

Existen dos tipos de implementaciones WEP, la versión clásica definida en el estándar, y una versión aumentada; la única diferencia entre una y otra radica en el tamaño de la llave, mientras que la versión clásica establece que el tamaño de la llave será de 40 bits, la versión aumentada establece 104 bits aunque algunos vendedores aseguran utilizar llaves de cifrado de 128 bits, **obviamente tanto la tarjeta del Usuario como el punto de acceso, tienen que ser configurados con la misma clave y tamaño, es aquí en donde se presenta el mecanismo de autenticación usando WEP, en el conocimiento del secreto compartido entre ambas partes.**

Desventajas en el uso de WEP

Como ya se mencionó con anterioridad, WEP utiliza RC4 como algoritmo de cifrado por flujo; una bien conocida falla de los algoritmos de cifrado por flujo, es que cuando se cifran dos mensajes utilizando el mismo vector de inicialización y la misma llave, entonces importante información puede ser revelada acerca de los dos mensajes, por ejemplo:

Sea $C_1 = P_1 \oplus RC4 (IV, k)$
Sea $C_2 = P_2 \oplus RC4 (IV, k)$
Entonces se tiene que $C_1 \oplus C_2 = [P_1 \oplus RC4 (IV, k)] \oplus [P_2 \oplus RC4 (IV, k)] = P_1 \oplus P_2$

En otras palabras, el resultado de aplicar la operación XOR a los dos textos cifrados nos arroja el resultado de aplicar la misma operación a los dos textos planos, esto nos lleva a una gran cantidad de ataques, en particular cuando alguno de los dos textos planos es conocido, inmediatamente el otro puede ser calculado.

Para resolver este problema, en ocasiones WEP utiliza un vector de inicialización diferente en cada uno de sus paquetes (en la práctica no todos los Puntos de Acceso lo hacen, ya que algunos siempre utilizan el mismo), aún así el Vector de Inicialización siempre es incluido en el paquete como texto en claro; otra desventaja es que el estándar recomienda (no obliga) a cambiar el Vector de Inicialización en cada paquete, pero no establece la forma de hacerlo, y muchos dispositivos lo hacen pero de una forma mucho muy predecible como incrementando en una unidad su valor cada vez.

El estándar 802.11 no especifica el intercambio de llaves periódicamente, lo cual resulta en que la misma llave se utilice por semanas, meses e incluso años, esto permite que un atacante que monitorea una red que utilice WEP, pueda llegar a obtener el secreto compartido en aproximadamente una hora de monitoreo continúo, haciendo fallar el mecanismo de autenticación. En el sitio oficial de los laboratorios RSA, puede ser encontrada una actualización para tratar de hacer más seguro el protocolo WEP.

Las vulnerabilidades de WEP radican principalmente en su relativamente pequeño tamaño de Vector de Inicialización, y al hecho de que las claves permanecen estáticas, *y es necesario hacer notar que estas vulnerabilidades no están relacionadas directamente con el algoritmo de cifrado RC4, como se puede verificar claramente en un reporte técnico localizado en el sitio oficial de los laboratorios RSA www.rsasecurity.com/rsalabs/node.asp?id=2009 y en un artículo denominado* ***"Weaknesses in the Key Scheduling Algorithm of RC4"*** [3].

En las referencias [14] , [17] y [22] se pueden encontrar una detallada descripción de las vulnerabilidades del protocolo WEP y de un gran número de ataques documentados que han tenido éxito sobre él.

Alternativas para implementar seguridad en una red inalámbrica

Opcionalmente, existen otras tecnologías de seguridad para las redes inalámbricas, como las mencionadas en la página del consorcio IEEE http://grouper.ieee.org/groups/802/11/ entre las cuales se cuentan las siguientes:

- WPA™ es un protocolo poderoso basado en el estándar para redes Wi-Fi para asegurar la confidencialidad de la información, provee un fuerte cifrado de datos así como control de acceso y autenticación de Usuarios. El cifrado de WPA utiliza RC4 y proviene del estándar 802.11i , WPA puede ser configurado en dos versiones, WPA Personal el cual protege contra accesos de red no autorizados, y WPA Enterprise, el cual verifica a los Usuarios de la red a través de un Servidor. WPA utiliza llaves de cifrado de 128 bits, y claves de sesión dinámicas para garantizar la seguridad y privacidad de la red. La principal desventaja de WPA es el hecho de que todos los dispositivos físicos tienen que ser actualizados y el WEP reemplazado, pero no forzosamente todos los dispositivos físicos pueden ser modificables. La dirección electrónica oficial del protocolo WPA es www.wi-fi.com/OpenSection/protected_access_archive.asp
- WPA2™ basado en el estándar IEEE 802.11i, provee a los administradores un alto nivel de seguridad que únicamente los Usuarios pueden acceder a la red; implementa el algoritmo de cifrado AES en lugar de RC4 como su antecesor WPA, y puede ser implementado en dos versiones, WPA2 Personal, que protege accesos a la red no autorizados utilizando una contraseña, y el modo WPA2 Enterprise que verifica a los Usuarios de la red a través de un Servidor. De nueva cuenta, la principal desventaja de WPA2 es el hecho de que todos los dispositivos físicos tienen que ser actualizados. Para mayor información, consultar el sitio oficial de WPA2 www.wi-fi.com/OpenSection/protected_access.asp
- Filtrado de direcciones MAC en el punto de acceso; esta tecnología representa una opción completamente insegura, puesto que como se documenta en este trabajo de tesis en la sección 3.10, la falsificación de direcciones IP o MAC es relativamente muy fácil de realizar, sin mencionar las incomodidades de tener que configurar manualmente cada uno de los puntos de acceso de nuestra red, y comprometiendo completamente nuestro sistema en el caso en el que

[3] Articulo publicado en agosto de 2001, elaborado por Fluhrer, Scott, Itsik Mantin y Adi Shamir

uno de los dispositivos fuese sustraído, puesto que ahí se encuentran almacenadas todas las direcciones MAC.

- Servidor de Autenticación RADIUS; esta tecnología estándar, es la más ampliamente difundida en todos los dispositivos físicos en cuanto a control de acceso se refiere, puesto que incluso es utilizada por EAP, WEP, WPA y WPA2 para autenticar Usuarios, y será tomada como base para los trabajos de esta tesis. Aunque como se documenta en la sección 3.10, el uso del Servidor RADIUS es igualmente inseguro, la unión de éste junto con otro mecanismo de autenticación resultan en una opción bastante aceptable en cuanto a seguridad se refiere, a la vez que se sigue manteniendo la base del estándar definido.
- Kerberos es un sistema de autenticación de red basado en distribución de llaves, permitiendo a las entidades que se comunican sobre una red, probar sus identidades al mismo tiempo que provee la capacidad de verificación de la integridad de los datos y secrecia utilizando algoritmos criptográficos como DES. La principal desventaja en su uso, radica en la distribución de las claves es por eso que no todos los dispositivos físicos soportan la autenticación mediante Kerberos.
- Existe una gran variedad de tecnologías propietarias, entre las cuales destacan LEAP (CISCO™), TLS (Microsoft™), con la desventaja que estos mecanismos de autenticación únicamente pueden ser empleados por la infraestructura de CISCO SYSTEMS ™ por un lado, o de Microsoft™ por el otro, y no garantizan ningún tipo de portabilidad entre dispositivos de otros fabricantes.

1.4.- Tipos de implementaciones de Redes Inalámbricas de Área Local

Modo infraestructura.- Un método de comunicación que requiere un Punto de Acceso inalámbrico. El Punto de Acceso controla el cifrado en la red, y puede establecer un "puente" o establecer una "ruta" para el tráfico inalámbrico a una red Ethernet alambrada, o Internet. Los puntos de acceso que actúan como ruteadores pueden también asignar direcciones IP a las diferentes PC's usando servicios DHCP.

Red Ethernet inalámbrica sin Punto de Acceso.- Dos o más computadoras con tarjetas inalámbricas (802.11b) pueden comunicarse entre sí sin un Punto de Acceso. Las tarjetas inalámbricas deben ser puestas en modo 'adhoc' en lugar de modo infraestructura.

Estableciendo un "puente" entre una red inalámbrica 802.11 b y una red Ethernet alambrada.- El Punto de Acceso inalámbrico actúa como puente de red, la PC inalámbrica parece estar en la misma red que las otras PC's tradicionales en la Ethernet .

Esquema con un Punto de Acceso inalámbrico (802.11b) como ruteador.- El Punto de Acceso inalámbrico actúa como ruteador de banda ancha. No todos los puntos de acceso tienen capacidades de ruteo; al igual que otros ruteadores de banda ancha, el Punto de Acceso mantiene el tráfico LAN separado del tráfico de Internet con un *firewall*.

La figura 1.6 ilustra las 4 diferentes implementaciones.

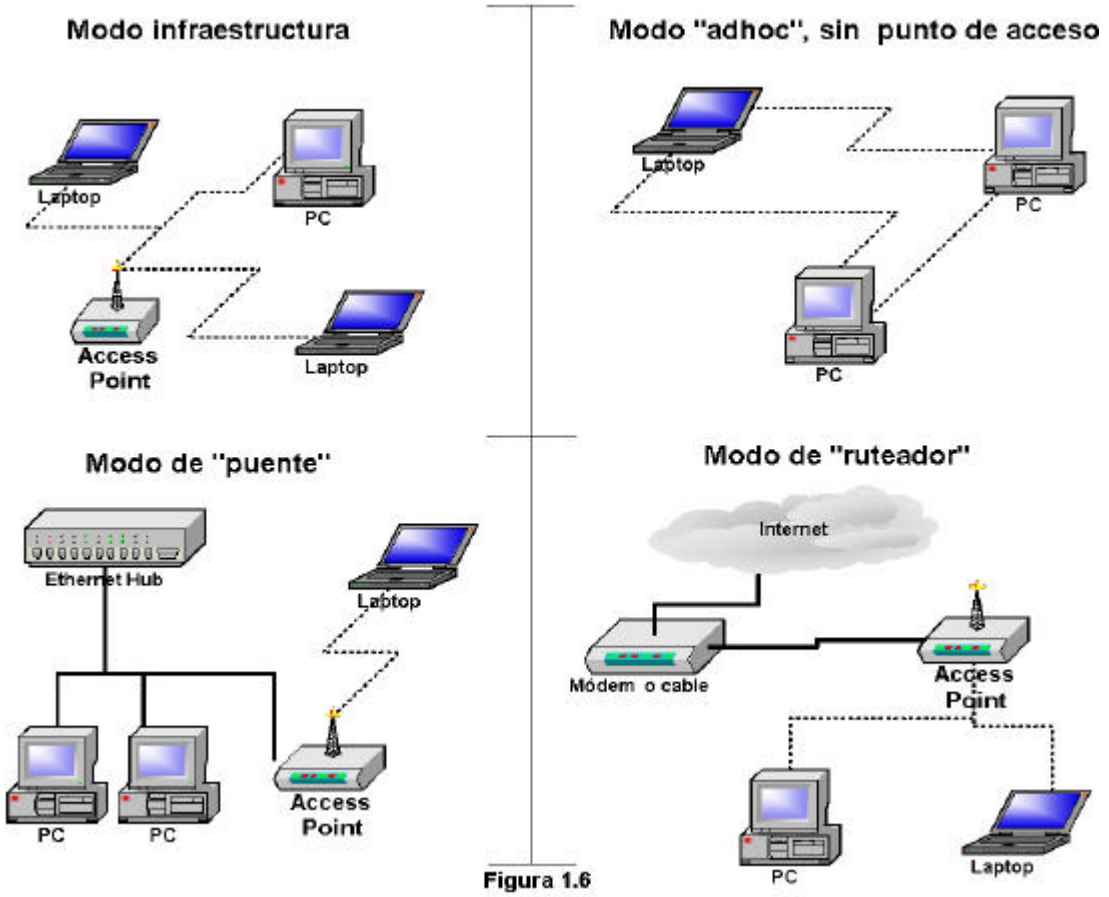

Modo infraestructura
Laptop
PC
Access Point
Laptop
Modo "adhoc", sin punto de acceso
Laptop
PC
PC
Modo de "puente"
Ethernet Hub
Laptop
PC
PC
Access Point
Modo de "ruteador"
Internet
Módem o cable
Access Point
PC
Laptop

Figura 1.6

CAPITULO 2.- Seguridad, Control de Acceso

2.1.- Marco del proceso AAA

El proceso AAA por sus siglas en inglés las cuales significan *Authentication, Authorization* y *Accounting*, -que en español lo podemos identificar como Autenticación, Autorización y Auditoria-, se puede entender respondiendo a estas tres preguntas ¿quién eres tú?, ¿qué servicios tengo permitido otorgarte? y ¿qué hiciste con mis servicios mientras estabas usándolos?. Pero, ¿por qué la arquitectura AAA es una mejor estrategia que otras?; antes que el modelo AAA fuera introducido, diferentes tipos de equipo individual tuvieron que ser utilizados para autenticar Usuarios, pero sin un estándar formal, cada máquina tenia diferentes métodos de autenticación, algunas utilizaban perfiles de Usuario mientras que otras usaban *Challenge Handshake Authentication Protocol* (CHAP, explicado en la sección 3.4), y algunas otras hacían consultas a una base de datos con SQL. El principal problema con este modelo era la escalabilidad porque aumentar la capacidad de una red añadiendo equipo nuevo cada uno de ellos con sus propios métodos de autenticación, rápidamente se convirtió en una pesadilla.

El grupo de trabajo AAA fue formado por IETF[4] para crear una arquitectura funcional que corrigiese las limitaciones del sistema descrito antes, obviamente había una necesidad de enfocarse en equipo descentralizado y en redes heterogéneas. Los proveedores de servicios de Internet comenzaron a ofrecer diversos servicios tales como ISDN, conectividad por cable-módem etc., por lo cual necesitaban una forma estándar con la cual los Usuarios pudieran accesar, ser verificados y monitoreados a través de la red.

La arquitectura AAA está diseñada para trabajar en ambientes con requerimientos de Usuarios variados e igual variedad de diseños de red. El modelo AAA depende de la interacción Cliente/Servidor, en la cual un sistema Cliente solicita los servicios o recursos de un sistema Servidor, y ambos roles están completamente definidos. Los ambientes Cliente/Servidor permiten que los Servidores sean distribuidos y descentralizados a través de la red, esto contrasta con el modelo de red punto a punto, en el cual todos lo sistemas despliegan características tanto de Cliente como de Servidor.

Una capacidad de Servidor proxy es una ligera variación a este esquema, un Servidor AAA puede ser configurado para autorizar una petición o retransmitirla a algún otro Servidor AAA el cual entonces tomará la decisión ya sea de pasarla a su vez a otro o atenderla, formando cadenas proxy, en éste esquema es fundamental revisar que no se formen “ciclos”.

Los Clientes solicitando servicios y recursos de un Servidor AAA pueden comunicarse entre sí ya sea utilizando una transacción hop-to-hop o end-to-end; la distinción entre estos dos modelos radica en el lugar en el cual la relación de confianza se ubica dentro de la cadena de transacción. En una transacción hop-to-hop, un Cliente hace una petición inicial a un Servidor AAA, en este punto existe

[4] IETF son las siglas de Internet Engineering Task Force, una comunidad internacional de diseñadores de red, operadores, vendedores e investigadores involucrados con la evolución de la arquitectura Internet, así como su operación, para más información sobre la comunidad IETF consultar [6].

una relación de confianza entre el Cliente y este Servidor; dicho Servidor determina que la petición tiene que ser reenviada a otro Servidor en una ubicación diferente, por lo cual él actúa como un proxy y reenvía la petición a otro Servidor AAA; ahora la relación de confianza se establece entre los dos Servidores AAA , con la primer máquina actuando como Cliente y la segunda actuando como Servidor, es importante destacar el hecho de que la relación de confianza no es inherentemente transitiva, esto significa que el Cliente inicial y la segunda máquina AAA no establecen una relación de confianza.

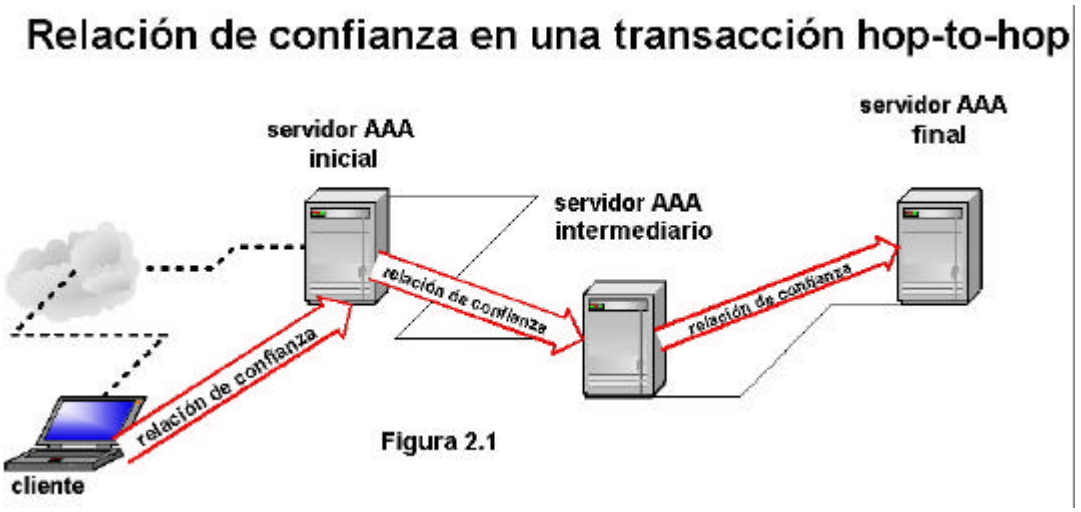


Figura 2.1

El modelo de transacciones end-to-end difiere del anterior en que la relación de confianza se da entre el Cliente y el Servidor AAA final que autoriza la petición, mientras que la cadena proxy se vuelve únicamente un paso de la petición de un Servidor a otro. Debido a su pobre diseño para transmitir información sensible en las peticiones proxy, algunos métodos para validación de la integridad de datos son necesarios, en donde los certificados digitales son muy usualmente utilizados para estos casos.

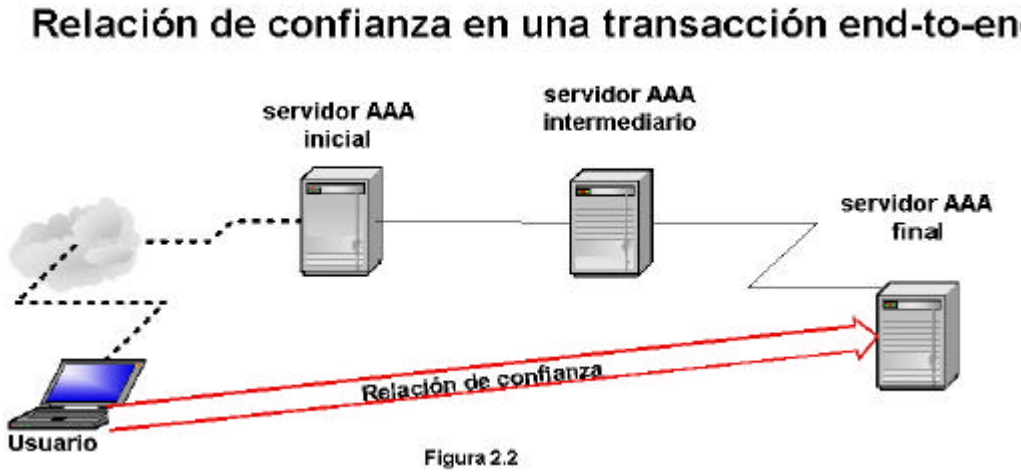


Figura 2.2

2.2.- Autenticación

La autenticación es el proceso de verificar la identidad de una persona o máquina. Las formas más comunes de autenticación utilizan una combinación de un identificador y una contraseña, en la cual el conocimiento de la contraseña es una representación de que el Usuario es auténtico; otra solución que está tomando mucho auge en los últimos años y se convertirá seguramente en el autenticador preferido en transacciones en Internet debido a su seguridad, es el uso de infraestructura de llave pública para la emisión de certificados digitales, desgraciadamente el uso de este tipo de infraestructura es un poco compleja para los Usuarios.

El proceso de identificación comienza con el registro de un Usuario en el sistema, este registro esta formado por un autenticador del Usuario, el cual puede ser un nombre, un número (como en las cuentas bancarias o los pasaportes) o cualquier otro tipo de información que pueda distinguir de forma eficiente y única a un Usuario de los demás, este autenticador es almacenado para su posterior consulta.

El proceso de autenticación comienza cuando el Usuario que intenta ser autenticado, provee al sistema con el autenticador que fue registrado identificando al portador como la persona que tiene permitido utilizar los recursos de dicho sistema, para lo cual el sistema tiene que realizar una comparación del autenticador proporcionado con el registrado, resultando en el otorgamiento de acceso siempre y cuando se pueda corroborar que ambos autenticadores son iguales en cuanto a sus características.

Los mecanismos de autenticación utilizados en la actualidad, se clasifican en cuatro grupos:

- ✓ Algo que el Usuario conoce (autenticación basada en conocimiento)
- ✓ Algo que el Usuario tiene (autenticación basada en posesión)
- ✓ Algo que caracteriza al Usuario (autenticación biométrica)
- ✓ Algo que determina su posición sobre la tierra (autenticación por posicionamiento)

Los mecanismos más seguros son aquellos que emplean al menos dos autenticadores de dos grupos diferentes.

La autenticación basada en conocimientos, se basa en el hecho de que existe un secreto compartido entre el Usuario y el sistema en cuestión, típicamente se trata de una contraseña *–una secuencia de caracteres alfanuméricos-* y el proceso de autenticación basada en conocimientos, comienza cuando un Usuario se identifica al solicitar acceso al sistema, exhibiendo su contraseña, la cual está registrada en el archivo de Usuarios del sistema. Puesto que se trata de un secreto compartido, hay que proteger este secreto en ambos extremos de la trayectoria, tanto en el archivo de Usuarios como en la memoria de la persona, también hay que proteger el secreto durante el tiempo en que éste viaja desde el Usuario hasta el sistema, puesto que si alguien ajeno al sistema llega a conocer el secreto compartido, puede suplantar la identidad del Usuario y tener acceso al sistema en lugar del verdadero dueño, para este tipo de ataque no hay una defensa posible, lo único que se puede hacer es tratar de concientizar al Usuario de memorizar y no compartir con nadie el secreto compartido. También existe el riesgo de que el suplantador obtenga la contraseña directamente desde el archivo en donde son almacenadas dentro del sistema, haciéndolo de igual forma vulnerable; en este caso, una solución posible es no

almacenar las contraseña en texto claro, sino en forma cifrada. De igual forma existe el riesgo de que el suplantador obtenga la contraseña cuando esta viaja en su tránsito hacia el sistema.

La autenticación basada en posesión es desarrollada por medio de dispositivos físicos, los cuales contienen información relativa al dueño o a la actividad que pretende realizar. Esta información aparte de servir como autenticador, puede ser utilizada para otros fines como historias clínicas, información bancaria, efectivo electrónico etc. La principal desventaja de este tipo de autenticación, es el hecho de que lo que realmente se autentica es el dispositivo, el cual puede estar en manos de algún suplantador. Otro inconveniente es el hecho de que en todos los casos se requiere de un dispositivo de lectura capaz de obtener la información contenida en el autenticador.

La autenticación biométrica, basada en características físicas del Usuario, las cuales sean difíciles de modificar y obviamente deben ser distintas de una persona a otra, también deben ser expresables matemáticamente en forma sintética y su visibilidad debe ser independiente del atuendo de las personas. La biometría se define como el uso automatizado de las características fisiológicas o conductuales de una persona para determinar o verificar su identidad. La biometría fisiológica esta basada en medidas o datos de partes del cuerpo humano, mientras que la biometría conductual se basa en la medida o datos de acciones de una persona, e indirectamente en sus características físicas. El proceso de autenticación comienza cuando el Usuario exhibe la característica registrada, el sistema calcula la descripción matemática y la compara con aquella almacenada durante el proceso de registro.

2.3.- Autorización

La autorización involucra utilizar un conjunto de reglas o tablas para decidir lo que un Usuario autenticado puede hacer en un sistema, esas reglas son definidas por el administrador del sistema. Los Servidores AAA contienen lógica que analizará una petición y resolverá otorgar o negar el acceso al recurso solicitado dependiendo si el Usuario tiene o no permitido dicho recurso.

2.4.- Auditoria

La Auditoria establece una medida y documenta los recursos que un Usuario utiliza durante una sesión, esto puede incluir la cantidad de tiempo de sistema, o la cantidad de datos que el Usuario ha enviado y/o recibido durante una sesión. Esta información estadística es usada para control de autorización, pagos de facturas, utilización de recursos etc, un administrador podría analizar las peticiones exitosas para determinar la capacidad y predecir el comportamiento futuro del sistema, un analizador de seguridad podría revisar las peticiones de acceso rechazadas para detectar algunos intrusos que hayan intentado tener acceso al sistema.

2.5.- Secuencias de autorización

Existen varios métodos con los cuales el Usuario final, el Servidor AAA y el equipo de red se comunican durante la transacción, específicamente, hay 3 secuencias diferentes en las cuales cada máquina es conectada.

a) La secuencia "AGENTE"

En esta secuencia, el Servidor AAA actúa como una persona "enmedio" entre el equipo y el Usuario final. El Usuario final inicialmente contacta al Servidor de autenticación AAA, el cual autoriza la petición del Usuario y envía un mensaje al equipo de servicio notificándole de proveer tal servicio. El equipo de servicio lo hace y notifica al Servidor AAA y la notificación es pasada al Usuario final, quien entonces comienza a hacer uso de la red. Esta secuencia es comúnmente utilizada en aplicaciones de banda ancha, en las cuales la calidad del servicio es parte de un contrato existente.

b) La secuencia "PULL"

En esta secuencia, el Usuario final se conecta directamente con el equipo de servicio, el cual revisa con un Servidor AAA para determinar otorgar o negar el uso del servicio, y este Servidor notifica al equipo de servicio su decisión para conectar o desconectar al Usuario de la red. Este tipo de secuencia de autenticación será el utilizado en el desarrollo del proyecto de tesis.

c) La secuencia "PUSH"

La secuencia PUSH altera la relación de confianza entre todas las máquinas en una transacción. El Usuario se conecta al Servidor AAA primero, y cuando la petición al Servidor es autorizada, el Servidor AAA distribuye algún recibo de autenticación, (un certificado digital por ejemplo) al Usuario final. El Usuario final entonces incluye ese recibo en su petición de acceso al equipo de servicio, el cual indica que la autorización ha sido otorgada; la principal diferencia es que el Usuario actúa como el agente entre el Servidor AAA y el equipo de servicio.

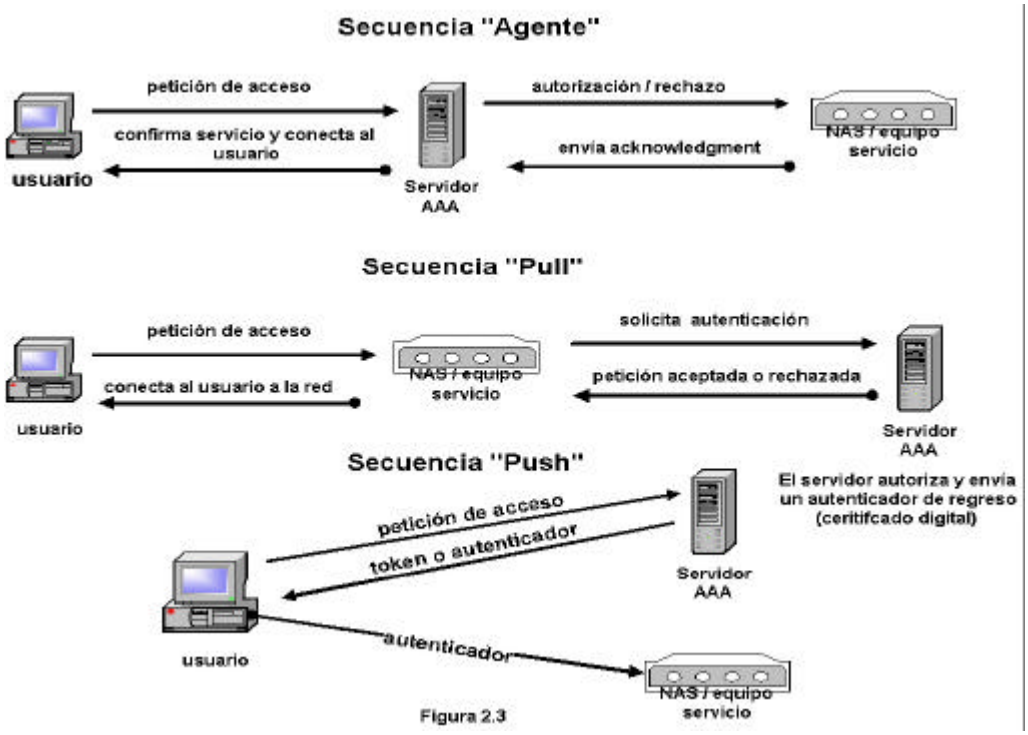


Figura 2.3

2.6.- Políticas de seguridad en un Servidor AAA

Las políticas de seguridad son las especificaciones utilizadas por un Servidor AAA para determinar si una petición de Usuario es válida y el servicio solicitado es otorgado. Cualquier Servidor AAA debe tener una forma de almacenar y recuperar la información referente a las políticas, dichas políticas son depositadas en un contenedor, el cual puede ser prácticamente cualquier dispositivo que almacena información como una base de datos o archivos de texto. El marco de trabajo AAA contiene un conjunto de políticas, el cual debe cumplir 3 tareas específicas, estas deben poder ser recuperadas en cualquier momento, evaluadas y cumplidas, la forma en que esto es realizado, depende del ambiente –sistema operativo y/o arquitectura-.

La administración de recursos es un componente del marco de autorización; el administrador de recursos es básicamente un programa o utilidad que tiene la capacidad de monitorear los recursos que han sido previamente otorgados, este programa debe ser capaz de desplegar y recibir información de los recursos en "tiempo real".

La administración de sesiones es la capacidad de un protocolo para notificar a un Servidor AAA de algún cambio en las condiciones de una sesión existente, dichas modificaciones pueden incluir incluso el término de la sesión.

La combinación de administración de recursos y sesiones permite que complicadas políticas sean implementadas y por consiguiente un control de acceso más eficiente.

Un sistema deberá ser siempre acotado en sus componentes y Usuarios, esto es que debe tener la capacidad de contar con un registro detallado de ambas partes. Cualquier sistema que pretenda hacer uso de mecanismos de identificación y autenticación, requiere de un procedimiento de registro de los componentes y Usuarios del mismo, esto es básicamente un inventario. El sistema se considera acotado a los elementos que aparezcan en este inventario, y aquellos elementos que no lo hagan por definición no forman parte del mismo.

CAPITULO 3.- Remote Authentication Dial In User Service (RADIUS)

3.1.- Introducción a RADIUS

La administración de grandes números de Usuarios y bancos de módems, pueden crear la necesidad de un soporte de administración bastante significativo. Dado que los bancos de módems son por definición una liga al mundo exterior, requieren cuidadosa atención en cuanto a seguridad, autorización y seguimiento de sesiones se refiere; esto puede ser alcanzado administrando una base de datos de Usuarios, lo cual permite autenticación (verificar el nombre de Usuario y la contraseña) así como información de la configuración detallando el tipo de servicio a suministrar al Usuario.

RADIUS es un servidor para la Autenticación remota de Usuarios y la Auditoría, su principal uso es con los proveedores de servicios de Internet (ISP), aunque también puede ser utilizado en cualquier red que necesite manejar un esquema de Autenticación centralizado para sus Estaciones de Trabajo, (que es el caso de las Redes Inalámbricas en específico y del trabajo de este proyecto de Tesis).

Principales características de RADIUS

- ✓ **Modelo Cliente-Servidor.-** Un Servidor de acceso a red (NAS), opera como un Cliente de RADIUS; el Cliente es responsable de pasar la información del Usuario al Servidor RADIUS destinado. Los Servidores RADIUS son responsables de recibir las peticiones de conexión del Usuario, autenticación, y regresar toda la información de configuración necesaria para que el Cliente otorgue el servicio al Usuario. Un Servidor RADIUS puede actuar como un Cliente proxy a otros Servidores RADIUS u otros tipos de Servidores de autenticación.
- ✓ **Seguridad de red.-** Las transacciones entre el Cliente y el Servidor RADIUS son autenticadas a través del uso de un secreto compartido, el cual nunca es enviado sobre la red, únicamente se registra en ambos lados; ese secreto compartido es utilizado para autenticar todos los mensajes intercambiados entre el Servidor RADIUS y el NAS, utilizado como la llave de la función hash MD5 de todo el paquete. En adición, todas las contraseñas de Usuario son enviadas cifradas a través de la red entre el Cliente y el Servidor, para eliminar la posibilidad de que algún extraño que se introMeta en una red insegura, pudiera determinar el password de un Usuario.
- ✓ **Mecanismos de autenticación flexible.-** El Servidor RADIUS puede soportar una variedad de métodos para autenticar a un Usuario. Los mecanismos soportados por RADIUS son PPP, PAP o CHAP, login de Unix y PAM, no obstante el mecanismo utilizado en las Redes Inalámbricas es la autenticación mediante la Dirección MAC de la NIC.
- ✓ **Protocolo extensible.-** Nuevos valores de atributos pueden ser añadidos, sin modificar las implementaciones existentes del protocolo.

3.2.- Operación de RADIUS

Cuando un Cliente (Punto de Acceso) es configurado para utilizar RADIUS, cualquier Usuario registrado en el Cliente tiene que presentar sus credenciales de autenticación a solicitud del Punto de Acceso, esto representa el intercambio de paquetes con la información necesaria para llevar a cabo el proceso de Autenticación.

Una vez que el Cliente ha obtenido tal información del Usuario, y si esta configurado para autenticar utilizando un servidor RADIUS, el Cliente comienza el proceso de Autenticación (mostrado en la figura 3.1), intercambiando paquetes con el servidor RADIUS, la sección 3.7 detalla la estructura, los tipos de paquetes y los atributos RADIUS en cada uno de ellos.

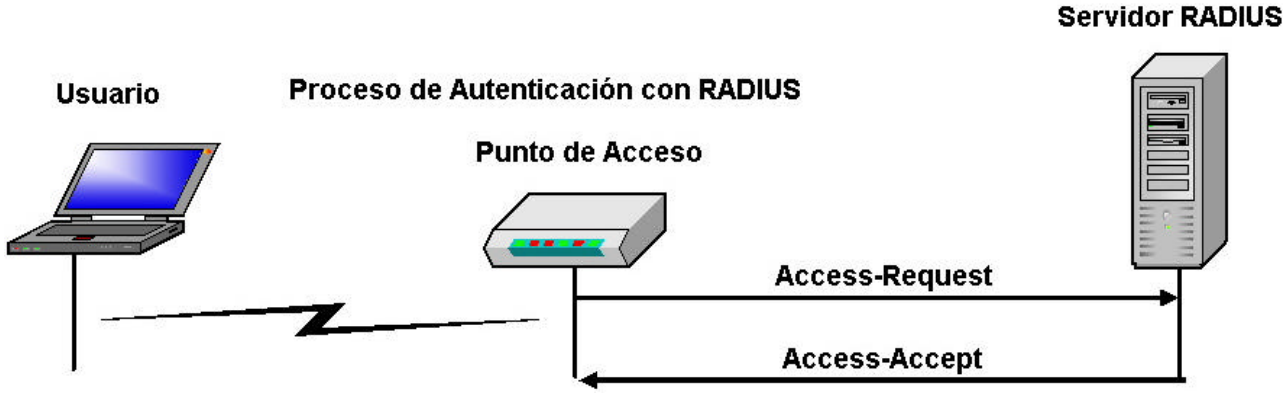


Figura 3.1

Primero, el Cliente envía un paquete de Petición de Acceso (Access-Request) al servidor RADIUS, a través de la red alambrada; una vez que el Servidor RADIUS recibe la petición, valida primero al Cliente que la envía, una petición de un Cliente para el cual el Servidor RADIUS no tiene un secreto compartido debe ser completamente descartada. Si el Cliente es válido, el Servidor RADIUS consulta una base de datos de Usuarios, para encontrar aquel Usuario cuyo nombre coincida con el de la petición; la entrada del Usuario en la base de datos contiene una lista de requerimientos los cuales deben ser conocidos previamente para permitir acceso al Usuario; esta lista puede especificar el Cliente y / o puerto al cual el Usuario tiene permitido accesar.

Si alguna condición no es satisfecha en su totalidad, el Servidor RADIUS envía un paquete de negación del acceso "Access-Reject", indicando que la petición del Usuario ha sido rechazada, si por el contrario, todas las condiciones son satisfechas, el paquete de regreso será del tipo Access-Accept indicando al Cliente que el Usuario si tiene permitido accesar. Otro posible paquete de respuesta es el denominado Access-Challenge, el cual se detalla en la sección 3.3.

Debido a que RADIUS intercambia paquetes UDP (Protocolo no orientado a conexión), siempre existe el riesgo de que los paquetes no lleguen a su destino, para poder manejar esta situación, los Clientes pueden ser configurados para retransmitir un paquete un cierto número de veces, cuando pasado cierto tiempo no se ha obtenido la respuesta (Timer retransmission), también existe la posibilidad de configurar el Cliente para enviar peticiones a un Servidor RADIUS alterno, en caso de que el Servidor primario no responda. Así mismo, el Servidor RADIUS puede enviar paquetes de autenticación a otros Servidores para satisfacer la petición, caso en el cual actuaría como un Cliente en modo proxy.

3.3.- Autenticación Challenge-Response o CHAP

En RADIUS existen varios mecanismos de autenticación que pueden ser implementados, destacando PAP y CHAP.

En el mecanismo de autenticación CHAP, al Usuario se le da un número impredecible generado aleatoriamente, y se le pide que lo cifre utilizando como llave de cifrado el secreto compartido con el servidor RADIUS y que regrese el resultado.

El mecanismo de autenticación funciona de la siguiente forma:

Definiciones:

- Sea A el usuario que pretende autenticarse, y **B** el servidor de autenticación
- Sea n el número generado aleatoriamente
- Sea k la llave de cifrado y denotamos C_k (x) como el resultado de cifrar utilizando como llave de cifrado k el texto x.

1) **B** envía a **A** el número n en texto claro
2) **A** Cifra ese número con la llave de cifrado k y envía de regreso el resultado cifrado a **B** esto es C_k (n)
3) **B** Descifra el mensaje con la llave del usuario representando el secreto compartido entre ellos o la contraseña del usuario.

Desventajas de este mecanismo de autenticación:

- Un atacante puede siempre estar "escuchando" el medio e interceptando los paquetes transmitidos, por lo que puede capturar por un lado el número aleatorio n, el cual viaja en texto claro, y por otro el resultado de cifrar ese valor usando la llave k C_k (n). Con esta información, el usuario puede aplicar un ataque de fuerza bruta para obtener el valor de la llave k.
- Únicamente se autentica el Usuario en el Servidor de Autenticación y no viceversa.

Es importante resaltar el hecho de que la contraseña del Usuario nunca viaja por la red, a diferencia del mecanismo PAP (explicado en la siguiente sección), esto conlleva una ventaja ligera sobre su contraparte.

3.4.- Autenticación con PAP

Para el caso de PAP, el mecanismo consiste en enviar la contraseña del usuario pero "escondida" en el atributo User-Password de un paquete de petición (Access-Request) enviado al Servidor de Autenticación.

Definiciones:

- Sea m el secreto compartido entre el NAS y el Servidor de Autenticación, utilizado para autenticar todos los paquetes intercambiados entre ellos.
- Sea p la contraseña del Usuario, utilizada para autenticar al Usuario en el sistema, la cual viaja en claro en el trayecto entre el usuario y el dispositivo NAS.
- Sea id el identificador de la transacción, generalmente es un número entero secuencial otorgado por el NAS para llevar un control de las operaciones.
- El atributo User-Password (descrito en la sección 3.9, referente a los atributos de los paquetes RADIUS)
- Definimos MD5 (m) como el resultado de aplicar el algoritmo MD5 para calcular la función de hash al mensaje 'm'.
- Definimos el símbolo XOR para la indicar la operación del mismo nombre

El resultado [MD5 (id + m) XOR p] es puesto en el campo User-Password, y enviado en un paquete de petición de acceso (Access-Request).

En otras palabras, el Cliente utiliza el identificador y el secreto compartido y los envía en una secuencia a una función hash calculada con MD5. La contraseña original del Cliente es pasada a través de un proceso de operación XOR con el resultado de la función de hash previamente calculada, el resultado de esas dos secuencias es puesto en el campo User-Password; el Servidor RADIUS entonces revierte el proceso para determinar si autoriza esa conexión. El objetivo principal de este mecanismo de autenticación es el de poder identificar cuando el proceso de autenticación falla, si la falla fue causada por un password incorrecto o por un secreto inválido.

3.5.- RADIUS como proxy

Cuando un Servidor RADIUS recibe una petición de autenticación de un Cliente (un NAS por ejemplo), reenvía esa petición a otro Servidor remoto RADIUS, espera la respuesta del Servidor remoto y la envía de regreso al Cliente, posiblemente con algunos cambios ligeros que reflejan el proceso. Un uso muy común del *proxy* RADIUS es lo que se conoce como *roaming*, esto es cuando un Usuario intenta conectarse con un Cliente con el que no esta autorizado, entonces el Servidor RADIUS puede entablar comunicación con otro Servidor RADIUS y solicitarle que verifique la identidad y autentique a dicho Usuario, de esta forma el Usuario puede repetir el procedimiento con cualquier otro Cliente y Servidor RADIUS que pueda autenticar remotamente con el Servidor en el cual el Usuario está registrado.

El NAS envía un paquete de petición de acceso al Servidor RADIUS "retransmisor", el cual lo reenvía al Servidor remoto encargado de atender esa petición y autenticar al Usuario, y éste último regresa un paquete de aceptación, rechazo o reto (Access-Accept, Access-Reject o Access-Challenge) al Servidor retransmisor, el cual a su vez lo envía al Cliente (NAS).

Un Servidor RADIUS puede trabajar tanto como un Servidor retransmisor o un Servidor remoto, dependiendo del entorno del que se trate, y puede retransmitir a otros Servidores retransmisores formando una cadena de proxies, la única restricción es el cuidado de no formar ciclos.

Esta es la serie de pasos a seguir detallando cada uno de ellos:

1.- Un NAS envía su paquete de petición de acceso al Servidor retransmisor, el cual descifra el campo User-Password si esta presente, usando el secreto compartido que él y el NAS comparten; si el atributo CHAP-Password está presente en el paquete, y no hay atributo CHAP-Challenge, el Servidor retransmisor debe dejar sin modificar el autenticador o copiarlo a un atributo CHAP-Challenge. El Servidor retransmisor puede agregar un atributo Proxy-State al paquete, pero no debe modificar ningún otro atributo del paquete ni el orden de los mismos.

2.- El Servidor retransmisor cifra el atributo User-Password utilizando el secreto que comparte con el Servidor remoto, y reenvía el paquete de petición de acceso al Servidor remoto.

3.- El Servidor remoto verifica y valida la información del Usuario y regresa al Servidor retransmisor un paquete de aceptación, rechazo o reto (Access-Accept, Access-Reject o Access-Challenge), el Servidor remoto debe copiar todos los atributos Proxy-State en el paquete de petición de acceso.

4.- El Servidor retransmisor verifica el autenticador de respuesta usando el secreto que comparte con el Servidor remoto, y descarta el paquete si la verificación falla. Si el paquete pasa la verificación, el Servidor retransmisor elimina el último atributo Proxy-State, "firma" el autenticador de respuesta usando el secreto que comparte con el NAS, restaura el identificador para que coincida con el de la petición original del NAS y envía el paquete de aceptación al NAS.

3.6.- Protocolo de transporte UDP y RADIUS

¿Por qué RADIUS utiliza UDP como protocolo de transporte y no TCP?.- La respuesta es únicamente por cuestiones técnicas, y hay una serie de problemas que tienen que ser entendidos, antes que nada, RADIUS es un protocolo basado en transacciones, el cual tiene las siguientes características:

1. Si la petición a un Servidor de autenticación primario es fallida, un segundo Servidor de autenticación puede ser consultado. Para cumplir con este requerimiento, una copia de esta petición debe ser mantenida arriba de la capa de transporte, para permitir una transmisión alterna.
2. Los requerimientos de tiempo de este protocolo particular, son diferentes de lo que TCP provee. RADIUS no requiere una detección de responsiva de pérdida de datos, porque el Usuario no tiene inconveniente en esperar algunos segundos a que el proceso sea completado, y por otro lado el Usuario no esta de acuerdo en esperar varios minutos para que el proceso de autenticación complete (ejemplo de la entrega confiable de datos dos minutos después en TCP), en RADIUS eso es innecesario debido a que el uso de un Servidor alternativo permite al Usuario tener acceso de una forma más rápida.
3. La naturaleza "sin estado" de este protocolo, simplifica el uso de UDP; los Clientes y Servidores vienen y van, los sistemas son apagados y encendidos constantemente, generalmente esto no causa ningún problema en UDP, ya que cada Cliente y Servidor puede

abrir su transporte UDP solo una vez, y dejarlo abierto a pesar de todos los tipos de fallas y eventos en la red.

4. UDP simplifica la implementación del Servidor.- Las primeras implementaciones de RADIUS consistían de un Servidor monohilo, lo cual ocasionaba que el Servidor fuera incapaz de atender muchas peticiones, y cuando aplicaciones requerían autenticar algunos miles de Usuarios, esto se convirtió en un problema muy serio y muy difícil de manejar; la solución obvia fue la implementación multihilo en el lado del Servidor para atender dichas peticiones, el uso de paquetes UDP facilitó en gran forma este proceso.

Pero obviamente no todas son ventajas, el uso de datagramas requiere un aspecto que ya está construido en TCP, que es el manejo de las retransmisiones de los paquetes al mismo Servidor, este es un pequeño precio que se tiene que pagar por las ventajas del uso de UDP en el protocolo.

Retransmisiones de los paquetes.- Si el Servidor RADIUS y el Servidor RADIUS secundario comparten el mismo secreto compartido, entonces es correcto retransmitir el mismo paquete al Servidor RADIUS secundario con el mismo identificador y autenticador de petición, debido a que el contenido de los atributos no ha cambiado. Si alguno de los atributos es cambiado, entonces el identificador y más aún el autenticador deberán ser igualmente modificados. Un NAS puede usar el mismo identificador con todos los Servidores, o puede manejarlos en forma independiente para cada uno de ellos.

3.7.- Formato de los paquetes RADIUS

El protocolo RADIUS utiliza paquetes UDP para las transmisiones entre el Cliente y el Servidor, el protocolo establece que el puerto utilizado es el 1812 (decimal).

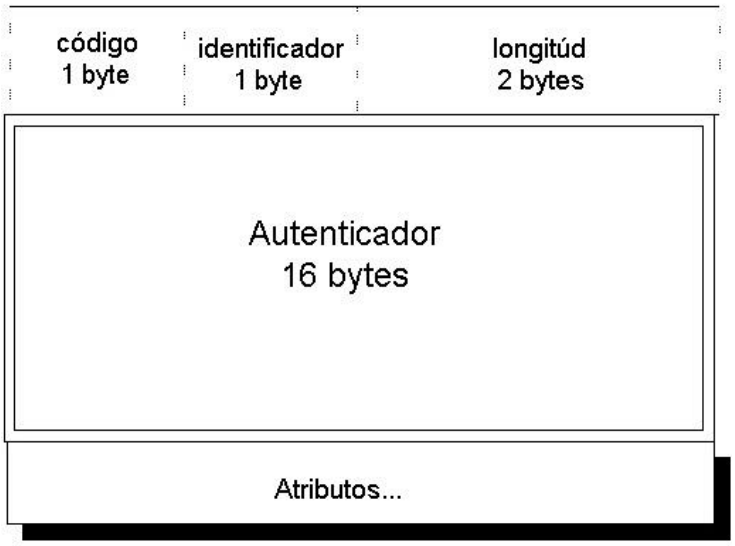


Figura 3.2 Formato de los paquetes RADIUS

Cada paquete está formado por 5 regiones distintas, las cuales se describen a continuación:

Código.- El campo del código es de un byte de longitud e identifica el tipo de paquete, los códigos válidos son:

1	Access-Request
2	Access-Accept
3	Access-Reject
4	Accounting-Request
5	Accounting-Response
11	Access-Challenge
12	Status-Server (experimental)
13	Status-Client (experimental)
255	reservado

Identificador.- El campo identificador es de un byte, y es utilizado para diferenciar e identificar las peticiones duplicadas cuando se trata de dos paquetes conteniendo la misma dirección IP y puerto en el lado del Cliente, además este campo permite establecer la correspondencia entre petición y respuesta.

Longitud.- El campo longitud es de dos bytes, e indica la longitud completa del paquete incluyendo el código, identificador, longitud, autenticador y los campos atributos. Los bytes fuera del rango del campo longitud deben ser tratados como relleno e ignorados en la recepción, si el paquete es más corto de lo que indica el campo longitud, entonces debe ser descartado, la longitud mínima es 20 y la longitud máxima es 4096.

Autenticador.- El campo autenticador es de 16 bytes, siendo el byte más significativo el primero en ser transmitido, este valor es utilizado para autenticar la respuesta del Servidor RADIUS, este campo permite revisar e inspeccionar la integridad del mensaje, y es usado en el algoritmo para ocultar el password. Hay dos tipos específicos de autenticadores, el autenticador de petición y el de respuesta. Los autenticadores de petición son usados en los paquetes Authentication-Request y Accounting-Request.

Autenticador de petición.- En los paquetes Access-Request el valor del autenticador es un número aleatorio de 16 bytes de longitud, llamado Request Authenticator. El valor debería ser único e impredecible sobre el tiempo de vida de un secreto (el secreto compartido entre el Cliente y el Servidor RADIUS), dado que la repetición de un valor de petición en conjunto con el mismo secreto, permitiría a un atacante contestar con una respuesta previamente interceptada. Dado que es esperado que el mismo secreto pueda ser usado par autenticar con Servidores en regiones geográficas dispersas, el campo autenticador de petición debería exhibir unicidad global y temporal.

El valor del autenticador de petición en el paquete Access-Request debería ser también impredecible, para que un atacante a una petición futura predicha, y entonces use la respuesta para enmascararse como el Servidor, en una posterior petición Access-Request.

A pesar de que los protocolos como RADIUS, son incapaces de proteger contra el robo de una sesión autenticada, la generación de peticiones únicas e impredecibles puede proteger contra un amplio rango de ataques activos contra autenticación.

Autenticador de Respuesta.- El valor del campo autenticador en los paquetes Access-Accept, Access-Request y Access-Challenge, es llamado el autenticador de respuesta, y contiene un hash MD5 calculado sobre un flujo de bytes consistente de: el paquete RADIUS, comenzando con el código, incluyendo el identificador, la longitud, el campo autenticador de petición del paquete Access-Request, los atributos de respuesta seguidos por el secreto compartido, esto es:
ResponseAuth = MD5(código + Identificador + longitud + RequestAuth +atributos + secreto compartido)

El secreto, (password compartido entre el Cliente y el Servidor RADIUS) debería ser al menos tan largo y no adivinable como un password bien escogido. Es preferible que el secreto sea al menos de 16 bytes, esto es para asegurar un rango suficientemente grande para que el secreto provea protección contra ataques de búsqueda exhaustiva, el secreto no debe ser vacío (longitud 0), dado que esto permitiría paquetes trivialmente falsificables.

Un Servidor RADIUS debe usar la dirección IP del paquete UDP RADIUS para decidir cual secreto compartido usar, para que las peticiones de RADIUS puedan ser retransmitidas.

Cuando se usa un proxy para retransmitir, el proxy debe ser capaz de alterar el paquete conforme pasa a través de cada dirección, cuando el proxy retransmite la petición, el proxy puede incluir un atributo Proxy-State como ya se mencionó en la sección 2.5, por lo cual ese atributo nuevo invalida la firma, con lo cual el Servidor proxy tendrá que volver a firmar el paquete.

3.8.- Tipos de paquetes RADIUS

EL tipo de paquete RADIUS está determinado por el campo código ubicado en el primer byte del paquete.

Access-Request (Paquete de petición de acceso).- Este tipo de paquetes son enviados a un Servidor RADIUS, e incluyen información usada para determinar si un Usuario tiene permitido el acceso a un determinado NAS, y a los servicios especiales solicitados por tal Usuario. Una implementación que desea autenticar a un Usuario, debe transmitir un paquete RADIUS con el campo código puesto en 1.

Una vez recibido un Access-Request de un Cliente válido, una respuesta apropiada debe ser transmitida. Un Access-Request debería contener un atributo User-Name, y debe contener ya sea un atributo NAS-IP-ADDRESS o NAS-IDENTIFIER o ambos.

Un paquete Access-Request debe contener ya sea un atributo User-Password o un CHAP-Password pero no deberá contener ambos; también debería contener atributos ya sea un NAS-PORT, un NAS-PORT-TYPE o ambos, a menos que el tipo de acceso solicitado no involucre un puerto, o el NAS no distinga entre sus puertos.

Cuando este paquete contiene un atributo *User-Password*, este debe estar oculto, usando un método basado en el algoritmo *Message Digest* MD5.

Descripción de los campos de este paquete:

Código:	1
Identificador:	El campo identificador debe ser cambiado cada vez que el contenido de los campos de atributos cambia, y cada vez que una respuesta válida ha sido recibida para una petición previa, para retransmisiones, el identificador debe permanecer sin cambios.
Autenticador:	El autenticador de petición debe ser cambiado cada vez que un nuevo identificador es usado.
Atributos:	El campo de atributos es variable en longitud, y contiene la lista de atributos que son requeridos para el tipo de servicio así como cualesquiera atributos opcionales.

Access-Accept (paquete de aceptación).- Este tipo de paquete es enviado por el Servidor RADIUS, y provee información de configuración específica necesaria para comenzar a entregar el servicio al Usuario. Si todos los valores de los atributos recibidos en un paquete Access-Request son aceptables, entonces la implementación RADIUS debe transmitir un paquete con el campo código puesto en 2.

En la recepción de un paquete de aceptación, el campo identificador es verificado que coincida con una petición de acceso pendiente. El campo autenticador de respuesta debe contener la respuesta correcta para la petición de acceso pendiente, todos los paquetes inválidos son descartados.

Descripción de los campos de este paquete:

Código:	2
Identificador:	El campo identificador es una copia del campo identificador del paquete de petición de acceso el cual originó este paquete de aceptación.
Autenticador:	El valor del autenticador de respuesta es calculado del valor del paquete de petición de acceso, como se describió antes.
Atributos:	El campo de atributos es variable en longitud, y contiene una lista de cero o más atributos.

Access-Reject (paquete de rechazo).- El Servidor RADIUS envía un paquete de rechazo al Cliente, si éste debe negar cualquiera de los servicios solicitados en el paquete de petición de acceso. La negación del servicio puede estar basada en políticas del sistema, privilegios insuficientes o cualquier otro criterio. El paquete de rechazo puede ser enviado en cualquier momento durante una sesión, lo cual lo hace ideal para forzar limites de tiempo en conexiones, sin embargo no todos los equipos soportan recibir un paquete de rechazo durante una conexión establecida.

Descripción de los campos de este paquete:

Código:	3
Identificador:	El campo identificador es una copia del campo identificador del paquete de petición de acceso el cual originó este paquete de rechazo.
Autenticador:	El valor del autenticador de respuesta es calculado del valor del paquete de petición de acceso, como se describió antes.
Atributos:	El campo de atributos es variable en longitud, y contiene una lista de cero o más atributos.

Access-Challenge (paquete de aceptación pendiente).- Cuando un Servidor RADIUS necesita más información, o simplemente desea disminuir el riesgo de una autenticación fraudulenta, puede regresar un paquete Access-Challenge al Cliente. Este paquete es enviado al Usuario, solicitando la respuesta a un "reto", para poder determinar si la autenticación es completada o no. El Cliente al recibir un paquete Access-Challenge, debe entonces solicitar una nueva petición de acceso con la información apropiada incluida.

Algunos Clientes no soportan el proceso Access-Challenge, por lo que al recibir un paquete de este tipo lo tratan como un paquete de rechazo, negando el servicio al Usuario. Por el contrario, un Cliente que soporta los paquetes Access-Challenge, al recibir uno, revisa que el campo identificador coincida con uno pendiente de alguna petición de acceso. Adicionalmente, el campo autenticador de respuesta debe contener la respuesta correcta a la petición de acceso pendiente, los paquetes inválidos son descartados.

Una vez que la nueva petición de acceso es enviada, el NAS puede desplegar el mensaje de texto, solicitando al Usuario la respuesta, y entonces envía su petición de acceso original con un nuevo identificador de petición y un autenticador de petición, con el atributo User-Password reemplazado por la respuesta del Usuario cifrada.

Descripción de los campos de este paquete:

Código:	11
Identificador:	El campo identificador es una copia del campo identificador del paquete de petición de acceso el cual originó este paquete.
Autenticador:	El valor del autenticador de respuesta es calculado del valor del paquete de petición de acceso, como se describió antes.
Atributos:	El campo de atributos es variable en longitud, y contiene una lista de cero o más atributos.

3.9.- Atributos de los paquetes RADIUS

Los atributos de los paquetes RADIUS contienen información específica para realizar los procesos de autenticación, autorización y detalles de configuración para los paquetes de petición y de respuesta. El final de la lista de los atributos es indicado por la longitud del paquete RADIUS. Algunos atributos

pueden ser incluidos más de una vez, el efecto de esto es específico del atributo, y es especificado en la descripción del atributo, el orden de los atributos de diferentes tipos no es trascendente. Un Servidor RADIUS o Cliente no deben tener dependencias en el orden de los atributos de diferentes tipos.

Figura 3.3.- Formato de los atributos RADIUS

Tipos de atributos.- El campo tipo es de un byte de longitud y puede contener un valor numérico, el nombre del atributo no es enviado en el paquete, los valores permitidos se listan a continuación:

1 User-Name
2 User-Password
3 CHAP-Password
4 NAS-IP-Address
5 NAS-Port
6 Service-Type
7 Framed-Protocol
8 Framed-IP-Address
9 Framed-IP-Netmask
10 Framed-Routing
11 Filter-Id
12 Framed-MTU
13 Framed-Compression
14 Login-IP-Host
15 Login-Service
16 Login-TCP-Port
17 (unassigned)
18 Reply-Message
19 Callback-Number
20 Callback-Id
21 (unassigned)
22 Framed-Route
23 Framed-IPX-Network
24 State
25 Class
26 Vendor-Specific
27 Session-Timeout
28 Idle-Timeout

29 Termination-Action
30 Called-Station-Id
31 Calling-Station-Id
32 NAS-Identifier
33 Proxy-State
34 Login-LAT-Service
35 Login-LAT-Node
36 Login-LAT-Group
37 Framed-AppleTalk-Link
38 Framed-AppleTalk-Network
39 Framed-AppleTalk-Zone
40-59 (reserved for accounting)
60 CHAP-Challenge
61 NAS-Port-Type
62 Port-Limit
63 Login-LAT-Port

Longitud del atributo.- El campo longitud es de un byte de tamaño, e indica la longitud de este atributo, incluyendo los campos tipo, longitud y valor. Si un atributo es recibido en una petición de acceso, pero con una longitud invalida, un paquete de rechazo debería ser transmitido. Si un atributo es recibido en un paquete de aceptación, rechazo o reto, pero con una longitud invalida, el paquete debe ser descartado.

Valor del atributo.- El campo valor es de cero o más bytes, y contiene información específica al atributo. El formato del campo valor es uno de los 5 tipos de datos siguientes:

Texto: 1-253 bytes conteniendo caracteres UTF-8, texto de longitud cero no debe ser enviado, en tal caso, omitir el atributo.

Cadena: 1-253 bytes conteniendo datos binarios (valores desde cero hasta 255 decimal inclusive). Las cadenas de longitud no deben ser enviadas, en tales casos omitir el atributo.

Dirección: valor de 32 bits (4 bytes), para almacenar direcciones IP.

Entero: valor de 32 bits sin signo.

Tiempo: valor sin signo de 32 bits, los atributos estándar no usan este tipo de dato, pero se define para posibles usos futuros.

Para mayor información sobre RADIUS, consultar las referencias [1] y [2].

3.10.- Desventajas y vulnerabilidades de RADIUS

1.- RADIUS utiliza MD5 como algoritmo para calcular la función de hash y en base a eso establecer la autenticidad de los paquetes provenientes tanto de los Usuarios como de los Clientes. Actualmente se han documentado ataques que hacen incierta la seguridad de este algoritmo.

2.- La principal vulnerabilidad en el uso de RADIUS como método de autenticación, es el hecho de que el estándar no define el seguimiento de sesiones, por lo cual se pueden presentar sesiones simultáneas con el mismo identificador, en este caso la dirección MAC, ya que es relativamente fácil encontrar en Internet, un programa capaz de "escuchar" el medio y suplantar una dirección MAC válida para poder autenticarse como un Usuario del sistema. Otra desventaja clara, es el hecho de que no se autentica al Usuario, sino a los dispositivos, por lo cual la persona que tenga en su poder un dispositivo válido (tarjeta de red), puede sin ningún problema vulnerar la seguridad en cuanto a la autenticación de RADIUS se refiere; razón por la cual RADIUS ha quedado completamente rebasado en su método de autenticación.

A continuación se detallan las pruebas de *spoofing* (falsificado de dirección MAC), realizadas sobre *freeradius*:

Para la realización de estas pruebas, se utilizó el programa *macmakeup* el cual fue obtenido de la dirección: *www.gorlani.com/publicbrj/macmakeup/macmakeup.jsp*

- **Suplantando la dirección MAC:** Esta prueba se desarrolló con éxito, únicamente basta con especificar la nueva dirección MAC que queremos que tome la Tarjeta de Red, y reiniciar el servicio, la figura 3.4 muestra la pantalla de salida.

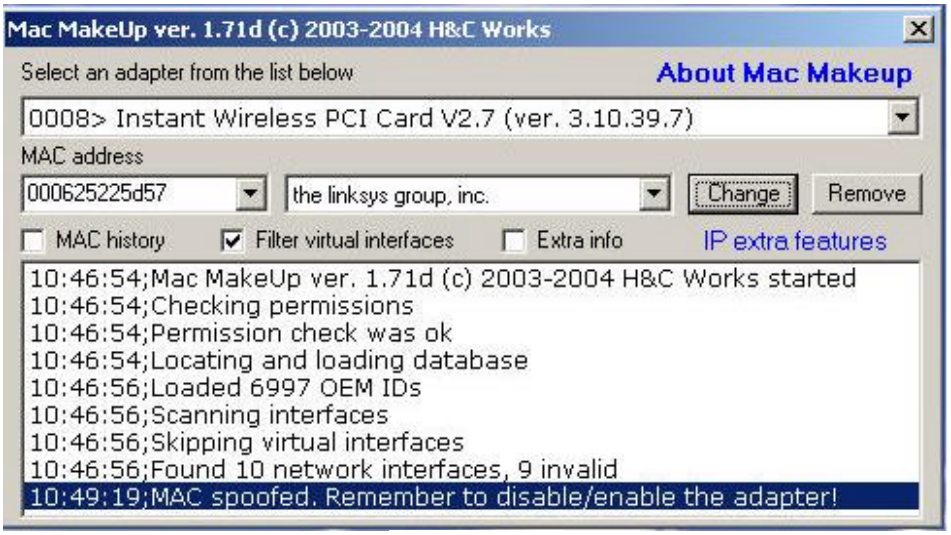


Figura 3.4

La figura 3.5 muestra el cambio de la dirección MAC de la tarjeta, después de haber reiniciado el servicio, la dirección real es la 00-06-25-22-63-21 y la dirección falsificada es la 00-06-25-22-5d-57.

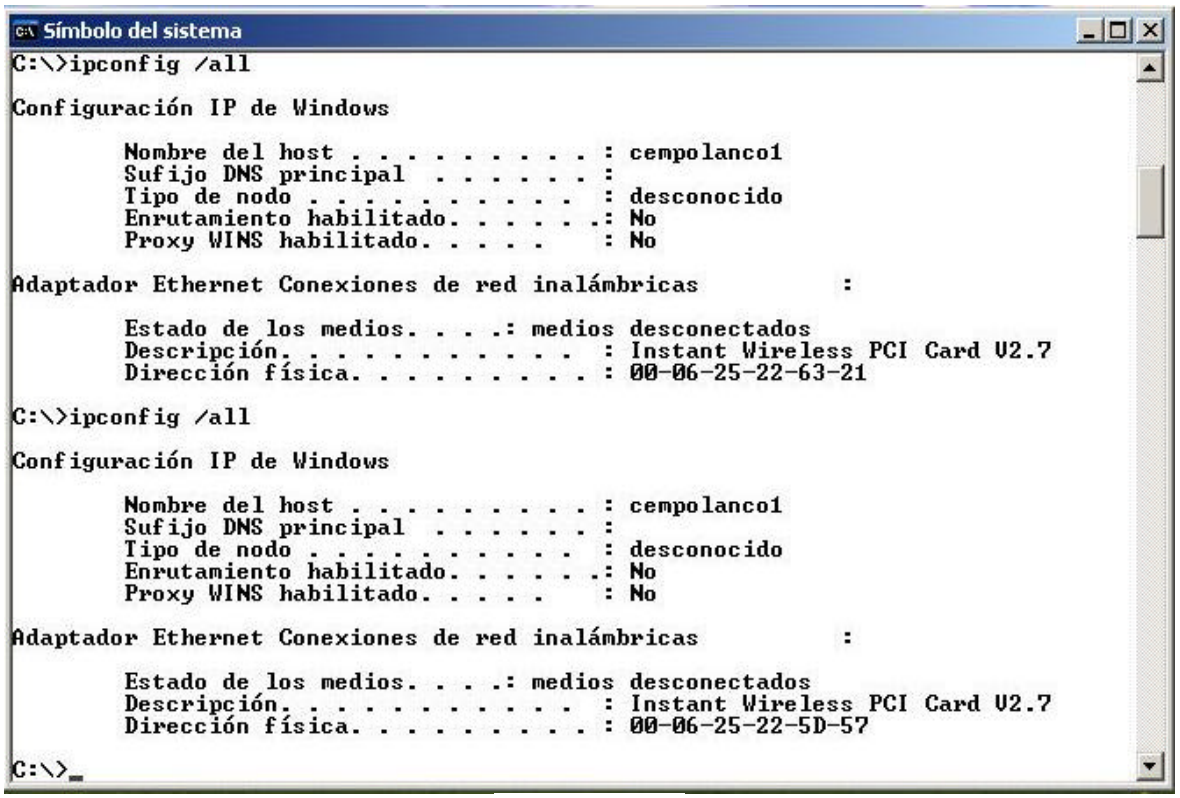


Figura 3.5

El servidor *freeradius* tiene configurada la dirección MAC 00:06:25:22:5d:57 como la dirección válida de uno de sus clientes, por lo cual cuando se solicita la autenticación, el servidor otorga el servicio; a continuación se muestra la salida del archivo de historial en el servidor *freeradius* aceptando al usuario.

```
Module: Instantiated radutmp (radutmp)
Listening on authentication *:1812
Listening on accounting *:1813
Listening on proxy *:1814
Ready to process requests.
rad_recv: Access-Request packet from host 132.248.127.98:6001, id=2,
length=69
        User-Name = "00:06:25:22:5d:57"
        User-Password = "proyecto"
        NAS-IP-Address = 132.248.127.98
        NAS-Port = 0
  Processing the authorize section of radiusd.conf
modcall: entering group authorize for request 0
  modcall[authorize]: module "preprocess" returns ok for request 0
  modcall[authorize]: module "chap" returns noop for request 0
  modcall[authorize]: module "mschap" returns noop for request 0
    rlm_realm: No '@' in User-Name = "00:06:25:22:5d:57", looking up realm
NULL
```

```
    rlm_realm: No such realm "NULL"
  modcall[authorize]: module "suffix" returns noop for request 0
  rlm_eap: No EAP-Message, not doing EAP
  modcall[authorize]: module "eap" returns noop for request 0
    users: Matched 00:06:25:22:5d:57 at 113
radius_xlat:  'Bienvenido, 00:06:25:22:5d:57'
  modcall[authorize]: module "files" returns ok for request 0
modcall: group authorize returns ok for request 0
  rad_check_password:  Found Auth-Type Local
auth: type Local
auth: user supplied User-Password matches local User-Password
radius_xlat:  'Bienvenido, 00:06:25:22:5d:57'
Sending Access-Accept of id 2 to 132.248.127.98:6001
        Reply-Message = "Bienvenido, 00:06:25:22:5d:57"
Finished request 0
Going to the next request
--- Walking the entire request list ---
Waking up in 6 seconds...
--- Walking the entire request list ---
Cleaning up request 0 ID 2 with timestamp 4332db67
Nothing to do.  Sleeping until we see a request
```

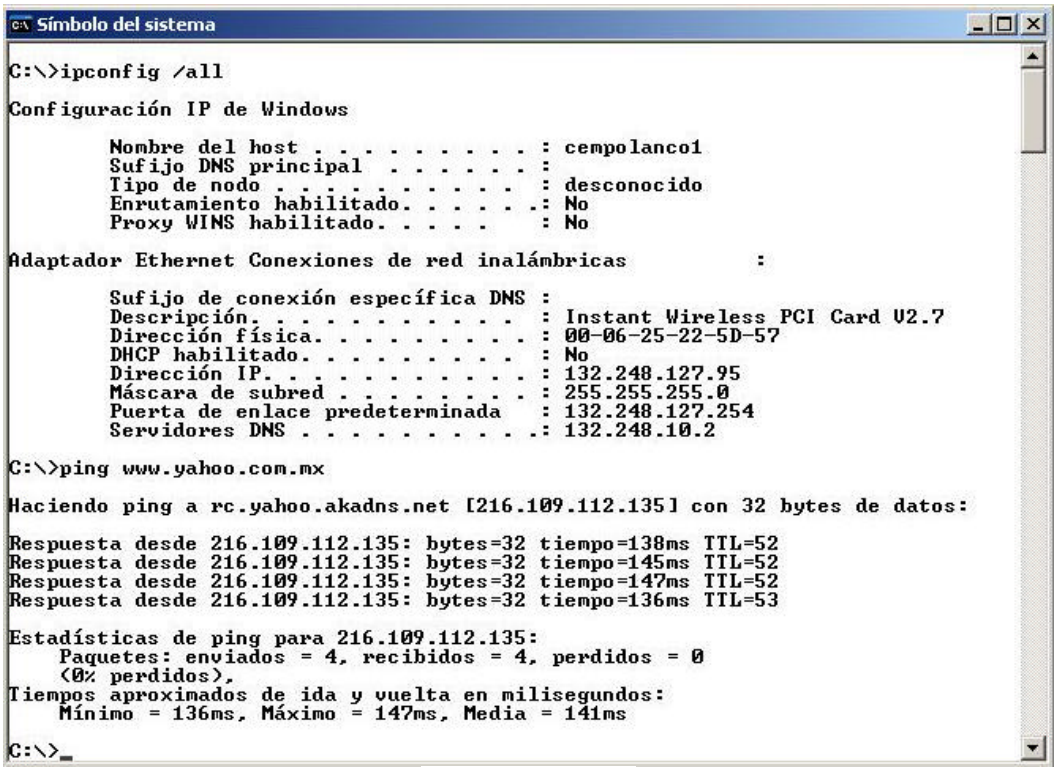


Figura 3.6

La figura 3.6 muestra que efectivamente el Usuario ha sido autenticado con éxito y tiene acceso a la red. Otro aspecto importante para resaltar del archivo de historial de *freeradius*, es el hecho de que en la penúltima línea se puede leer claramente que una vez realizado el proceso de autenticación, la petición se limpia y no hay un seguimiento de sesiones, esto es que si surge una nueva petición de autenticación con la misma dirección MAC el resultado va a ser el mismo, una aceptación.

CAPITULO 4.- DEFINICIÓN DEL PROTOCOLO

4.1.- Introducción.

Este capítulo describe en detalle la definición del protocolo de autenticación de Usuarios, el cual fue implementado como objetivo del trabajo de tesis de maestría. Debido a que los dispositivos físicos (puntos de acceso y tarjetas inalámbricas) están muy limitados en cuanto a sus capacidades y no se puede incluir funcionalidad en ellos (programación adicional), el protocolo tuvo que ser implementado sobre la infraestructura existente, estando consciente de las limitantes, y por lo cual me permito hacer las siguientes observaciones:

1.- Autenticación parcial validando la dirección MAC y la asociación de la tarjeta MAC con algún Usuario.- Esta validación parcial se debe a las limitantes del estándar RADIUS, el cual únicamente define la dirección MAC como autenticador para el caso de las redes inalámbricas[5], por lo cual el protocolo tiene que primeramente, autenticar de acuerdo al estándar, y posteriormente se realiza la ejecución propia del protocolo para establecer una autenticación final sin restricciones. Una solución para evitar esto seria la inclusión de nuevos atributos dentro del estándar que soportaran el protocolo.

2.- El Usuario "ejecuta" el programa del protocolo.- El Usuario tiene que ejecutar e iniciar el programa para el protocolo justo en el momento en que se realiza la autenticación parcial, puesto que en ese preciso momento comienza la sesión del Usuario y existe una asociación entre ambas partes Usuario-Servidor para el intercambio de paquetes, una solución elegante seria que el sistema operativo iniciara la ejecución del programa. Una vez comenzada la ejecución del programa, el Usuario ya no tiene que hacer nada mas que introducir su nombre de usuario y contraseña.

3.- Seguimiento de sesiones.- El seguimiento por parte del Servidor, de cada una de las sesiones activas es un aspecto fundamental el cual no es cubierto por el estándar RADIUS. La forma en que se resolvió este problema, fue implementando del lado del Usuario, un programa que periódicamente envía un paquete denominado PAQUETE DE SESIÓN, el cual únicamente sirve para notificar al Servidor que la sesión del Usuario permanece activa. La solución más obvia seria que el Punto de Acceso notifique mediante un paquete especial al Servidor de autenticación, el momento en que la sesión de un Usuario finaliza, esto es cuando el Punto de Acceso desasocia a dicho Usuario.

4.2.- Operación.

El protocolo comienza cuando un Usuario desea autenticarse dentro de un sistema, para hacer uso de recursos, en este caso un Usuario intenta tener acceso a una red inalámbrica. Tomando en cuenta que el Usuario posee un equipo de cómputo conteniendo un dispositivo de red inalámbrico, el cual va a permitir el intercambio de información en forma de ondas de radio.

[5] Debido a que RADIUS es un servidor de Autenticación utilizado para diversos ámbitos, existen múltiples mecanismos de autenticación, pero el único de ellos definido en el estándar para el caso de redes inalámbricas es mediante la autenticación de las direcciones MAC (MAC based authentication).

El primer paso consiste en que el dispositivo del Usuario tiene que asociarse con el Punto de Acceso (Cliente) para poder intercambiar información, el proceso de asociación ha sido detallado en la sección 1.2. El Punto de Acceso, filtra todo el tráfico que no corresponda con paquetes definidos en el protocolo EAPOL [6], hasta que se lleva a cabo la autenticación del Usuario, momento en el cual el Cliente libera todos los filtros, permitiendo así la libre comunicación e intercambio de información.

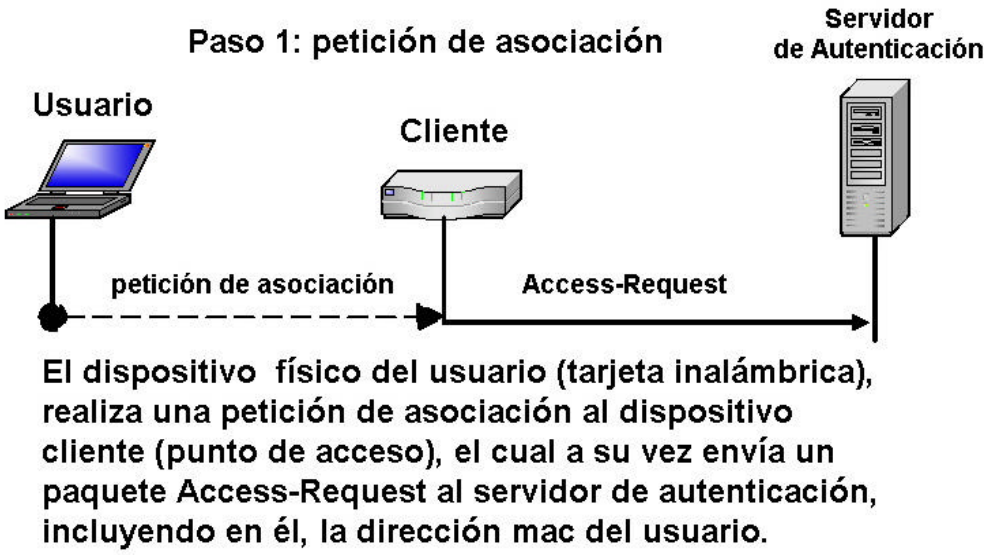


Figura 4.1

Una vez que el Cliente recibe una petición de asociación por parte de un Usuario, el Punto de Acceso genera un paquete de petición de acceso RADIUS (Access-Request descrito en el capítulo 3.8), y lo envía al Servidor de autenticación, quien al recibirlo tiene que verificar que se cumplen las siguientes características:

1) Que la dirección MAC contenida en el paquete, se encuentre registrada en su base de datos.
2) Que el Punto de Acceso que envió el paquete, también esté registrado en la base de datos y el secreto compartido entre ambos (Cliente y Servidor) esté disponible.
3) Que el paquete no haya sido modificado (validación de la integridad), esto es calculando la función de *hash* MD5 -con un valor de ceros en la sección del autenticador- del paquete utilizando el secreto compartido entre el Servidor y el Cliente, y comparando este resultado con el valor del *hash* contenido en el paquete en la sección de autenticador (ver capítulo 3.7).

Cuando los incisos 1, 2 y 3 han sido verificados, esto es que la dirección MAC esta registrada y que el paquete no ha sido modificado en su integridad, el Servidor de Autenticación genera y envía al Cliente, un paquete RADIUS Access-Accept (descrito en el capítulo 3.8), al cual se le incluye como

[6] EAPOL son las siglas de Extended Authentication Protocol Over LAN, para mayores referencias consultar el sitio del consorcio IEEE en [4].

atributo la duración de la sesión actual (el atributo Session-Timeout es mencionado en el capítulo 3.9), con un valor de aproximadamente 30 segundos[7], tiempo suficiente para que se lleve a cabo todo el protocolo de autenticación, obligando al Cliente a que terminado dicho periodo de tiempo vuelva a solicitar una nueva petición de autenticación por medio de otro paquete Access-Request, resultando en una autenticación parcial.

Para el caso en el cual alguno de los incisos anteriores no es satisfecho en su totalidad, el paquete generado y enviado por el Servidor de autenticación será uno de rechazo (Access-Reject).

Una vez que el Usuario ha sido autenticado en esta primera parte, los filtros puestos por parte del Cliente son retirados, permitiendo así el completo intercambio de paquetes de cualquier tipo.

La figura 4.2 ilustra el esquema para el paso 2 en el proceso de autenticación.

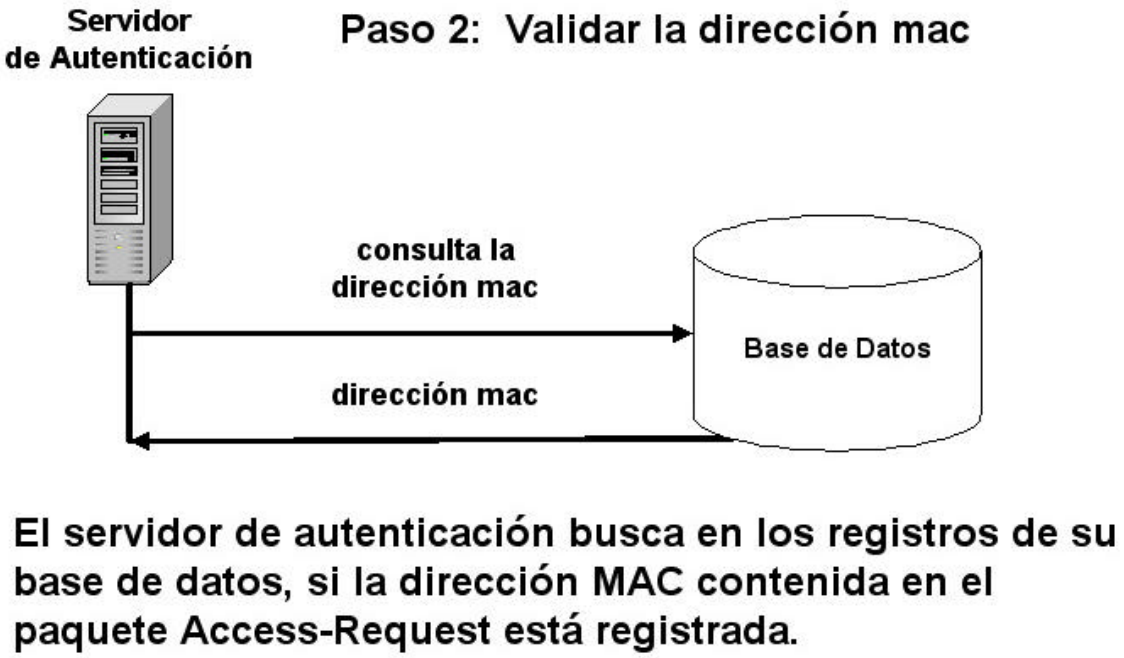
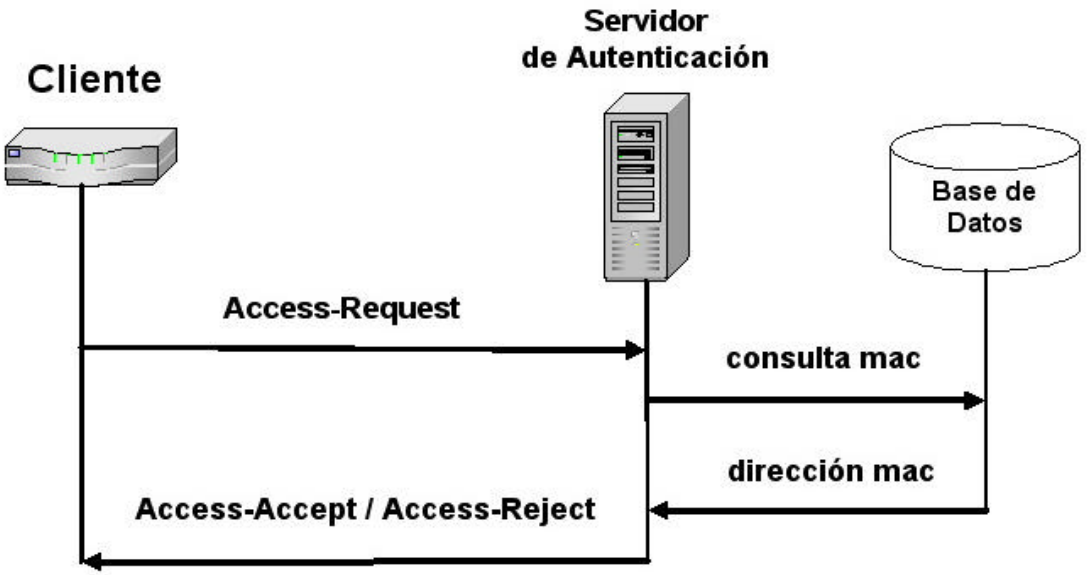


Figura 4.2

[7] Este intervalo de tiempo fue escogido tomando en consideración un tiempo razonable para llevar a cabo el protocolo, sin embargo este valor puede ser modificado y reducido.

Paso 3: Aceptación o rechazo



El servidor de autenticación, contesta al cliente con un paquete Access-Accept cuando la dirección mac está registrada en caso contrario envía un paquete Access-Reject,negando el acceso a los servicios solicitados por el usuario.

Es necesario recalcar el hecho de que cuando el servidor de autenticación envía un paquete Access-Accept, establece un atributo de límite de sesión, obligando en este caso al cliente a solicitar una petición de autenticación una vez finalizado dicho periodo de tiempo.

Figura 4.3


Paso 4: Asociación Establecida
Servidor de Autenticación
Usuario
Cliente
Una vez que se ha establecido una asociación, el usuario puede comunicarse con el servidor de autenticación, para enviar y recibir paquetes de datos.

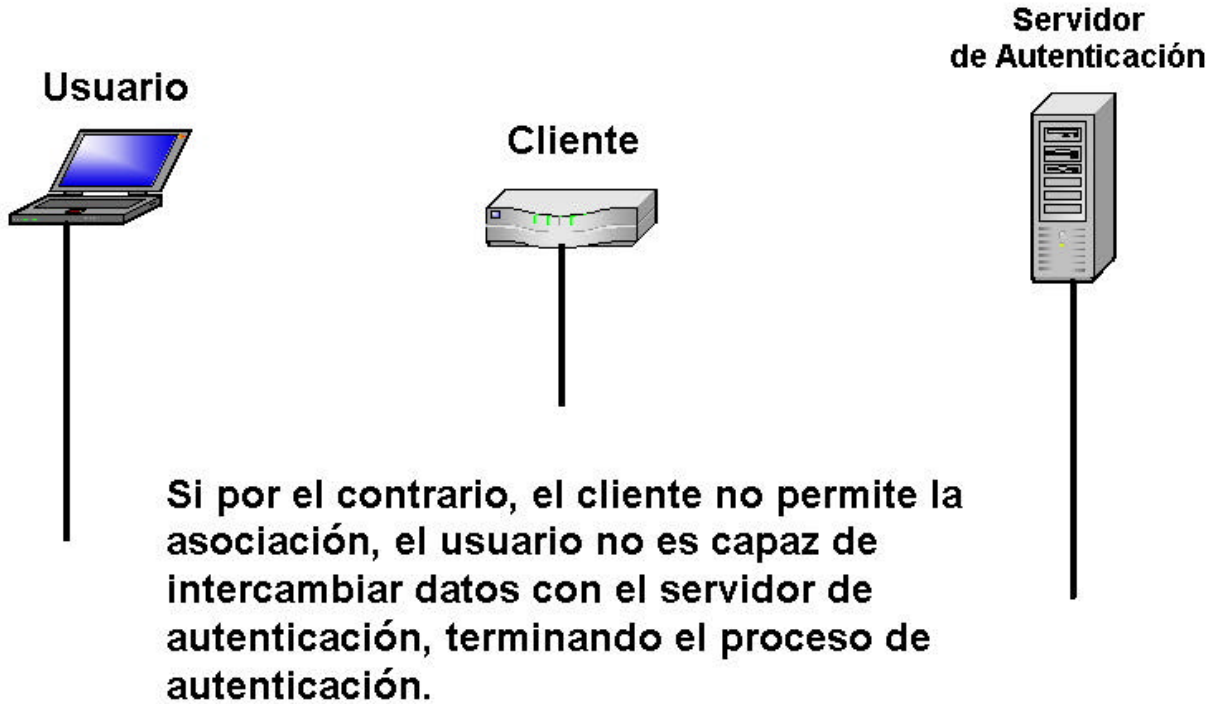

Servidor de Autenticación
Usuario
Cliente
Si por el contrario, el cliente no permite la asociación, el usuario no es capaz de intercambiar datos con el servidor de autenticación, terminando el proceso de autenticación.

Figura 4.4

Si el proceso de autenticación en esta primera fase tuvo éxito, entonces se procede con la segunda parte, la que representa propiamente hablando el protocolo de autenticación objeto del presente trabajo de tesis.

Primeramente, el Usuario tiene que iniciar un intercambio de paquetes de comunicación (el tipo y formato de estos paquetes se define en la sección 4.3) con el Servidor, en un puerto conocido por ambos y definido para ser utilizado únicamente para este fin (para el caso de esta implementación el puerto utilizado fue el 1815). El Usuario tiene que proveer su nombre de usuario y contraseña, (de la misma forma en la que ambos datos fueron registrados en la base de datos del Servidor de Autenticación).

El **PAQUETE DE PETICIÓN DE AUTENTICACIÓN**, es generado y enviado al Servidor por el Usuario de acuerdo a las siguientes características:

1) El autenticador es un número de 8 a 10 dígitos de longitud representado en un tipo de dato entero de 32 bits[8], de los cuales realmente se utilizan 31 ya que el bit más significativo utilizado para representar el signo del valor numérico siempre contiene un valor de cero indicando un número positivo (ya que en el lenguaje de programación java no se puede utilizar el modificador *unsigned* como en algunos otros lenguajes para quitar el signo); este autenticador es generado aleatoriamente y cifrado con AES[9], usando como llave de cifrado el resultado de aplicar la función de *hash* MD5 a la contraseña del Usuario, esto con el objeto de ajustar la contraseña o frase, al tamaño requerido por el algoritmo AES (128 bits).
2) El atributo nombre de Usuario contiene el valor del campo *login* introducido por el Usuario
3) El *hash* de verificación de integridad, es calculado colocando todos los bytes del paquete en el orden establecido, poniendo ceros en la sección del *hash* dentro del paquete, y reemplazando finalmente este valor con el resultante de aplicar el algoritmo SHA1[10] con contraseña al paquete formado, con la contraseña incluida al final, para efectos de verificación de la integridad.

Definiciones:

- Sea n el número generado aleatoriamente de tamaño 32 bits, representado como dígitos en una cadena de caracteres.
- Sea l el nombre de usuario o *login* proporcionado por el Usuario
- Sea p la contraseña del usuario o (*password* o *passphrase*) también proporcionado por el Usuario.
- Sea k la llave de cifrado utilizada en el algoritmo AES, entonces se tiene que k = MD5 (p), el tamaño de k es 128 bits resultado de aplicar la función de *hash* MD5.

[8] El tamaño de este autenticador se fijó en 8, 9 o 10 dígitos, no obstante puede ser extendido a cualquier tamaño y representado en algún tipo de dato mayor de 32 bits.
[9] En una segunda versión del protocolo fue reemplazado el algoritmo DES por su contraparte AES para lograr un mayor grado de seguridad.
[10] En una segunda versión del protocolo fue reemplazado el algoritmo MD5 por su contraparte SHA1

- Sea ***d*** el paquete generado
- Sea AES_k (***n***) el resultado de aplicar el algoritmo de cifrado AES utilizando la llave ***k*** al valor ***n***, entonces AES_k (***n***) es de tamaño 16 bytes.
- Sea SHA1 (***d***) el resultado de aplicar la función de *hash* SHA1 al paquete ***d***.

Dadas las definiciones previas se tiene que:

1) El valor calculado de AES_k (***n***) es puesto en la sección de autenticador en el paquete de PETICIÓN DE AUTENTICACIÓN (este paquete es descrito en detalla en la sección 4.3).
2) El verificador de integridad (20 bytes) es puesto con valores de cero y se calcula SHA1 (d) con la contraseña y se sustituye el resultado en la misma sección del verificador de integridad.

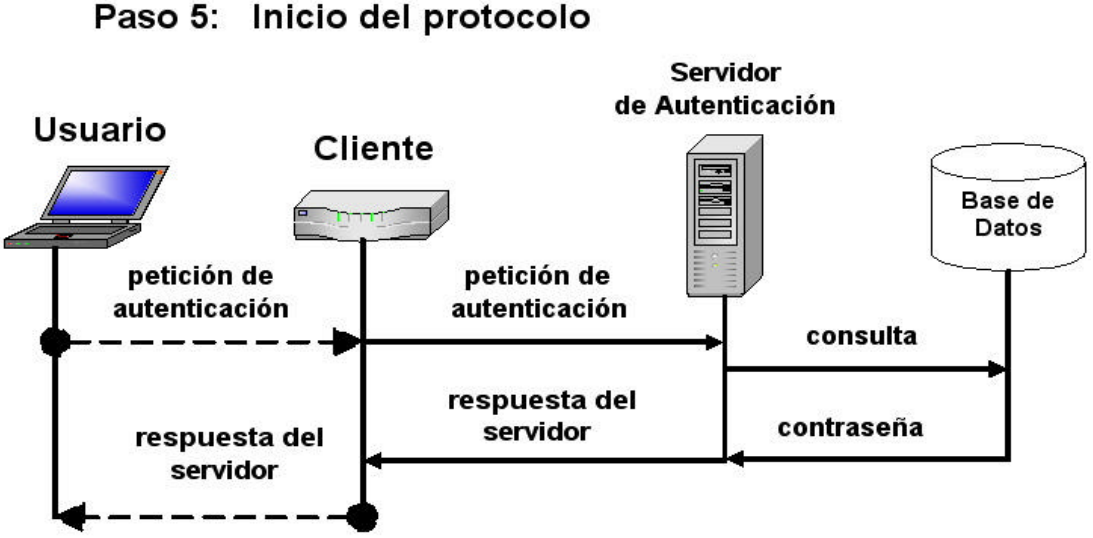


Figura 4.5

Una vez que el Servidor recibe el paquete de petición de autenticación, tiene que verificar que se cumplan las siguientes condiciones:

1) Obtener el valor del atributo nombre de Usuario del paquete, verificar que en su base de datos se encuentre registrado el Usuario, y de ser así recuperar su contraseña.
2) Validar la integridad del paquete, esto es primero colocando ceros en la sección del verificador de integridad del paquete recibido, segundo aplicando el algoritmo SHA1 a todo el paquete, utilizando como llave de nueva cuenta el resultado de aplicar la función de *hash* MD5 a la contraseña o *passphrase* del Usuario, y finalmente se compara que el valor resultante sea exactamente igual al valor del hash contenido en la sección de verificador de integridad del mismo paquete.
3) Verificación del autenticador cifrado, descifrando la sección del autenticador, usando AES y como llave de cifrado / descifrado la contraseña.

Definiciones:

- Sea d el datagrama recibido por el Servidor y enviado por el Usuario denominado paquete de PETICIÓN DE AUTENTICACIÓN, el tamaño del paquete depende del tamaño del nombre de usuario.
- Sea m el autenticador cifrado y recibido en el paquete d, tal que m es de tamaño 16 bytes, entonces m debe corresponder a AES_k (n).
- Sea l el nombre de usuario o *login* recibido como atributo en el mismo paquete d.
- Sea p la contraseña del usuario o (*password* o *passphrase*) obtenida de consultar la base de datos, buscando por el atributo l.
- Sea k la llave de cifrado utilizada en los algoritmos AES y SHA1, entonces se tiene que k = MD5 (p), el tamaño de k es 128 bits.
- Sea i el verificador de integridad proveniente en el paquete d, de tamaño 20 bytes.
- Sea d' el datagrama igual a d pero con valores de cero reemplazando el valor de i.
- Sea SHA1 (d') el resultado de aplicar la función de *hash* al paquete d' con k concatenado al final.
- Sea $t = AES_k (m)^{-1}$, el resultado de aplicar el algoritmo de cifrado AES utilizando la llave k al valor m pero en modo de descifrar, por lo cual t es el autenticador cifrado previamente por el Usuario.

Entonces se tiene lo siguiente:
1.- Si SHA1 (d') es igual a i se dice que el paquete es íntegro, esto es que no fue alterado o modificado por nadie durante su trayecto.
2.- Si t es posible calcularlo, esto implica que el valor de m fue cifrado y descifrado con la misma llave (en otras palabras que esa llave representa el secreto compartido entre el servidor de autenticación y un Usuario).

Si estos 2 puntos son satisfactoriamente cubiertos, el Servidor genera y envía de regreso, un denominado **PAQUETE DE RESPUESTA DEL SERVIDOR**, con las siguientes características:

1) El nuevo autenticador se forma concatenando el autenticador del Usuario recibido en el paquete de petición, con un número generado aleatoriamente por el Servidor, de entre 8 y 10 dígitos, y conteniendo el carácter '|' entre los dos valores, de la forma < *autenticador de petición* | *autenticador del Servidor* >; el resultado de esta concatenación es cifrado utilizando el algoritmo AES y como llave de cifrado, de nueva cuenta el resultado de aplicar el algoritmos MD5 a la contraseña del Usuario obtenida de la Base de Datos.
2) El *hash* de verificación de integridad, es calculado colocando todos los bytes del paquete en el orden establecido, poniendo ceros en la sección del *hash* dentro del paquete, y reemplazando finalmente este valor con el resultante de aplicar el algoritmo SHA-1 al paquete formado incluyendo la llave.
3) Esta paquete no contiene atributos puesto que no son necesarios.

Definiciones

- Sea n' un número aleatorio generado por el Servidor, con las mismas características que n definido en el paso 1.
- Sea m' el nuevo autenticador cifrado con AES de la forma AES_k (t | n'), con t definido en el paso anterior, la longitud de m' es de 32 bytes, con k = MD5(p), y p obtenido de la Base de Datos.
- Sea d' el paquete generado por el servidor denominado PAQUETE DE RESPUESTA DEL SERVIDOR, d' es de tamaño 52 bytes.
- Sea i' (el verificador de integridad), el valor resultado de aplicar SHA1 (d') con i' conteniendo únicamente ceros, e incluyendo k, la longitud de i' es de 20 bytes.

Dadas las definiciones previas se tiene lo siguiente:

1.- El servidor crea un nuevo autenticador m' (cifrado utilizando la contraseña obtenida de la Base de Datos) y lo coloca en la sección del autenticador cifrado en el paquete d'.

2.- Se calcula i' y se sustituyen los bytes de la sección de autenticador por el nuevo valor de i'.

La figura 4.6 ilustra la secuencia de verificación de autenticidad del servidor.

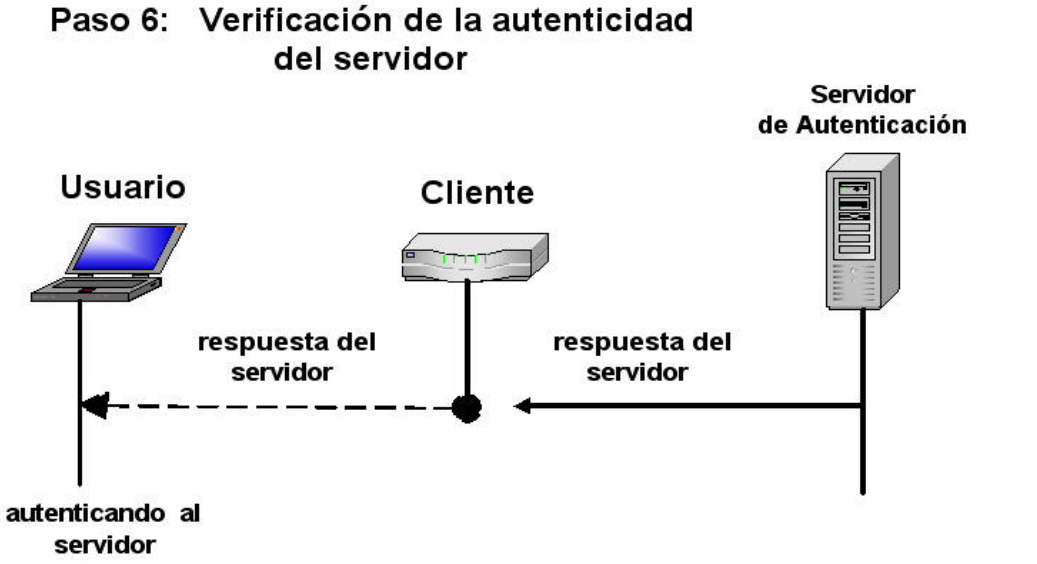


Figura 4.6

El Usuario al recibir el paquete de respuesta del Servidor, verifica su integridad aplicándole el algoritmo SHA1 con llave, comparando el resultado con los bytes del paquete localizados en la sección del verificador de integridad; una vez que la integridad es verificada, el Usuario descifra utilizando como llave el MD5(contraseña), y aplicando el algoritmo AES, la sección de autenticador del paquete, la cual debe contener tanto el autenticador enviado por el Usuario, como el autenticador generado por el Servidor, si estas dos partes son verificadas exitosamente, quiere decir que el Usuario ha autenticado al Servidor.

Definiciones

- Sea j el resultado de aplicar SHA1 (d'), con d' definida en el paso anterior, reemplazando la sección del verificador de integridad por únicamente ceros, la longitud de j es de 20 bytes, incluyendo la llave MD5(p).
- Sea o el resultado de aplicar AES_k (m')$^{-1}$, con m' definida previamente.

Se tiene lo siguiente:

1.- Si se cumple que $j = i'$ se dice que el paquete d' es íntegro, con i' definida en el paso anterior.

2.- Si se cumple que $o = n \mid v$, con n el número generado previamente por el Usuario y v un número que el Usuario presume fue generado por el Servidor, entonces el Servidor ha sido autenticado por el Usuario.

El Usuario genera y contesta al Servidor, con un paquete denominado **PAQUETE DE RESPUESTA DEL USUARIO**, el cual debe estar formado de la siguiente manera:

1) El Usuario vuelve a cifrar con AES_k usando como llave el resultado de MD5(p) el autenticador que recibió por parte del Servidor, y lo coloca en la sección de autenticador del paquete.
2) El *hash* de verificación de integridad, es calculado colocando todos los bytes del paquete en el orden establecido, poniendo en ceros la sección del *hash* dentro del paquete, y reemplazando finalmente este valor con el resultante de aplicar el algoritmo SHA1 con la contraseña al paquete formado.
3) Se incluye el atributo login colocando como valor el nombre de Usuario

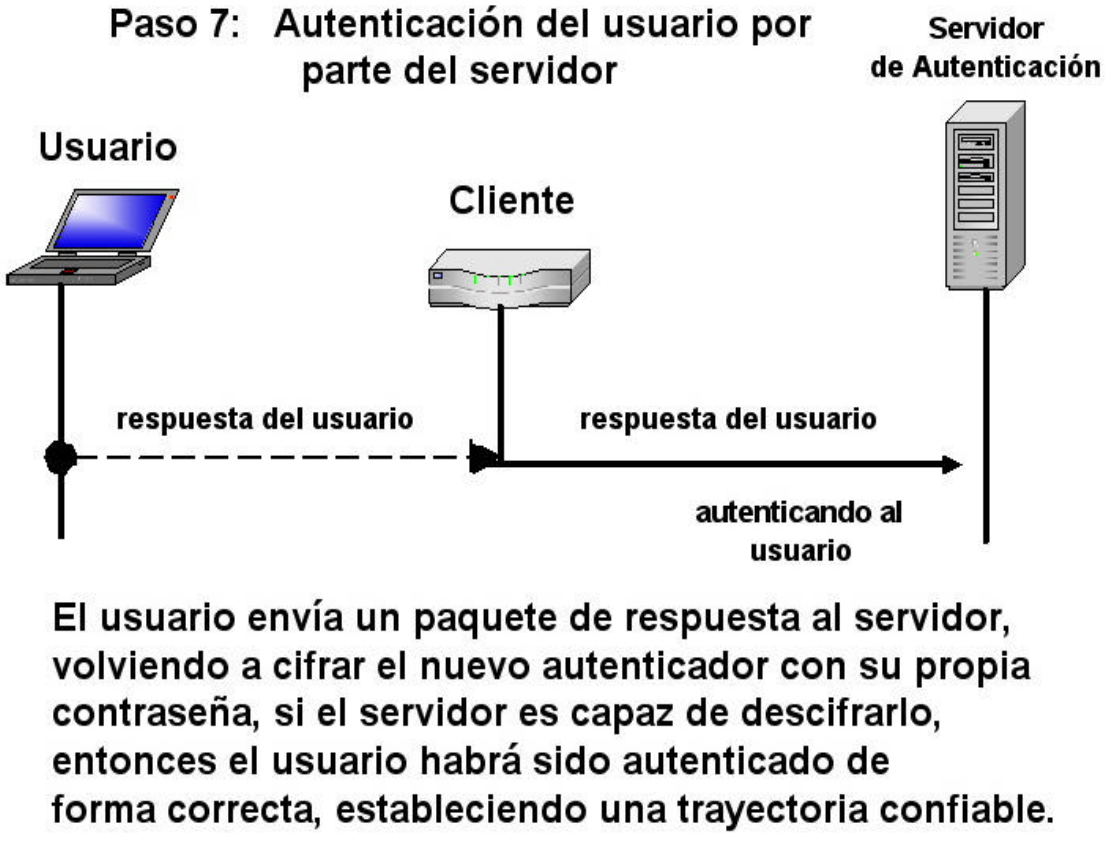


Figura 4.7

Definiciones

- Sea ***d''*** el denominado PAQUETE DE RESPUESTA DEL USUARIO
- Sea ***v'*** = AES_k (***v***), con ***v*** definida previamente, el nuevo autenticador en el paquete ***d''***, ***v'*** es de tamaño 16 bytes.
- Sea ***i''*** el nuevo verificador de integridad calculado de la misma forma que ***i*** e ***i'***.

Dadas las definiciones previas se tiene lo siguiente:

1.- El usuario calcula ***v'*** y lo coloca en la sección del autenticador del paquete ***d''***.

2.- El paquete ***d''*** contiene ***l*** como único atributo, con ***l*** definida previamente.

3.- ***i''*** es calculada y puesta en la sección del verificador de integridad de ***d''***.

Finalmente, el Servidor de autenticación al recibir el paquete de respuesta del Usuario, verifica que se cumplan las siguientes condiciones:

1) Obtener el valor del atributo que contiene el login del Usuario, y recuperar de su base de datos la contraseña del Usuario.
2) Verificar la integridad del paquete, calculando el *hash* SHA1 con llave (contraseña) del paquete recibido.
3) Aplicar AES para descifrar utilizando como llave el resultado MD5(***p***), el autenticador del paquete.

Si se cumple que el autenticador descifrado es exactamente igual al autenticador generado por el Servidor y enviado al Usuario en su paquete de respuesta, entonces en ese momento, el Usuario habrá sido autenticado exitosamente por el Servidor, y una trayectoria confiable entre ambas partes habrá sido establecida. Esto es:

1.- Si se cumple que ***i''*** es igual a SHA1 (***d''***), usando como llave MD5(***p***) colocando ceros en la sección del verificador de integridad, entonces se dice que el paquete es íntegro.

2.- Si se cumple que AES_k (***v'***)$^{-1}$ = ***n'***, con ***k*** = MD5(***p***) y ***p*** obtenido de la Base de Datos, entonces el Servidor ha autenticado al Usuario satisfactoriamente.

Es en este momento cuando el protocolo finaliza su ejecución, la sesión del Usuario es puesta en un estado de ACTIVA, lo cual significa que mientras esta sesión permanezca en ese estado, nadie puede volver a autenticarse utilizando esa dirección MAC ni ese nombre de Usuario, esto con el objeto de implementar el seguimiento de sesiones.

El proceso completo de seguimiento de sesiones se detalla mas adelante en este mismo capítulo.

Una vez terminada la duración establecida para la sesión inicial (aproximadamente 30 segundos), el Cliente vuelve a solicitar un paquete Access-Request para autenticar de nueva cuenta al Usuario. Es entonces cuando el Servidor de autenticación contesta con un definitivo paquete Access-Accept o un Access-Reject, como lo muestra la figura 4.8.

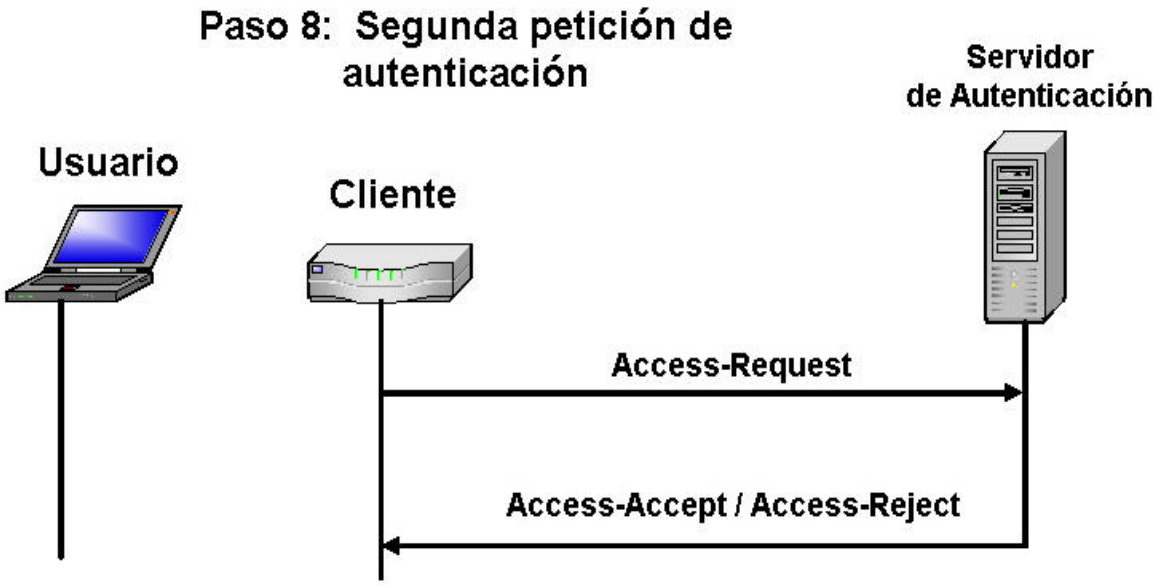


Figura 4.8

Este es el esquema general del proceso de autenticación, incluyendo el diagrama de secuencia entre los mensajes intercambiados por el Usuario y el Servidor de autenticación, siempre actuando el Punto de Acceso como intermediario puesto que es necesario modular la información, esto es convertir de digital a señales de radio que viajan por el aire y viceversa.

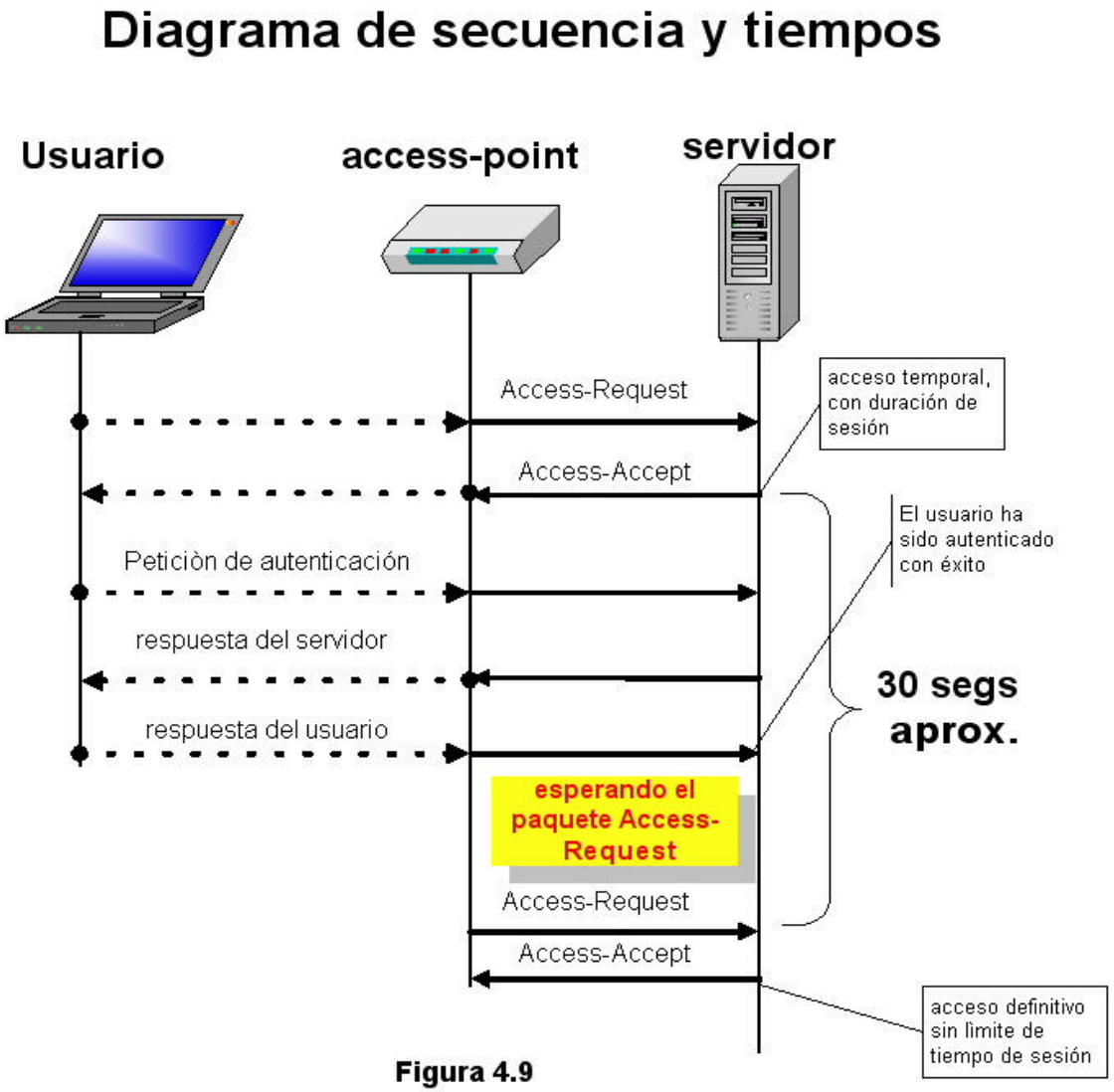


Figura 4.9

Seguimiento de sesiones

En el momento en que el Usuario ha sido autenticado de forma exitosa, entonces da inicio formalmente el seguimiento de su sesión. El primer paso consiste en que el Usuario mantiene un proceso que periódicamente (para efectos de este proyecto se definió el valor en 1 minuto) envía un paquete al Servidor de Autenticación, ese paquete se definió con el nombre de PAQUETE DE SESIÓN (es detallado en la sección 4.3), y únicamente es utilizado para indicarle al Servidor de Autenticación que la sesión de un Usuario permanece activa.

El servidor periódicamente revisa para todas las sesiones activas de sus Usuarios y determina si alguna sesión ha finalizado, esto es que durante un lapso de tiempo no ha recibido ningún paquete de sesión de ese Usuario; cuando esto ocurre, la sesión del Usuario es eliminada de la cola de sesiones activas.

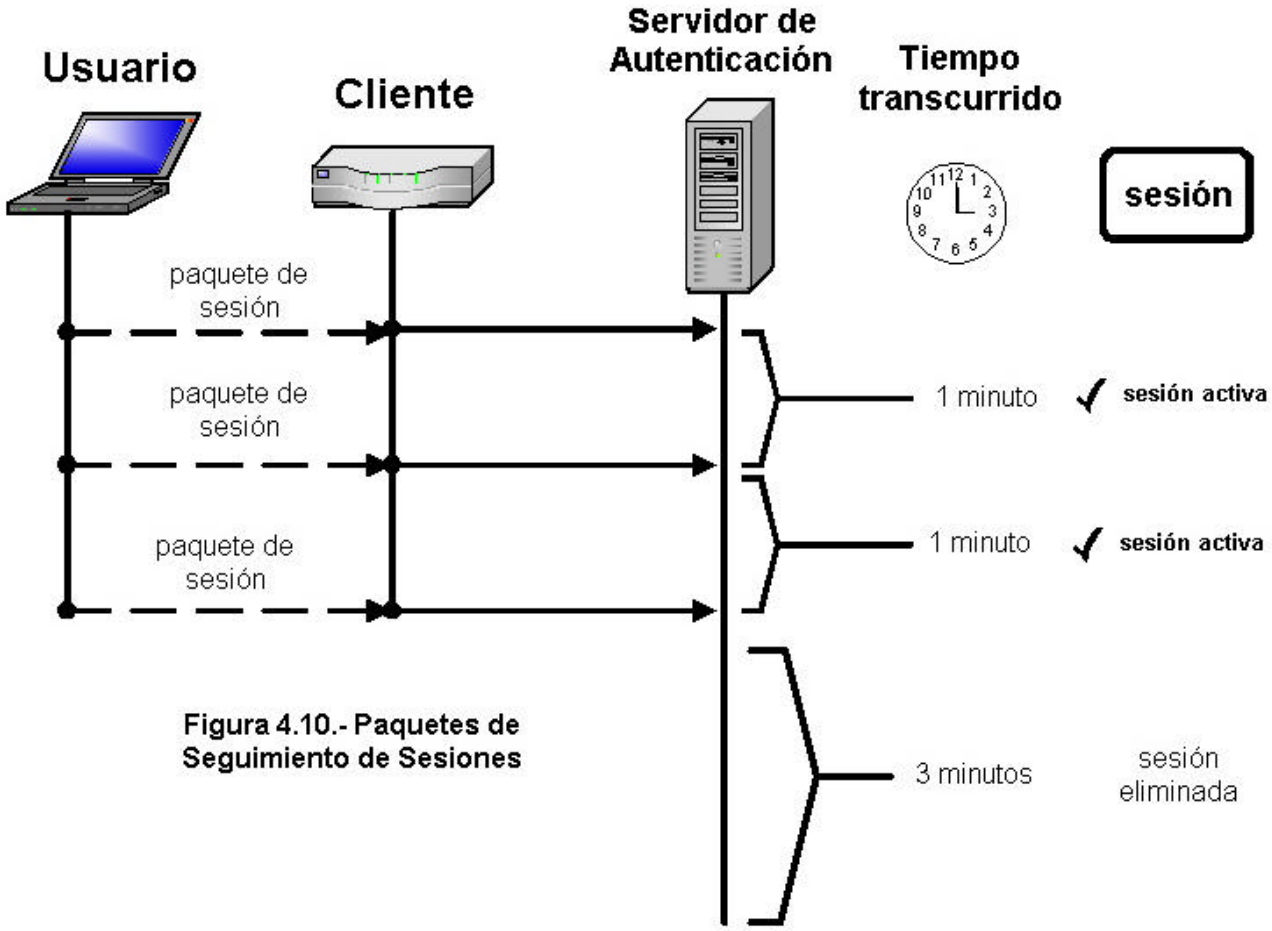


Figura 4.10.- Paquetes de Seguimiento de Sesiones

4.3.- Formato y tipo de los paquetes del protocolo

Al igual que el estándar RADIUS, el protocolo utiliza paquetes UDP para las transmisiones entre el Usuario y el Servidor de autenticación. El protocolo se comunica usando el puerto 1815, se decidió usar éste en lugar del 1812 utilizado por RADIUS, para evitar conflictos con los propios paquetes RADIUS.

Los paquetes del protocolo están formados por dos partes: una sección de encabezado y una sección de atributos. La sección de encabezado esta a su vez formada por tres secciones: una sección que contiene el código del paquete cuya longitud es constante y siempre igual a 1 byte, una sección de verificación de integridad, de longitud constante e igual a 20 bytes, en donde se coloca el resultado de aplicar el algoritmo SHA1 con llave para obtener la función hash del paquete, y por último una sección de tamaño variable, en donde se coloca el resultado de aplicar el algoritmo de cifrado AES al autenticador. La sección de atributos es de tamaño variable, y está formada por una secuencia de atributos mas adelante detallados.

Exceptuando los paquetes para el seguimiento de una sesión activa, los cuales no incluyen el autenticador cifrado.

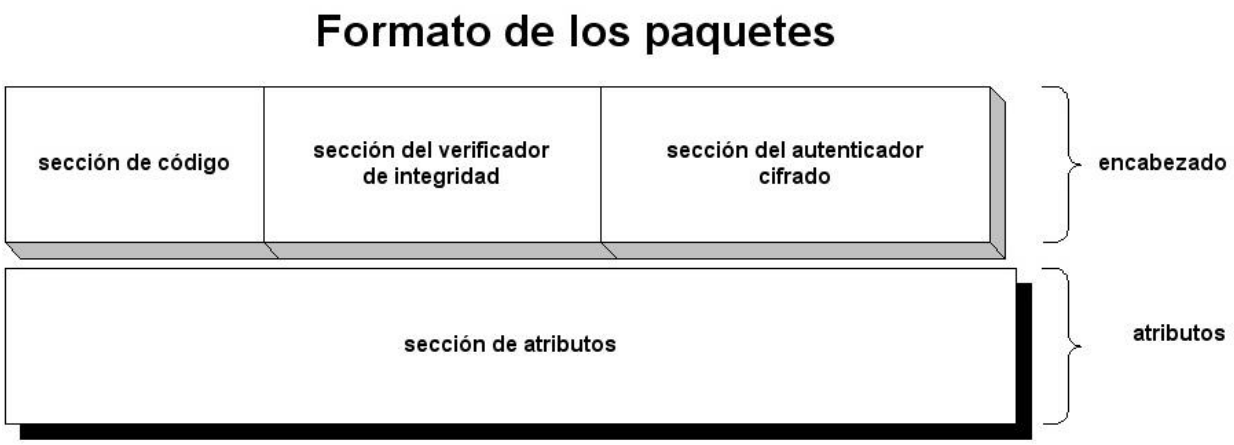


Figura 4.11

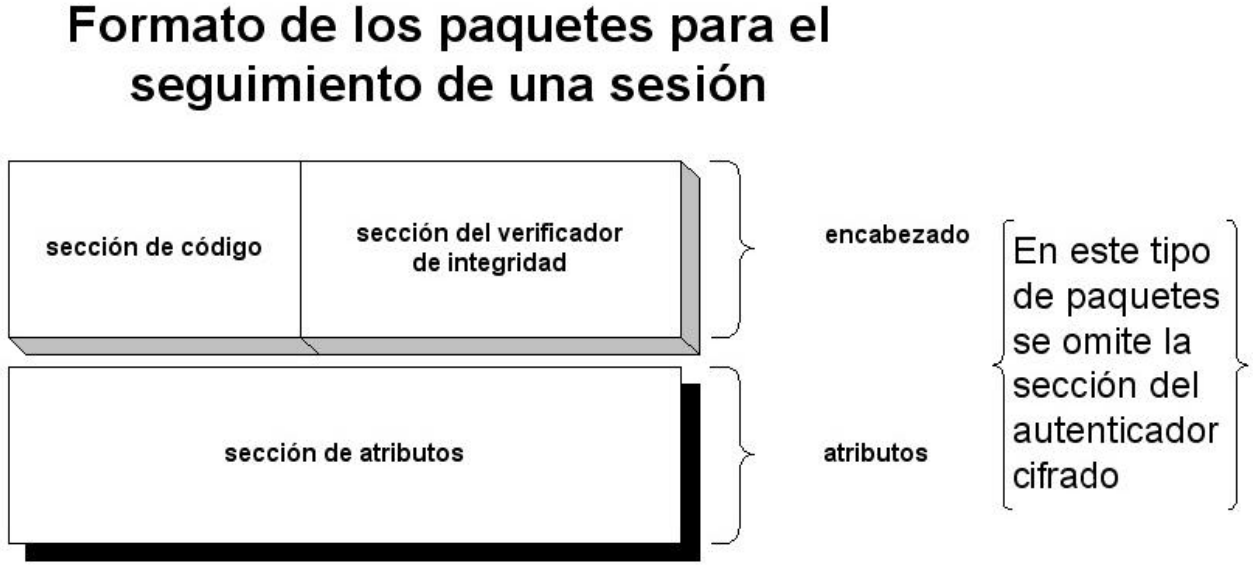


Figura 4.12

PAQUETE DE PETICIÓN DE AUTENTICACIÓN DES-MD5			
Código	**Verificador de integridad**	**Autenticador cifrado**	**Atributos**
1	16 bytes, resultantes de aplicar el algoritmo MD5 al paquete ya formado, colocando ceros en esta sección al momento de calcular la función *hash*, y reemplazando esos ceros con el resultado obtenido.	24 bytes, resultantes de aplicar el algoritmo de cifrado DES a un número generado aleatoriamente.	Se incluye el atributo login del Usuario

PAQUETE DE RESPUESTA DEL SERVIDOR DES-MD5			
Código	**Verificador de integridad**	**Autenticador cifrado**	**Atributos**
2	16 bytes, resultantes de aplicar el algoritmo MD5 al paquete ya formado, colocando ceros en esta sección al momento de calcular la función hash, y reemplazando esos ceros con el resultado obtenido.	32 bytes, resultantes de aplicar el algoritmo de cifrado DES a una trama formada de la siguiente manera: *autenticador1*|*autenticador2* donde autenticador1 es el autenticador generado por el Usuario, y autenticador2 es el autenticador generado por el Servidor.	No se incluyen atributos

PAQUETE DE RESPUESTA DEL USUARIO DES-MD5			
Código	**Verificador de integridad**	**Autenticador cifrado**	**Atributos**
3	16 bytes, resultantes de aplicar el algoritmo MD5 al paquete ya formado, colocando ceros en esta sección al momento de calcular la función hash, y reemplazando esos ceros con el resultado obtenido.	24 bytes, resultantes de aplicar el algoritmo de cifrado DES al número generado aleatoriamente por el Servidor, y enviado en el paquete de respuesta del Servidor, con un tamaño de entre 8 a 10 dígitos.	Se incluye el atributo login del Usuario

PAQUETE DE RECHAZO DE AUTENTICACIÓN DES-MD5			
Código	**Verificador de integridad**	**Autenticador cifrado**	**Atributos**

4	16 bytes, resultantes de aplicar el algoritmo MD5 al paquete ya formado, colocando ceros en esta sección al momento de calcular la función hash, y reemplazando esos ceros con el resultado obtenido.	24 bytes puestos en ceros	No se incluyen atributos

PAQUETE DE SEGUIMIENTO DE SESIÓN		
Código	**Verificador de integridad**	**Atributos**
5	16 bytes, resultantes de aplicar el algoritmo MD5 al paquete ya formado, colocando ceros en esta sección al momento de calcular la función hash, y reemplazando esos ceros con el resultado obtenido.	Se incluye el atributo login del Usuario

PAQUETE PETICIÓN DE AUTENTICACIÓN AES-SHA1			
Código	**Verificador de integridad**	**Autenticador cifrado**	**Atributos**
6	20 bytes, resultantes de aplicar el algoritmo SHA1 al paquete ya formado, colocando ceros en esta sección al momento de calcular la función *hash*, y reemplazando esos ceros con el resultado obtenido.	16 bytes, resultantes de aplicar el algoritmo de cifrado AES a un número generado aleatoriamente.	Se incluye el atributo nombre de usuario

PAQUETE DE RESPUESTA DEL SERVIDOR AES-SHA1			
Código	**Verificador de integridad**	**Autenticador cifrado**	**Atributos**
7	20 bytes, resultantes de aplicar el algoritmo SHA1 al paquete ya formado, colocando ceros en esta sección al momento de calcular la función *hash*, y reemplazando esos ceros con el resultado obtenido.	32 bytes, resultantes de aplicar el algoritmo de cifrado AES a una trama formada de la siguiente manera: *autenticador1*|*autenticador2* donde autenticador1 es el autenticador generado por el Usuario, y autenticador2 es el autenticador generado por el Servidor.	Este paquete no lleva atributos.

PAQUETE DE RESPUESTA USUARIO AES-SHA1			
Código	**Verificador de integridad**	**Autenticador cifrado**	**Atributos**
8	20 bytes, resultantes de aplicar el algoritmo SHA1 al paquete ya formado, colocando ceros en esta sección al momento de calcular la función hash, y reemplazando esos ceros con el resultado obtenido.	16 bytes, resultantes de aplicar el algoritmo de cifrado AES al número generado aleatoriamente por el Servidor, y enviado en el paquete de respuesta del Servidor, con un tamaño de entre 8 a 10 dígitos.	Se incluye el atributo login del Usuario

PAQUETE DE RECHAZO DE AUTENTICACIÓN AES-SHA1			
Código	**Verificador de integridad**	**Autenticador cifrado**	**Atributos**
9	20 bytes, resultantes de aplicar el algoritmo SHA1 al paquete ya formado, colocando ceros en esta sección al momento de calcular la función hash, y reemplazando esos ceros con el resultado obtenido.	16 bytes puestos en ceros	No se incluyen atributos

Consideraciones generales:

1. En una primera versión del protocolo se utilizó el algoritmo MD5 para calcular las funciones de *hash*, en la segunda versión se incluyó el algoritmo SHA1 para garantizar una mayor seguridad. Todos los *hash* incluyen la llave formada como MD5(password).
2. De la misma forma, en la primera versión del protocolo se utilizó como algoritmo de cifrado DES, el cual fue reemplazado en la siguiente versión por su contraparte AES.
3. El atributo login es incluido únicamente en los paquetes enviados desde el Usuario hacia el Servidor.
4. Todos los paquetes están en notación Big-Endian.

4.4.- Atributos de los paquetes

La sección de atributos está formada por una secuencia de atributos seguidos uno tras otro, de longitud variable. Los atributos dentro de un paquete siguen un formato específico, y para cada atributo que se quiera incluir, se debe seguir el siguiente formato:

- Código del atributo, tamaño 1 byte
- Longitud del atributo, obtenido sumando el tamaño de los campos código (1 byte), longitud (1 byte) y valor (tamaño variable).
- El valor del atributo

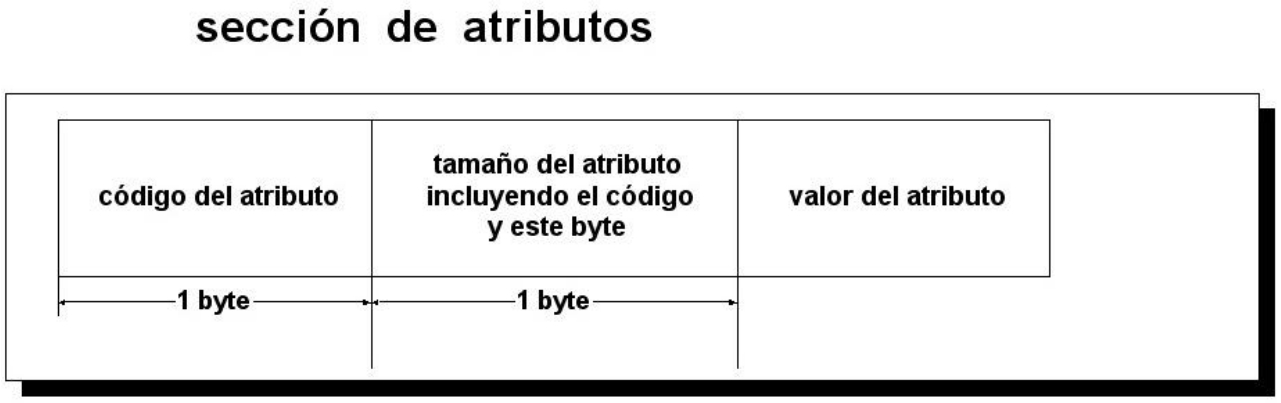


Figura 4.13

ATRIBUTO LOGIN		
Código	**Tamaño**	**Valor**
1	2 bytes + tamaño en bytes del valor	Dependiendo del Usuario

NOTA: El atributo login es incluido en todos los paquetes enviados desde el Usuario al Servidor.

4.5.- Implementación

La implementación del protocolo de acuerdo a la definición descrita en el presente capítulo, se realizó bajo las siguientes características:

Dispositivos Físicos: Los dispositivos utilizados para la implementación son detallados a continuación:

1. Punto de Acceso de la marca Avaya, modelo AP-8
2. Computadora Personal con sistema operativo Linux RedHat 9.0, para la implementación del Servidor de autenticación.
3. Computadora Personal con sistema operativo Windows XP ™ para la ejecución del Usuario
4. Tarjeta de red inalámbrica marca linksys PCI card v2.7 instalada en la computadora personal del Usuario.

Figura 4.14

Para el desarrollo de este proyecto, se utilizaron los dispositivos físicos antes mencionados, puesto que estos me fueron facilitados; no se realizaron pruebas con otras marcas o modelos, no obstante, todo dispositivo de cualquier marca o modelo que cumpla con el estándar RADIUS debe de la misma forma poder ser capaz de implementar este protocolo de autenticación, puesto que únicamente estamos haciendo uso de las características definidas para el propio estándar, siendo diferente posiblemente la forma de configurar los dispositivos, sin embargo tampoco debe haber una notable diferencia en este aspecto.

Configuración de los dispositivos:

a) Configuración del Punto de Acceso:

1. El primer paso para configurar el Punto de Acceso, es incluirlo dentro de la subred, esto es asignándole una dirección IP fija, perteneciente a la subred.
2. Configurar las opciones de modo de operación de la(s) tarjeta(s); estos modos de operación son 802.11 a, 802.11b, 802.11g o combinaciones como 802.11bg u 802.11g-wi-fi; para el caso de la implementación del protocolo, la opción 802.11bg fue seleccionada.
3. Asignar al Punto de Acceso un identificador SSID o nombre de red, este es el identificador del Punto de Acceso con los Usuarios.
4. Configurar la autenticación con RADIUS, con los valores: dirección IP del Servidor de autenticación RADIUS, puerto de comunicaciones con RADIUS, secreto compartido, tiempo de respuesta en segundos para desechar un paquete, número máximo de retransmisiones de los paquetes RADIUS.

La siguiente tabla muestra los valores utilizados en la implementación del protocolo

Opción	valor
Dirección IP	132.248.127.98
Modo de operación	802.11bg
identificador SSID	cempolancowireless2
Dirección IP Servidor RADIUS	132.248.127.99
Puerto de comunicaciones	1812
Secreto compartido	proyecto
Tiempo de respuesta en segundos	10
Máximo número de retransmisiones	0

Una vez habilitada la opción que le indica al Punto de Acceso autenticar usando el Servidor RADIUS, no es necesario realizar ningún otro proceso de configuración. El Punto de Acceso provee una interfaz gráfica en forma de páginas web para el monitoreo y verificación de los Usuarios asociados al Punto de Acceso; por medio de esta interfaz, se puede tener conocimiento de todas las estadísticas de información intercambiada entre el Punto de Acceso y el Servidor de autenticación.

b) Configuración del Servidor de autenticación:

1. En el Servidor de autenticación se instaló *freeradius* [11] únicamente para realizar algunas pruebas, pero no forma parte de los objetivos del proyecto, ni se documenta en este trabajo de tesis nada respecto a él.
2. Debido a que el protocolo se desarrolló utilizando java™ [12] como lenguaje de programación, fue necesaria la instalación de la máquina virtual en el Servidor.
3. Como manejador de base de datos se utilizó *MySQL*® [13] en su distribución libre.
4. El Servidor de autenticación también tiene que estar dentro de la subred, con una dirección IP fija, puesto que ésta se incluye en la configuración del Punto de Acceso.

c) Configuración de la computadora personal del Usuario:

1. En la computadora del Usuario también tuvo que ser instalada la máquina virtual de java™.
2. La computadora personal del Usuario, tiene que tener instalada una tarjeta de red inalámbrica.

NOTA: Los códigos fuente más relevantes de la aplicación se incluyen en la parte final de este documento, como anexo.

Lógica de negocios de la aplicación:

La lógica de negocios de la aplicación comprende la parte más importante, puesto que es ahí en donde se define toda la funcionalidad del sistema, teniendo los componentes que a continuación se detallan:

[11] El proyecto freeradius puede ser consultado en www.freeradius.org
[12] El sitio oficial de java puede ser consultado en [9]
[13] El sitio oficial del proyecto MySQL es www.mysql.com

1. *Servidor de autenticación.-* el cual fue implementado teniendo como idea principal el manejo de un Servidor "multihilo" capaz de atender cada petición que recibe (paquete UDP), generando un nuevo subproceso o "hilo" que se ejecuta independientemente de los restantes procesos y de forma concurrente a los mismos, esto garantiza que el Servidor tendrá la capacidad de atender a todas y cada una de las peticiones sin que el rendimiento del sistema se vea afectado cuando el número de peticiones sea elevado. Por otra parte, cabe destacar el hecho de que el Servidor atiende los paquetes RADIUS y los paquetes del protocolo en puertos diferentes, esto con el objeto de no saturar en un mismo puerto, comunicaciones provenientes de diferentes Clientes, puesto que los paquetes RADIUS son enviados por el(los) punto(s) de acceso, mientras que los paquetes del protocolo son enviados por los Usuarios, (RADIUS utiliza el puerto 1812 mientras que el protocolo utiliza el puerto 1815).
2. *Seguimiento de las sesiones activas de los Usuarios.-* el seguimiento de las sesiones activas de los Usuarios se realizó utilizando una referencia estática a una clase que almacena toda la información necesaria para una sesión; la cualidad de ser estática es obligada puesto que los procesos (hilos) son independientes, lo cual resulta en la generación de una sola instancia la cual será compartida por todos los procesos y además requiere el uso de restricciones de acceso a los datos compartidos, sincronización, exclusión mutua, candados, etc. Debido a que el estándar RADIUS no especifica el seguimiento de sesiones, la forma más sencilla de implementarlo fue iniciando un proceso del Usuario, el cual periódicamente envía paquetes de sesión al Servidor, y éste último al recibirlos, mantiene al Usuario en su lista de sesiones activas, cosa que por el contrario cuando un Servidor no ha tenido comunicación con el Usuario, es eliminado de la lista de sesiones activas. El Servidor ejecuta un proceso que periódicamente revisa los paquetes recibidos por los Usuarios y actualiza su lista de sesiones activas.
3. *Creación de los datagramas.-* Dadas las especificaciones tanto de RADIUS como del protocolo, la creación de los paquetes (datagramas UDP) tiene que ser coherente con estas definiciones, por lo cual se optó por diseñar una clase encargada de generar todos los diferentes tipos de paquetes, resultando en un arreglo de bytes con la secuencia, orden y formato especificados.
4. *Generación de números pseudoaleatorios.-* Puesto que el protocolo conlleva al uso y generación de números pseudoaleatorios, la forma en que estos son generados, funciona de la siguiente manera y depende completamente de la función tiempo: el principio básico para la generación de números pseudoaleatorios radica en la utilización de una semilla generadora, la cual es obtenida accesando al número total de milisegundos del sistema para el momento en el cual se hace la llamada, lo cual nos asegura que nunca se podrá utilizar dos veces la misma semilla generadora, puesto que esta es función del tiempo y cada milisegundo que transcurre el valor resultante para la semilla es diferente. Una vez calculada la semilla generadora, se van obteniendo los números aleatorios generados por dicha semilla, en dependencia directa del día de la semana, del día del mes y de la hora con precisión de segundos en que se hace la llamada, esto es si el día del mes es el 20, el día de la semana es el segundo, y la hora es 20:30:15, al realizar la suma de todos estos valores obtenemos un resultado de 20+2+20+30+15 = 87, por lo cual tenemos que descartar y desechar los primeros 87 números generados por esa semilla, y tomaremos como válido el número próximo siguiente. Los números generados son de una longitud mínima de 8 dígitos, y de una longitud máxima de 10.

5. *Almacenamiento de la información.-* Exceptuando la información referente a la lista de sesiones activas, la cual permanece en la memoria volátil del sistema operativo, el Servidor de autenticación almacena toda la restante información en tablas relacionales, usando el modelo jdbc (Java Database Connectivity) para la administración de los datos persistentes del sistema.
6. *Criptografía del sistema.-* Un aspecto fundamental en cualquier sistema de seguridad, es la parte criptográfica, en el protocolo se incluyó un módulo para resolver este problema: tanto el estándar RADIUS como el protocolo utilizan por un lado, criptografía simétrica para el cifrado de los autenticadores, y verificación de integridad por medio de funciones de hash. En cuanto a la criptografía simétrica, se implementaron clases para los algoritmos DES, AES , así como para los algoritmos MD5 y SHA1 para el cálculo de la función *hash* de una secuencia de bytes (datagrama).

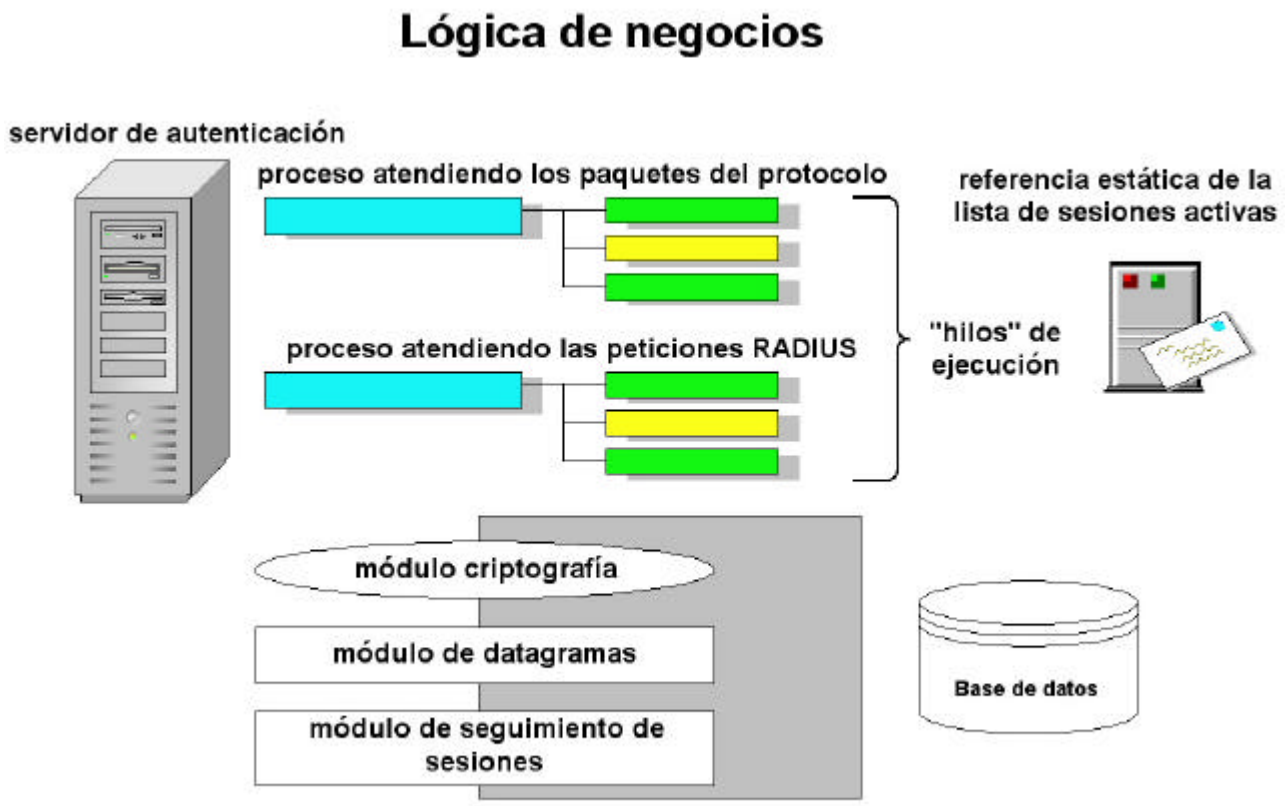


Figura 4.15

Interfaz gráfica de Usuario:

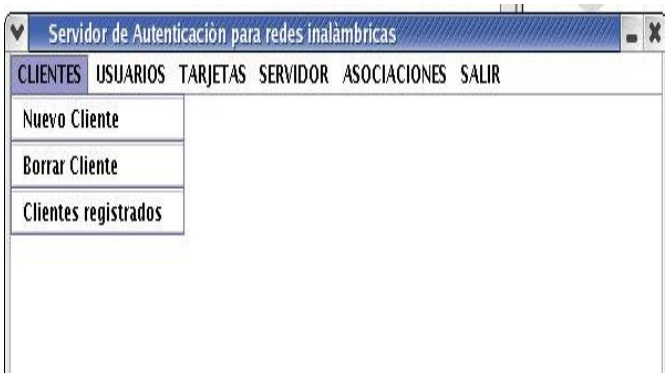
La interfaz gráfica del Usuario comprende los componentes visuales (pantallas, ventanas, botones, etc) por medio de los cuales el sistema interactúa con el Usuario o con el administrador del Servidor.

Componentes que forman la interfaz gráfica del sistema:

1. Pantalla principal, contiene el menú de opciones de administración del Servidor, y es la pantalla de inicio del sistema (Servidor).
2. Pantalla del Usuario, la cual permite al Usuario introducir los valores para su nombre de Usuario o login y su contraseña.
3. Pantalla de alta de Clientes; permite al administrador registrar un nuevo Cliente (Punto de Acceso) en la base de datos.
4. Pantalla de baja de Clientes; permite al administrador eliminar un registro de Cliente
5. Pantalla de lista de Clientes; despliega todos los registros en la base de datos para los Clientes (puntos de acceso).
6. Pantalla de altas de Usuarios; permite al administrador del sistema registrar un nuevo Usuario en el Servidor.
7. Pantalla de baja de Usuarios; permite al administrador eliminar un registro de Usuario de la Base de Datos
8. Pantalla de lista de Usuarios; despliega la relación de todos los Usuarios registrados en el sistema.
9. Pantalla de alta de tarjetas; permite al administrador del Servidor, registrar una tarjeta de red.
10. Pantalla de baja de tarjetas; permite al administrador del sistema, eliminar el registro de una tarjeta
11. Pantalla de lista de tarjetas; despliega la relación de tarjetas registradas en el sistema.
12. Pantalla de asociaciones; permite al administrador del sistema establecer asociaciones entre Usuarios y tarjetas registradas.
13. Inicio del Servidor; opción del menú que comienza la ejecución del demonio de MySQL, y la ejecución de los hilos para atender las peticiones (Servidor de autenticación).
14. Detener el Servidor; opción del menú que detiene la ejecución de todos los hilos y por consiguiente del Servidor de autenticación.

A continuación se muestran las pantallas del sistema:

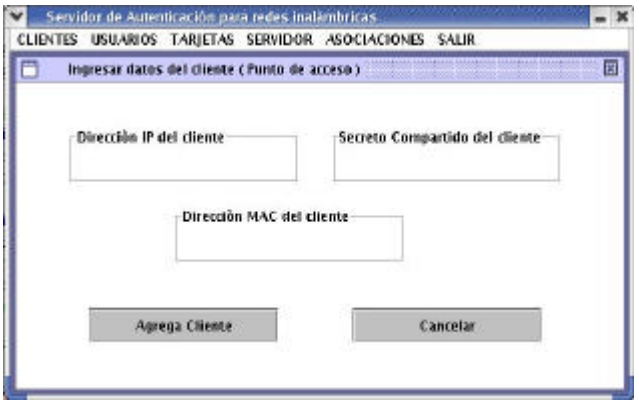
1.- Pantalla Principal

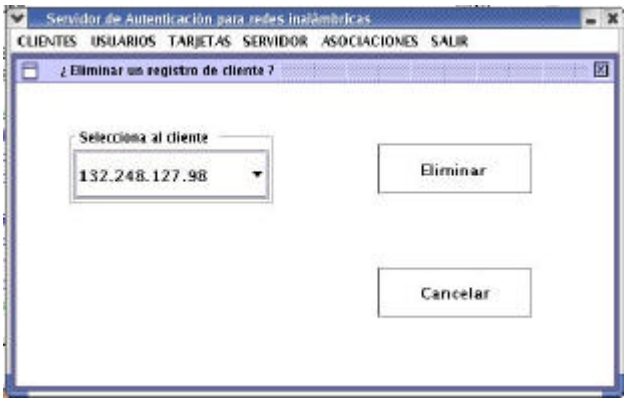


2.- Pantalla de los Usuarios

3.- Pantalla Alta de Clientes

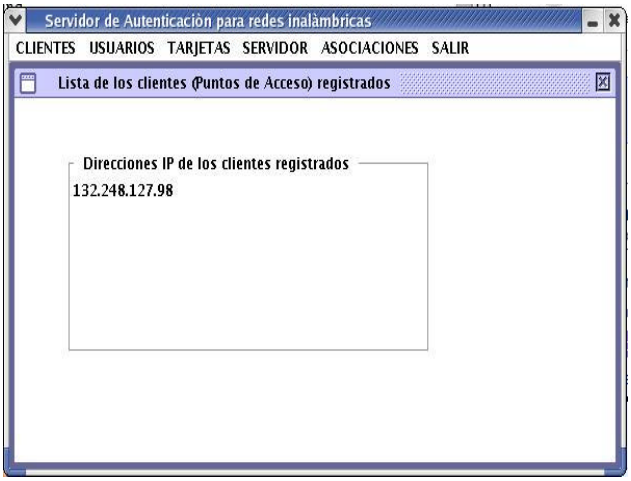
4.- Pantalla de Baja de Clientes

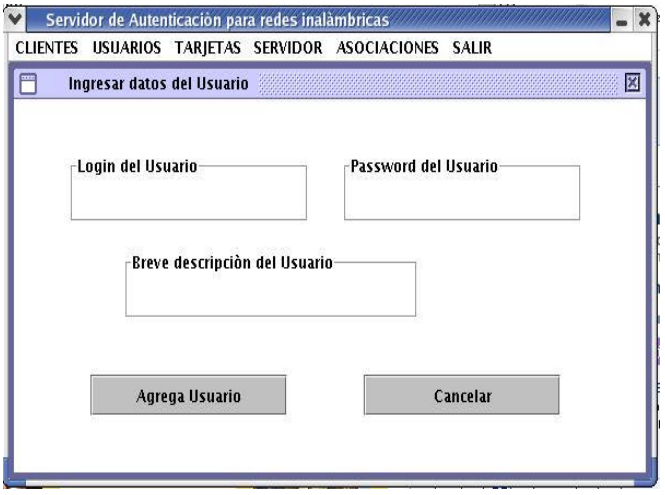




5.- Pantalla Lista de los Clientes registrados

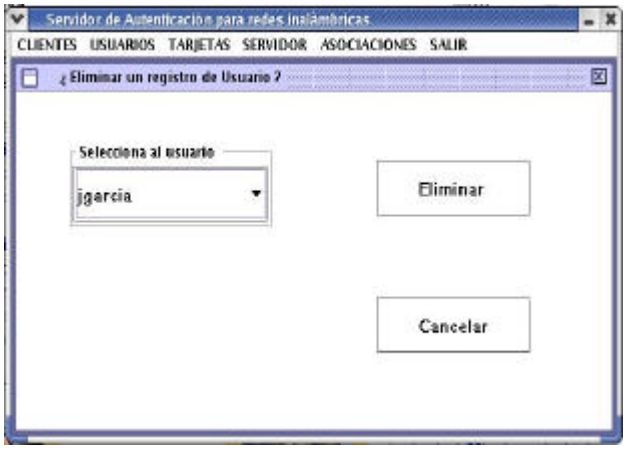
6.- Pantalla Altas de Usuarios

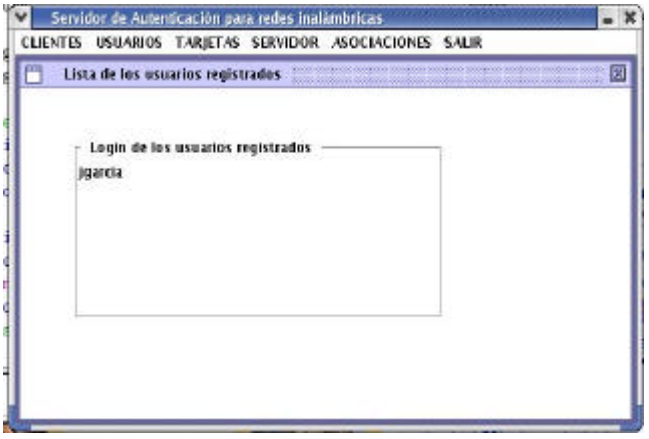




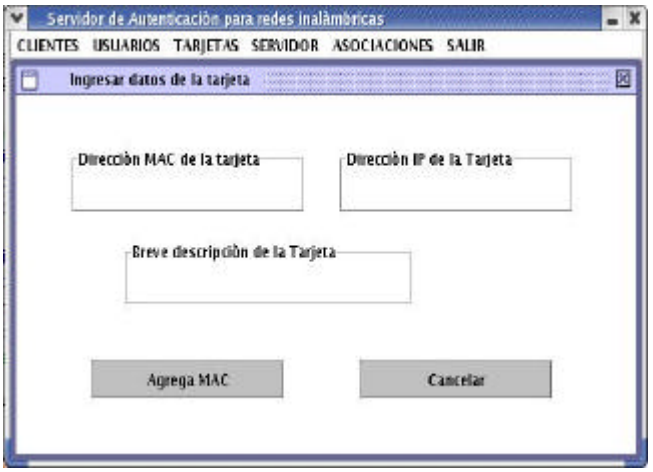
7.- Pantalla Bajas de Usuarios

8.- Pantalla Lista de Usuarios

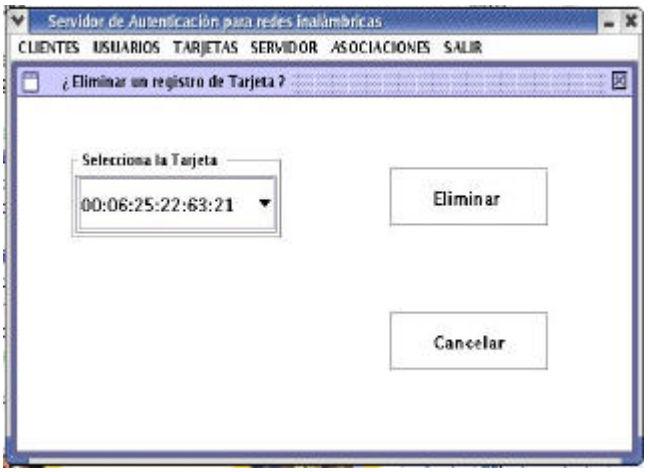




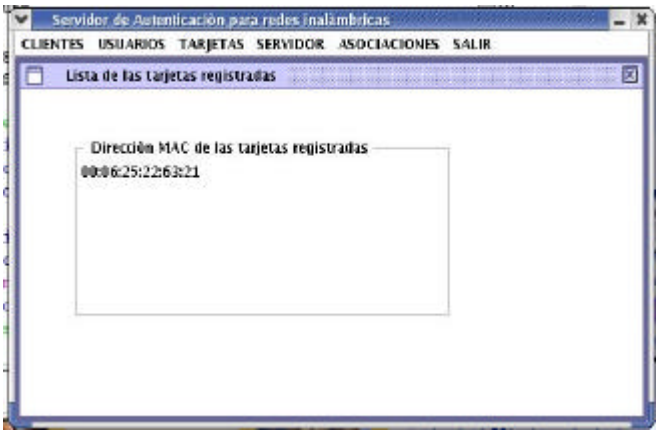
9.- Pantalla Alta de Tarjetas



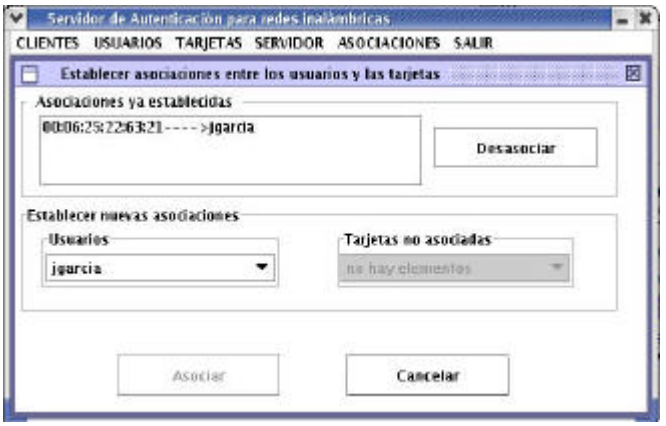
10.- Pantalla Baja de Tarjetas



11.- Pantalla Lista de las Tarjetas registradas



12.- Pantalla de Asociaciones



Descripción de las tablas:

Todos los campos de las tablas son de tipo VARCHAR, (cadenas de caracteres).

TABLA USUARIOS		
login	**contraseña**	**descripción**

TABLA TARJETAS		
MAC	**IP**	**descripción**

TABLA CLIENTES		
IP	**secreto**	**MAC**

TABLA RELACION MAC-USUARIO	
MAC	**login**

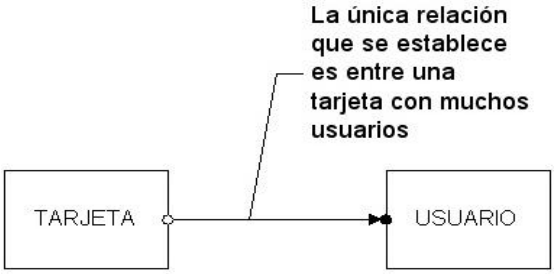


Figura 4.16

Pruebas y escenarios de ataques.- El objetivo principal de este trabajo de tesis, fue el de robustecer la funcionalidad del Servidor de autenticación RADIUS mediante el uso de un protocolo que permita corregir las vulnerabilidades que se presentan con el protocolo original. El protocolo corrige las vulnerabilidades documentadas en la sección 3.10 referentes a la falsificación de direcciones MAC, las pruebas realizadas son a continuación documentadas:

1. El primer escenario se presenta cuando el atacante intenta autenticarse falsificando la dirección MAC de un Usuario registrado (MAC spoofing), cuando éste último no se encuentra conectado al sistema. En este tipo de ataque, la dirección IP del atacante puede ser cualquiera. Este ataque tiene éxito parcial en cuanto a la autenticación de la dirección MAC se refiere, pero al no saber el atacante la contraseña del Usuario, no le es posible ejecutar el protocolo, por lo cual en el momento de ser nuevamente solicitado el paquete de petición de acceso, el Servidor de

autenticación contesta con un paquete de rechazo, negando toda posibilidad de autenticación de dicho atacante. La siguiente figura ilustra este escenario:

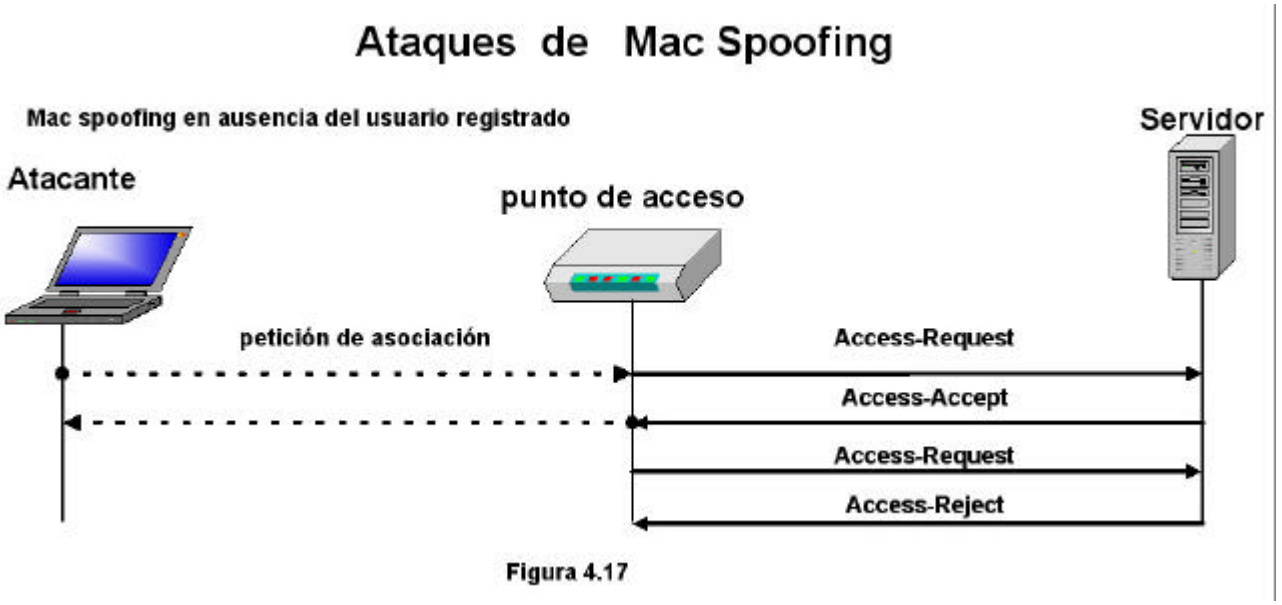


Figura 4.17

2. El segundo escenario posible, se presenta cuando el atacante intenta autenticarse falsificando la dirección MAC de un Usuario registrado (MAC *spoofing*), pero estando el Usuario conectado al sistema, esto es cuando el Usuario ya ha sido autenticado con éxito. En este tipo de ataque, el Intruso puede optar por 2 situaciones; la primera se presenta cuando el atacante utiliza una dirección IP diferente a la del Usuario válido en el sistema, por lo cual requiere ser asociado como un nuevo Usuario, esta situación es muy fácil de controlar, puesto que para este escenario, el Servidor de autenticación se apoya en el seguimiento de sesiones implementado, por lo cual automáticamente en el momento en el que le llega la petición de acceso del atacante, y detecta que existe una sesión activa para esa dirección MAC, el Servidor de autenticación contesta un paquete de negación de acceso, imposibilitando completamente al atacante, el uso de los servicios, o en su defecto la autenticación, la figura 4.18 ilustra este escenario. La segunda situación se vuelve más compleja, cuando el atacante opta por falsificar tanto la dirección MAC como la dirección IP del usuario, las cuales puede ser capaz de obtener simplemente utilizando un programa conocido como *sniffer*[14]. Para este escenario, no existe una defensa posible, ya que los paquetes son transmitidos a través del medio inalámbrico y únicamente pueden ser direccionados desde el remitente hacia el destinatario por medio de la dirección IP. Es en este escenario cuando entra en juego el protocolo IP, puesto que para cualquier tipo de red sea inalámbrica o cableada, "montada" sobre IP, se cumple que al haber dos dispositivos físicos con la misma dirección IP y la misma dirección MAC, los paquetes tienden a perderse y no llega la secuencia completa ni a una ni a otra computadora, provocando

[14] Es un programa utilizado para monitorear y analizar el tráfico en una red, puede ser utilizado para capturar lícita o ilícitamente los datos transmitidos.

una situación en la que ninguna de las dos computadoras llega a intercambiar paquetes de forma correcta.

3. Un tercer intento de ataque se da cuando el intruso intenta obtener la contraseña del usuario interceptando los paquetes intercambiados durante el protocolo. Para contrarrestar este escenario de ataque, los algoritmos utilizados fueron aquellos que presentan una mayor robustez (AES y SHA1 en sustitución de MD5 y DES), y a menos que estos algoritmos pudieran ser vulnerados (por un ataque de fuerza bruta o cualquier otra forma de criptoanálisis), entonces la seguridad del protocolo se vería afectada.

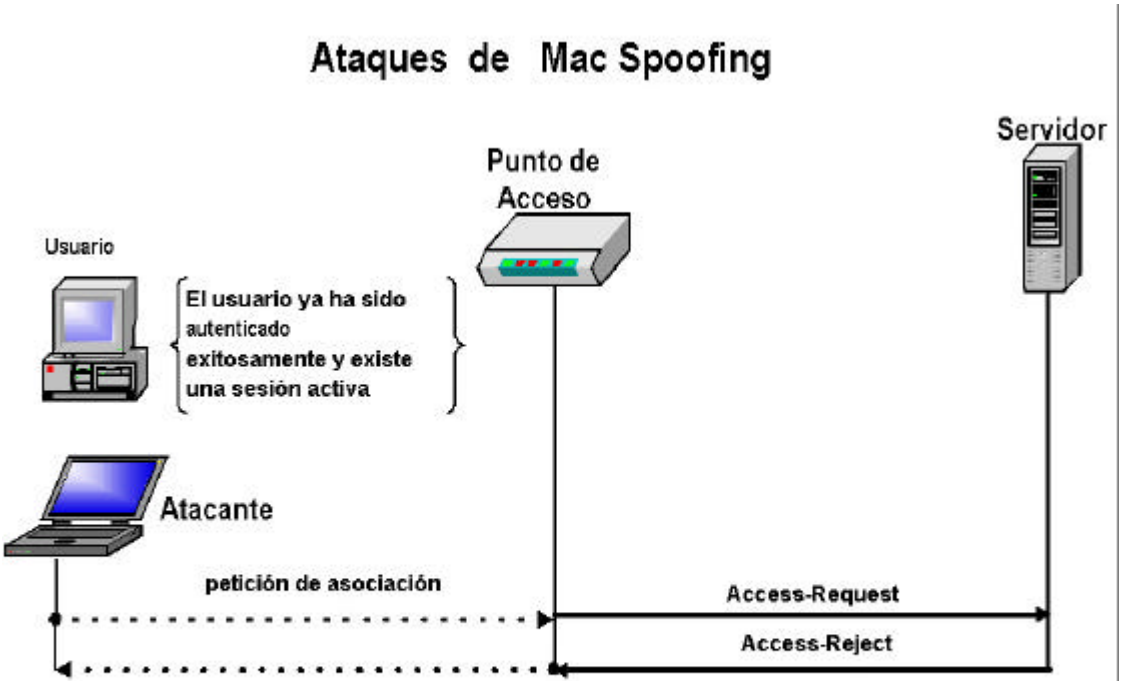


Figura 4.18
Mac spoofing en presencia del usuario registrado

Pruebas realizadas.

Descripción del primer escenario de ataque.- Se tiene que en el Servidor de Autenticación existe una relación entre la tarjeta 00:06:25:22:5d:57 con el Usuario jgarcia (como lo muestra la figura 4.19), con lo cual esta tarjeta es válida para este usuario.

El atacante que originalmente tiene la dirección MAC 00:06:25:22:63:21 ha falsificado su dirección como se detalla en la sección 3.10 y en las figuras 3.4, 3.5 y 3.6, para ahora tener la dirección 00:06:25:22:5d:57.

El ataque tiene éxito parcial porque la dirección MAC es válida, pero posteriormente el servicio es negado al usuario, la figura 4.20 ilustra el hecho de que el servicio le es interrumpido al usuario.

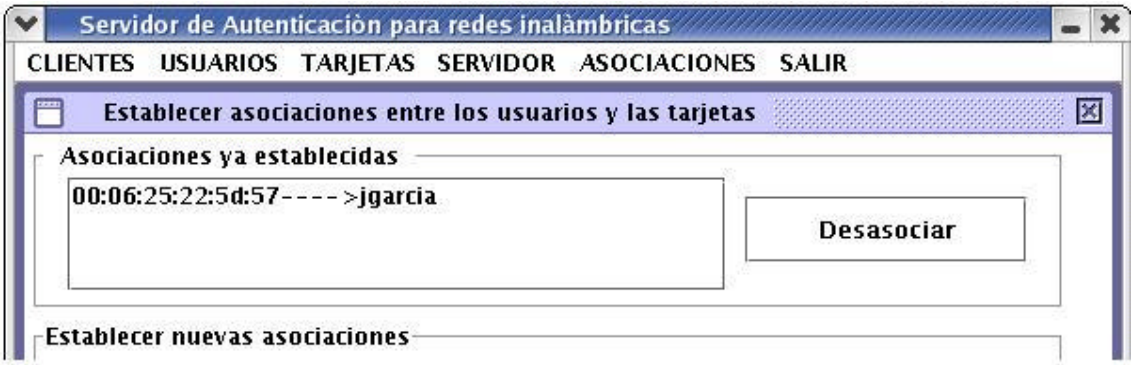


Figura 4.19

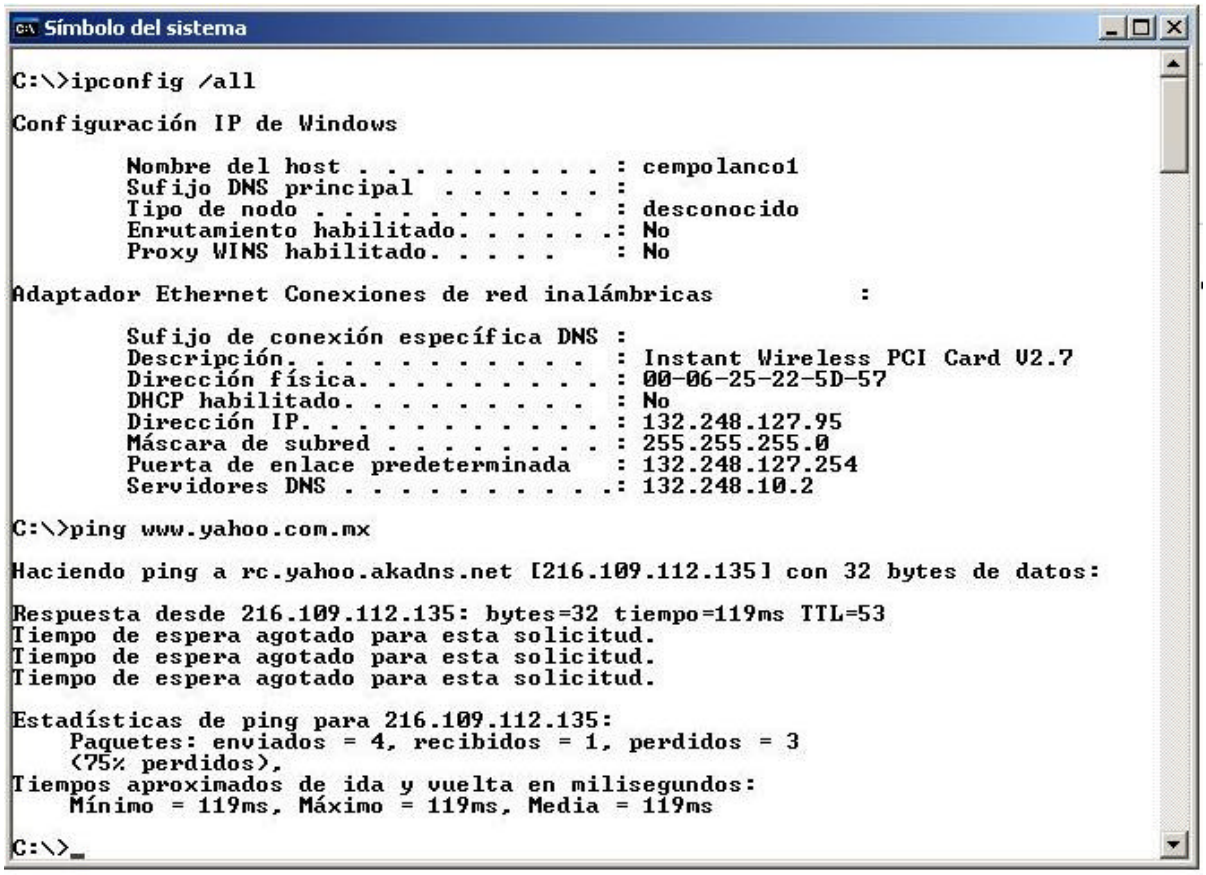


Figura 4.20

A continuación se muestra la salida del archivo de historial del Servidor de Autenticación, en la cual se puede observar que en el instante de tiempo minuto 57, segundo 26, el Servidor de Autenticación envía al Punto de Acceso un Paquete RADIUS de Aceptación (código 2), incluyendo un atributo con código número 27 (indicando la longitud de la sesión) y un valor de 30 (segundos). Al minuto 58 con 5 segundos, se recibe un paquete RADIUS de Petición de Acceso, el cual es contestado con una negación de servicio (paquete RADIUS con código 3).

```
[root@localhost proyecto]# java Inicio
Servidor de autenticaciÃ²n escuchando en el puerto 1815
Servidor RADIUS escuchando en el puerto 1812

Datagrama recibido desde el A.P, en el tiempo: 57 : 25
byte 0 : cÃ²digo = 1
byte 1 : identificador = 3
bytes 2 y 3 : longitÃ¹d del paquete = 0 69
bytes 4 -19 (16 bytes): autenticador del paquete 0 0 42 -114 0 0 6 -88 0
0 42 88 0 0 36 86

Atributo nÃ¹mero 1
longitÃ¹d del atributo 19
48 48 58 48 54 58 50 53 58 50 50 58 53 100 58 53 55
Atributo nÃ¹mero 2
longitÃ¹d del atributo 18
45 26 -51 -78 -64 13 -82 -92 -118 89 -39 -67 -51 -36 -94 -17
Atributo nÃ¹mero 4
longitÃ¹d del atributo 6
-124 -8 127 98
Atributo nÃ¹mero 5
longitÃ¹d del atributo 6
0 0 0 0

Datagrama enviado al Access Point en el instante: 57 : 26
byte 0 : cÃ²digo = 2
byte 1 : identificador = 3
bytes 2 y 3 : longitÃ¹d del paquete = 0 26
bytes 4 -19 (16 bytes): autenticador del paquete 54 76 32 -66 -71 2 -67 -
89 -44 -14 19 -77 84 -111 -111 -72

Atributo nÃ¹mero 27
longitÃ¹d del atributo 6
0 0 0 30
estado del usuario despues de haber validado:MAC_OK

Datagrama recibido desde el A.P, en el tiempo: 58 : 5
byte 0 : cÃ²digo = 1
byte 1 : identificador = 4
bytes 2 y 3 : longitÃ¹d del paquete = 0 69
bytes 4 -19 (16 bytes): autenticador del paquete 0 0 113 -117 0 0 112 16
0 0 61 -3 0 0 85 90

Atributo nÃ¹mero 1
longitÃ¹d del atributo 19
```

```
48 48 58 48 54 58 50 53 58 50 50 58 53 100 58 53 55
Atributo nÃ¹mero 2
longitÃ¹d del atributo 18
-24 -124 -103 -14 108 -50 122 -112 118 -80 -30 -66 7 80 52 -79
Atributo nÃ¹mero 4
longitÃ¹d del atributo 6
-124 -8 127 98
Atributo nÃ¹mero 5
longitÃ¹d del atributo 6
0 0 0 0
estado de la transaccion al final: MAC_OK

Datagrama enviado al Access Point en el instante: 58 : 5
byte 0 : cÃ²digo = 3
byte 1 : identificador = 4
bytes 2 y 3 : longitÃ¹d del paquete = 0 20
bytes 4 -19 (16 bytes): autenticador del paquete 90 -117 89 111 -95 86 -
39 99 -43 -35 50 102 -53 -34 124 117
```

El estado final de la transacción es MAC_OK, indicando que únicamente se ha validado la dirección MAC, ya que efectivamente el protocolo no se llevó a cabo.

Descripción del segundo escenario de ataque.- En este escenario, existe un usuario registrado en el sistema cuya dirección MAC es la 00:06:25:22:5d:57 y su dirección IP es la 132.248.127.95, el cual ya ha sido autenticado con éxito; por otro lado, hay un atacante cuya dirección MAC es la 00:06:25:22:63:21 y utiliza la dirección IP 132.248.127.96, como lo ilustra la figura 4.18.

Primer caso.- El atacante intenta autenticarse en el sistema, y suplanta la dirección MAC del usuario 00:06:25:22:5d:57 (procedimiento detallado en la sección 3.10), e inicia el proceso de asociación con el Punto de Acceso. Debido a que ya existe una sesión activa para esa dirección MAC en el Servidor de Autenticación (la del usuario), entonces el Servidor de Autenticación envía un paquete RADIUS de rechazo y se le niega el servicio al atacante, situación que con el estándar RADIUS hubiera terminado como una aceptación debido a que como ya se ha mencionado en múltiples ocasiones, RADIUS no implementa el seguimiento de sesiones.

Segundo caso.- El atacante y el usuario tienen los mismos valores para las direcciones MAC e IP que en el caso anterior. De nueva cuenta el atacante falsifica su dirección MAC y suplanta la del usuario, y hace lo mismo con la dirección IP, por lo cual se tiene que tanto el usuario como el atacante tienen la dirección IP 132.248.127.95 y la dirección MAC 00:06:25:22:5d:57, este caso representa la principal vulnerabilidad en el proceso de autenticación, en el cual el protocolo falla debido a que no existe defensa alguna contra este tipo de ataque. El resultado es que la red se satura con paquetes provenientes de ambas computadoras, logrando el atacante concluir con un ataque de denegación de servicios.

Descripción de un posible tercer escenario de ataque.- En este escenario, un atacante seguramente intentaría recuperar la mayor cantidad de mensajes posible, para tratar de vulnerar la seguridad de los algoritmos SHA1 y AES. Existen dos posibilidades para un atacante de este tipo.
La primera consiste en atacar directamente al algoritmo AES, esto es cuando el atacante logra interceptar los datagramas del protocolo y recuperar los autenticadores:

- El autenticador contenido en el paquete de Petición de Autenticación
- El autenticador del paquete de Respuesta del Servidor
- El autenticador del paquete de Respuesta del Usuario

Una vez recuperados estos tres autenticadores, se puede aplicar un ataque de fuerza bruta para intentar recuperar la contraseña del usuario, debido a que los tres han sido cifrados utilizando como llave de cifrado la contraseña del usuario.

La segunda opción consiste en atacar el algoritmo SHA1, dado que el atacante puede interceptar alguno de los paquetes, la sección del verificador de integridad ha sido calculada con el paquete y la contraseña incluida al final del mismo, por lo cual un ataque por fuerza bruta consistiría en que conocido el paquete intentar aplicar el algoritmo SHA1 con una variedad de posibles contraseñas hasta obtener el mismo *hash* contenido en la sección de integridad.

Vulnerabilidades del protocolo.- El protocolo puede llegar a ser vulnerable en los siguientes casos:

1.- Debido a que el protocolo utiliza dos autenticadores, (un autenticador de posesión y otro de conocimiento descritos en la sección 2.2), una vulnerabilidad que puede experimentar se presenta cuando el Usuario ha comprometido tanto el dispositivo físico (tarjeta de red), como su contraseña personal, y el atacante ha podido ser capaz de tener acceso a ambos autenticadores (incluso basta con que el atacante obtenga la contraseña del Usuario, puesto que la dirección MAC de la tarjeta puede ser falsificada con relativa facilidad), éste representa el peor escenario posible, para el cual el protocolo no tiene defensa alguna.

2.- Otra vulnerabilidad, que no es propia ni exclusiva de este protocolo sino de las redes inalámbricas en general, son los ataques por denegación de servicio, ya que un atacante puede dados los 2 casos de falsificado de dirección MAC mencionados anteriormente, tener para el primero de ellos, un lapso de tiempo (30 seg.) para poder enviar una gran cantidad de paquetes e intentar saturar la red, antes de que sea desasociado, y para el segundo de los casos el atacante puede hacer lo mismo pero con un lapso mayor de tiempo.

4.6.- Tolerancia a fallas

La implementación del protocolo realmente no tiene un enfoque en cuanto a la tolerancia a fallas se refiere. Debido a la naturaleza misma no orientada a conexión del protocolo UDP, es muy factible que en ocasiones los datagramas pierdan sincronía o incluso lleguen a perderse, situación que obligaría a una retransmisión del paquete para poder asegurar que el protocolo se lleve a cabo. Sin embargo, esto implica también la implementación de un algoritmo para la detección y corrección de errores

(principalmente datagramas perdidos o fuera de sincronía), cosa que para los fines de esta tesis no se consideró; la solución posible seria la inclusión de un algoritmo para la retransmisión de paquetes después de un cierto lapso definido de tiempo, solución que implementan los Puntos de Acceso, en los cuales esta opción puede ser configurada para establecer un número máximo de retransmisiones y también un intervalo definido de tiempo para esperar una respuesta del receptor.

Para esta implementación del protocolo, cuando se presenta una falla por un datagrama perdido o fuera de sincronía, es necesario volver a comenzar de nuevo todo el proceso de autenticación. Las estadísticas del porcentaje de paquetes perdidos en una prueba realizada, con un cierto tiempo de diferencia entre el envío de cada uno de los paquetes (150 milisegundos) se detallan en la siguiente tabla:

Número de iteración	Total de paquetes enviados	Total de paquetes recibidos	Porcentaje de paquetes recibidos	Porcentaje de paquetes perdidos
1	100	94	94%	6%
2	100	96	96%	4%
3	100	100	100%	0%
4	100	100	100%	0%
5	100	97	97%	3%
6	100	94	94%	6%
7	100	94	94%	6%
8	100	93	93%	7%
9	100	96	96%	4%
10	100	94	94%	6%
11	100	94	94%	6%
12	100	95	95%	5%
13	100	96	96%	4%
14	100	91	91%	9%
15	100	95	95%	5%

Arrojando un promedio de paquetes perdidos de 4.73 % y redondeando 5 % del total de los paquetes perdidos. No obstante, en pruebas realizadas con un envío constante de datagramas, esto es sin incluir un retardo entre cada uno de ellos, la totalidad de los paquetes fueron recibidos, esto me hace concluir que la causa de las pérdidas de paquetes esta directamente relacionada con la pérdida de sincronía entre el envío de los mismos.

CONCLUSIONES

Una vez terminada la implementación del Servidor de autenticación y del protocolo, me permito concluir lo siguiente:

- El objetivo fundamental de este trabajo de tesis, fue el de establecer una implementación robusta para el proceso de autenticación, corrigiendo la vulnerabilidad del Servidor RADIUS en cuanto a la falsificación de direcciones MAC se refiere. Considero que la robustez del protocolo comparado con los restantes definidos, es el hecho del uso de dos autenticadores de diferentes categorías, uno de posesión y otro de conocimiento, ya que la mayoría de los mecanismos de autenticación hacen uso exclusivamente de uno de ellos.
- Debido a la gran cantidad de dispositivos (puntos de acceso y tarjetas de red inalámbricas) de diferentes marcas existentes en el mercado, se tomó como base el Servidor RADIUS ya que es el estándar implementado en casi todos ellos en prácticamente todo el mundo, con lo cual se buscó garantizar la portabilidad del protocolo con cualquier dispositivo que soporte RADIUS; desgraciadamente no se pudieron llevar a cabo pruebas con dispositivos de otras marcas para corroborar este hecho, solo puedo concluir que **la portabilidad debe estar garantizada siempre y cuando un Punto de Acceso soporte el uso de un Servidor de autenticación RADIUS.**
- Las nuevas tecnologías desarrolladas (como WPA, WPA2), proveen mecanismos de seguridad muy robustos, por ejemplo la utilización de nuevos algoritmos criptográficos como AES, aunque por otro lado, el proceso de reemplazar todos los equipos anteriores con sus contrapartes nuevas, es bastante costoso y tedioso, por lo cual siempre tiene que haber opciones para que todos esos equipos sean actualizados de alguna forma, sin tener que ser reemplazados por completo. Este trabajo de tesis, estuvo dirigido a la actualización de esos equipos que todavía tienen implementado RADIUS como mecanismo de autenticación.
- Actualmente, el Comité IEEE está trabajando en un nuevo estándar relacionado con redes inalámbricas de corta distancia, redes de área local y metropolitana, en el que seguramente se verán corregidas las vulnerabilidades presentadas por el estándar anterior. Los trabajos, avances y resultados del nuevo estándar 802.15 pueden ser consultados en www.ieee802.org/15/ .
- Un aspecto importante que considero resaltar, es que a diferencia de la mayoría de los protocolos que involucran contraseñas las cuales son enviadas en el mensaje de autenticación, en este protocolo la contraseña nunca viaja por el canal inseguro entre el Usuario y el Servidor, únicamente se utiliza para el cifrado de paquetes y permanece siempre en memoria volátil de los equipos del Usuario por un lado, y del Servidor por el otro, modificando el antiguo esquema de autenticación consistente en el envío del nombre de Usuario y la contraseña en el mensaje, haciéndolo más seguro.

GLOSARIO DE TERMINOS

- **AES (Advanced Encryption Standard).-** Algoritmo de cifrado simétrico, utiliza llaves de cifrado de 128 bits, para mayor información consultar la referencia [21].

- **Autenticador.-** Un autenticador se define como un instrumento el cual pretende garantizar la identidad de quien lo porta, ejemplos de autenticadores pueden ser una credencial o tarjeta, una contraseña, una huella dactilar, etc.

- **DES (Data Encryption Standard).-** Algoritmo de cifrado simétrico, utiliza llaves de 56 bits, actualmente ha sido sustituido por su contraparte AES.

- **Dirección MAC.-** Toda NIC (Network Interface Card o tarjeta adaptadora de red), independientemente del medio que utilice, ya sea cable, aire etc, dispone de un identificador llamado dirección MAC. Este identificador "trabaja" en la capa 2 –enlace de datos- del modelo OSI, y es un identificador exclusivo para cada NIC o tarjeta. Esta MAC está formada por 48 bits, de los cuales los 24 primeros identifican al fabricante (estos bits son otorgados por la IEEE a todos los fabricantes de tarjetas), mientras que los 24 siguientes son el número de serie / referencia que el propio fabricante le ha asignado a la NIC o tarjeta; por ello se supone que no existen dos tarjetas con la misma MAC, o no deben existir, aunque en el mercado existen tarjetas de red a las cuales se les puede cambiar la dirección MAC; no obstante, existen una gran variedad de programas "spoofer" con los cuales una tarjeta puede suplantar una dirección MAC diferente a la suya, haciendo que el mecanismo de autenticación basado en el filtrado de direcciones mac, resulte completamente vulnerable.

- **DSSS (Direct Sequence Spread Spectrum).-** Es una tecnología de transmisión de datos, en donde las señales enviadas son combinadas con una sequencia de bits o código, que dividen los datos de acuerdo al rango de dispersión.

- **EAP (Extensible Authentication Protocol).-** Es un mecanismo mediante el cual se lleva a cabo un intercambio de información entre un Usuario y el Servidor de Autenticación; dispositivos intermedios como Puntos de Acceso y Servidores *Proxy* no toman parte en la conversación. 802.1X utiliza EAP como el marco de referencia para el proceso de autenticación.

- **EAPOL (EAP OVER LAN).-** Es un protocolo utilizado para el intercambio de mensajes específicos del estándar 802.1X entre el Punto de Acceso y el Usuario. 802.1X define un estándar para encapsular los mensajes del protocolo EAP, de tal forma que estos puedan ser manejados directamente por el servicio de una LAN.

- **FHSS (Frecuency Hopping Spread Spectrum).-** Es un espectro de dispersión de ondas de radio. FHSS es una tecnología de transmisión inalámbrica, en donde las señales de datos son

moduladas en una secuencia aleatoria pero predecible, de una frecuencia a otra como función del tiempo, sobre una banda ancha de frecuencias.

- **Función de Hash.-** Es un algoritmo basado en operaciones matemáticas las cuales transforman una secuencia finita de datos, en otra secuencia pero de longitud fija para cualquier tamaño de secuencia de datos de entrada. La secuencia obtenida como resultado se puede entender como la "huella digital" del mensaje. Dado que no existe límite en la longitud de las secuencias de entrada, existe una cantidad infinita de mensajes posibles y por otra parte una cantidad finita de resultados posibles para la función de *hash*. Esto inevitablemente significará que existirán secuencias distintas que arrojen el mismo resultado tras ser evaluadas por la función de *hash*, lo que en la jerga se denomina **colisión** y será lo que deberá evitarse por todos los medios posibles a la hora de diseñar una función de *hash* utilizada con fines de seguridad (*hash* criptográfico).

  Es por ésto que para que podemos calificar como "libre de colisión" a una función de *hash* la misma debe cumplir algunos requisitos:

  Si m1 y m2 son secuencias de datos de longitud finita y H una función de *hash*, H(m1) y H(m2) son los valores resultantes de aplicar la función sobre m1 y m2, puede decirse que H es libre de colisión cuando:

  1. Si dado un número en la imagen de H, llamémoslo A, es "computacionalmente difícil" construir una secuencia m1 tal que H(m1)=A. Esta propiedad hace de H una función de un solo sentido, virtualmente no inversible (*one-way function*)

  2. Dada una secuencia m1 es "computacionalmente difícil" hallar otra secuencia m2 distinta de m1 tal que H(m1)=H(m2)

  3. Es "computacionalmente difícil" hallar secuencias m1 y m2 distintas tales que H(m1)=H(m2).

  Si la función sólo cumple 1 y 2, entonces se dice que es "libre de colisiones" en forma débil. Mientras que si cumple 1 y 3 (cumplir 3 implica cumplir 2) se dice la función es "libre de colisiones" en forma fuerte.

- **ISDN.-** Son las iniciales en inglés de Integrated Services Digital Network, es un sistema para conexiones de teléfono digital, el cual ha estado disponible por más de una década, este sistema permite la transmisión simultánea de voz y datos utilizando para ello una conectividad digital.

- **ISP (Internet Service Provider).-** Es un servidor que proporciona servicios de conexión a Internet a múltiples Usuarios.

- **LAN (Local Area Network).-** Una red "cableada" de área local

- **MAC Spoofing.-** Falsificado de una dirección MAC

- **MD5.-** Algoritmo para calcular la función de *hash* de una secuencia de bits

- **Mecanismo de autenticación.-** Es el procedimiento mediante el cual la persona muestra su(s) autenticador(es) al Servidor de Autenticación con el objeto de identificarse y tener acceso a los servicios.

- **NAS.-** Network Access Server por sus siglas en inglés, representado en las redes inalámbricas por un Punto de Acceso, es el dispositivo por medio del cual las señales radiales son convertidas a señales digitales, también llamado Cliente, el cual ayuda en el proceso de autenticación de un usuario.

- **Passphrase.-** Es una frase larga regularmente fácil de recordar para un usuario (por ejemplo la estrofa de una canción, el título de un libro, una frase famosa etc.) la cual sirve como autenticador de la misma manera en que lo hace un password.

- **Password.-** Es un secreto que únicamente el Usuario conoce o debería conocer, y que permite autenticarlo ante una autoridad que obviamente también lo conoce.

- **Puerto.-** Un Puerto en el contexto de este trabajo, es simplemente un punto de unión por medio del cual se establece la comunicación uno-a-uno entre dos dispositivos físicos.

- **RADIUS (Remote Dial In User Service).-** Es el servidor estándar para proveer servicios de Autenticación, Autorización y Auditoría a una red, aunque no es reestricitvo para las redes.

- **RC4.-** Algoritmo de cifrado simétrico, para mayor información, consultar la referencias [12] y [15].

- **Servicio.-** El NAS provee un servicio, esto es permite que el usuario realice un telnet, un PPP, etc.

- **Servidor AAA.-** Es el servidor encargado de realizar los procesos de Autenticación, Autorización y Auditoría.

- **Servidor de Autenticación.-** Es la entidad con la suficiente autoridad y conocimiento para decidir si el o los autenticador(es) presentados por una persona garantizan su identidad, su trabajo es resolver si de acuerdo a la identidad de quien dice ser dicha persona, se le otorga o niega el servicio que solicita.

- **Sesión.-** Cada servicio provisto por el NAS a un usuario constituye una sesión, siendo el inicio de una sesión definido como el punto donde el servicio es por primera vez provisto, y el final

de la sesión está definido como el punto donde el servicio es finalizado. Un usuario puede tener múltiples sesiones en paralelo o series si el NAS soporta tal funcionalidad.

- **SHA1.-** Algoritmo para calcular la función de *hash* de una secuencia de bits

- **Sniffer.-** Programa que permite monitorear los paquetes en una red, esto interceptarlos poniendo en estado promiscuo la tarjeta de red.

- **SNMP (Simple Network Management Protocol).-** Protocolo para la administración de una red.

- **Usuario.-** Es la persona que pretende hacer uso de un cierto servicio, el cual debe autenticarse antes de poder hacerlo.

- **WEP (Wired Equivalent Privacy).-** protocolo de cifrado en redes inalámbricas para realizar la autenticación y garantizar la confidencialidad.

- **Wi-Fi.-** La alianza *Wi-Fi*. es una asociación global de industrias sin fines de lucro, formada por más de 200 compañías miembro dedicadas a promover el crecimiento de las redes inalámbricas de área local, con el propósito de aumentar la experiencia de los usuarios en los dispositivos móviles inalámbricos, la alianza *Wi-Fi* se encarga de probar y certificar la interoperabilidad de los productos de redes inalámbricas, basados en la especificación 802.11. Desde la introducción del programa de certificación *Wi-Fi* en marzo del 2000, más de 2000 productos han sido certificados, fomentando el uso de los productos y servicios Wi-Fi.

- **WLAN (Wireless Local Area Network).-** Una red inalámbrica de área local

- **WPA (Wi-Fi Protected Access).-** Es un protocolo de autenticación de usuarios desarrollado como reemplazo del anterior WEP debido a las vulnerabilidades de éste último.

ANEXOS

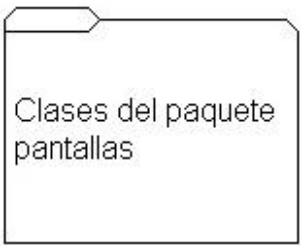
ANEXO.- En este anexo se incluyen algunos códigos fuente del proyecto.

Clase Inicio.- Esta clase permite iniciar la ejecución del servidor de autenticación.

```
/*********************************************************************
 * ARCHIVO : Inicio.java
 * DESCRIPCION : Clase que comienza la ejecución del proyecto
 * Created on 25 de abril de 2005, 10:15 AM
 *********************************************************************/
import clases.pantallas.*;
/*********************************************************************
 * @notas :
 * @author Juan Francisco García Navarro
 *********************************************************************/
public class Inicio
{

    /**************************************************
     * @param args the command line arguments
     * @notas:
     **************************************************/
    public static void main(String[] args)
    {
        try{
            PantallaPrincipal pantallaPrincipal = new PantallaPrincipal();
            /*clases.AuthenticationServer authServer = new clases.AuthenticationServer();
            clases.ServidorRadius radiusServer = new clases.ServidorRadius();
            clases.ActualizaSesion update = new clases.ActualizaSesion();
            radiusServer.start();
            authServer.start();
            update.start();*/
        }catch(Exception ex)
        {
            ex.printStackTrace();
            System.exit(1);
        }
    }/// fin del metodo main

}/// fin de la clase Inicio
```



Este paquete contiene las clases para la manipulación e interacción gráfica con el usuario. Por simplicidad se omite el código fuente de estas clases, numeradas en la sección 4.5.

Clase PantallaPrincipal

Clase PantallaLogin

Clase PantallaListaUsuarios

Clase PantallaListaTarjetas

Clase PantallaListaClientes

Clase PantallaBorraUsuario

Clase PantallaBorraMac

Clase PantallaBorraCliente

Clase PantallaAsociaciones

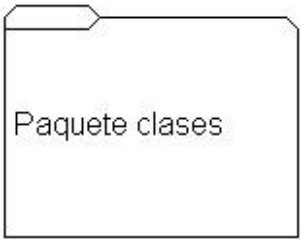
Clase Mensaje

Clase IngresaUsuario

Clase IngresaMac

Clase IngresaCliente

Clase Confirmacion



Este paquete contiene clases de propósito general.

Clase ActualizaSesion

```
/*************************************************************
 * ARC HIVO : ActualizaSesion.java
 *
 * Created on 3 de mayo de 2005, 05:13 PM
 *************************************************************/

package clases;

/*************************************************************
 * DESCRIPCION : Clase que permite periodicamente revisar si
 * hay alguna sesiòn de usuario que ya no està activa, para
 * sacar al objeto de la lista de sesiones.
 * @author Juan Francisco GarcÃ¬a Navarro
 *************************************************************/
public class ActualizaS esion extends Thread
{
    /** constante que define el tiempo que este hilo se "duerme" **/
    private static final int TIEMPO_SESION =  (1000 * 60) * 3;/// 3 minutos
    /** objeto Enumeration para recuperar todas las llaves de la tabla **/
    private java.util.Enumeration llaves = null;
    /** objeto String para almacenar el login temporal **/
```

```
    private String login = null;
    /** variable que permite almacenar temporalmente el valor del tiempo **/
    private long tiempo = 0;
    /** objeto que permite almacenar temporalmente la Direcciòn mac **/
    private String mac = null;
    /** objeto para almacenar el tiempo actual **/
    private long tiempo_actual = 0;

    /*************************************************************
     * CONSTRUCTOR :
     *************************************************************/
    public ActualizaSesion()
    {
    }/// fin del constructor()
    /************************************************
     * inicia la ejecuciòn del hilo
     ************************************************/
    public void run()
    {
        try{
            while(true)
            {
                Thread.sleep(TIEMPO_SESION);/// esperar un tiempo

                /// desplegar aviso:
                System.out.println("Actualizando la sesión: "+((java.util.Calendar.getInstance()).get(java.util.Calendar.MINUTE))
                    +" : "+((java.util.Calendar.getInstance()).get(java.util.Calendar.SECOND)));

                /// para cada elemento en la lista....
                for (this.llaves = ListaAccesos.getLista().getLlaves(); llaves.hasMoreElements(); )
                {
                    this.mac = (String)llaves.nextElement();/// obtener la llave
                    /// obtener el login de este usuario
                    this.login = ((ConsultaUsuario) ListaAccesos.getLista().getConsultaUsuario(this.mac)).getLogin();
                    this.tiempo = ListaAccesos.getLista().getTiempo(this.login);/// tiempo del ultimo mensaje

                    System.out.println("revisando para: "+login+"  "+mac+" tiempo: "+tiempo);
                    /// revisar si ese tiempo excede al intervalo de tiempo permitido
                    /// caso asirmativo eliminar el registro de la sesión
                   tiempo_actual = System.currentTimeMillis();
                    System.out.println("tiempo actual "+tiempo_actual+" diferencia "+ (tiempo_actual - tiempo));
                    if (tiempo_actual - this.tiempo > TIEMPO_SESION)
                        ListaAccesos.getLista().terminaSesion(this.mac);
                }/// fin del ciclo que recorre todos los elementos de la lista
            }/// fin del cilco que otera mientras este hilo se encuentre activo
        }catch(InterruptedException ex)
        {
            this.interrupt();
        }catch(Exception ex)
        {
            ex.printStackTrace();
            System.out.println(ex.getMessage());
            this.interrupt();
        }
    }/// fin del metodo run()

}/// fin de la clase ActualizaSesion
```

Clase AuthenticationServer

```
/*****************************************************************
 * CLASE : AuthenticationServer.java
 *
 * Created on 28 de abril de 2005, 04:00 PM
 *****************************************************************/
    package clases;
    import java.io.*;
    import java.net.*;

/*********************************************************************
 * DESCRIPCION : Clase que implementa un servidor de autenticaciòn
 * multihilo, cada petición genera un nuevo hilo para atenderla.
 * @author  Juan Francisco GarcÃ¬a Navarro
 *********************************************************************/
public class AuthenticationServer extends Thread
{
    ////////////////////////////////////////////////////
    /////       DECLARACION DE ATRIBUTOS            //////
    ////////////////////////////////////////////////////
    /** constante del puerto para leer los datagramas del access point **/
    private static final int PUERTO_1815 = 1815;
    /** escuchar en el puerto 1815 los paquetes que vienen del access point **/
    private DatagramSocket socket1815 = null;
    /** arreglo para guardar los bytes recibidos **/
    private byte[] datosRecibidos = new byte[128];
    /** datagrama que se recibe **/
    private DatagramPacket paquete_recibido = null;

    ////////////////////////////////////////////////////
    /////       DECLARACION DE ATRIBUTOS            //////
    ////////////////////////////////////////////////////

    /*****************************************************************
     * CONSTRUCTOR : Instancia los objetos socket y DatagramPacket
     *****************************************************************/
    public AuthenticationServer()
    {
        try{
            this.socket1815 = new DatagramSocket(this.PUERTO_1815);
            System.out.println("Servidor de autenticaciònescuchando en el puerto 1815");
            this.paquete_recibido = new DatagramPacket(this.datosRecibidos,this.datosRecibidos.length);
        }catch(java.io.IOException ex){
             ex.printStackTrace();
        }
        catch(Exception ex){
            ex.printStackTrace();
        }
    }/// fin del constructor()

    /*****************************************************************
     * metodo que comienza la ejecuciòn del hilo
     *****************************************************************/
    public void run()
    {
        while(true)/// ciclo que itera indefinidamente
        {
            try{
```

```
            /// esperar a recibir algÃ¹n datagrama
            this.socket1815.receive(this.paquete_recibido);

          /// generar un hilo que atienda la petición
          (new clases.PeticionProtocolo(this.paquete_recibido.getAddress(),
            this.paquete_recibido.getPort(),this.datosRecibidos,this.socket1815)).run();

          }catch(java.io.IOException ex)
          {
            System.out.println(ex.getMessage());
          }catch(Exception ex)
          {
            System.out.println(ex.getMessage());
          }
        }/// fin del ciclo que itera indefinidamente
    }/// fin del metodo run()

    /**************************************************************
     * CUERPO PRINCIPAL para ejecutar pruebas
     **************************************************************/
    public static void main(String args[])
    {
        AuthenticationServer authServer = new AuthenticationServer();
        authServer.run();
    }
}/// fin de la clase AuthenticationServer
```

Clase ConsultaUsuario

Clase Datagramas

Clase GeneraNumeroAleatorio

Clase ListaAccesos

Clase PeticionProtocolo

```
/*********************************************************
 * ARCHIVO : PeticionProtocolo.java
 *
 * Created on 28 de abril de 2005, 04:06 PM
 *********************************************************/

    package clases;
    import java.io.*;
    import java.net.*;
    import clases.jdbc.ConsultasBD;
    import clases.beans.Usuario;
    import cripto.*;

/*********************************************************
 * DESCRIPCION : Clase que atiende las peticiones del
 * servidor de autenticaciòn.
 * @author  Juan Francisco GarcÃ¬a Navarro
 *********************************************************/
public class PeticionProtocolo extends Thread
{
```

```
//////////////////////////////////////////
///// DECLARACION DE ATRIBUTOS /////
//////////////////////////////////////////
/** objeto que guarda la Direcciòn IP del usuario para respuesta **/
private InetAddress IPAddress = null;
/** variable que permite almacenar el puerto del usuario **/
private int puerto = 0;
/** arreglo que permite almacenar localmente los bytes recibidos **/
private byte[] datosRecibidos = null;
/** arreglo para guardar los bytes para enviar **/
private byte[] datosEnviar = null;
/** datagrama que se envÃ¬a **/
private DatagramPacket paquete_enviar = null;
/** objeto para realizar las consultas a la base de datos **/
private ConsultasBD consultas = null;
/** escuchar en el puerto 1815 los paquetes que vienen del access point **/
private DatagramSocket socket1815 = null;
/** objeto que permite guardar el tipo del paquete recibido,
 para hacer algunas validaciones **/
private byte tipoPaquete = 0;
/** valor para guardar el texto cifrado **/
private String texto_cifrado = "";
/** valor para almacenar el texto ya descifrado **/
private String texto_descifrado = "";
/** objeto para cifrar usando el algoritmo DES **/
private AlgoritmoDES cifradorDES = null;
/** objeto que genera un nÃ¹mero aleatorio **/
private GeneraNumeroAleatorio generador = null;
/** autenticador que se desea cifrar **/
private String autenticador = null;
/** arreglo de bytes para hacer una copia local de los bytes del datagrama
 con el tamaño justo del datagrama **/
private byte[] datagramaRecibido = null;

/*****************************************************************
 * CONSTRUCTOR : Inicializa valores
 * @param InetAddress IPAddress .- la Direcciòn IP del usuario
 * @param int puerto .- puerto de comunicaciones
 * @param byte[] datosRecibidos .- es el datagrama recibido
 * @param DatagramSocket socket .- el objeto para contestar
 *****************************************************************/
public PeticionProtocolo(InetAddress IPAddress,int puerto,byte[] datosRecibidos,DatagramSocket socket)
{
   this.IPAddress = IPAddress;
   this.puerto = puerto;
   this.datosRecibidos = datosRecibidos;
   this.socket1815 = socket;
   /// instanciar el objeto que realiza las consultas a la base de datos
   this.consultas = new ConsultasBD();

}/// fin del constructor()

/*******************************************************
 * metodo que inicia la ejecuciòn del hilo
 *******************************************************/
public void run()
{
        /// variables temporales
   String texto_cifrado = "";
```

```
String login_temporal = "";
String llave = null;
String direccion_mac = null;
byte[] login = null;

try{
        /// copiar a otro arreglo de bytes, el datagrama recibido con el tamaño original
        copiaDatagrama();/// lanzar una excepciÃ²n

        /// desplegar el paquete recibido
        System.out.println("\n\n Datagrama recibido \n");
        Datagramas.despliegaPaquete(this.datagramaRecibido);

        /// obtener el valor del login dentro del paquete recibido
        login = Datagramas.getAtributo(datagramaRecibido,Datagramas.ATRIBUTO_LOGIN);
        if (login == null)
            throw new Exception("¡ERROR!, el paquete no contiene atributo login");

        login_temporal = new String(login);

        /// hacer una consulta a la base de datos para traer la llave del usuario
        llave = consultas.buscaLlave(login_temporal);

        /// validar que la llave exista, sino existe regresa un valor null
        if (llave == null)
            throw new Exception("¡ERROR!, esa llave para el cliente no existe");
        else
            System.out.println("llave: "+llave+"  longitÃ¹d: "+llave.length());

        /// validar el datagrama bien formado, y que el hash este correcto
        if (!Datagramas.validaDatagrama(datagramaRecibido,llave))
            throw new Exception("¡ERROR!, el datagrama es invÃ lido");

       /// el tipo de paquete sesión no contiene autenticador
        /// pero los restantes si, por lo que hay que validarlo
        this.tipoPaquete = this.datagramaRecibido[0];

        if (this.tipoPaquete != Datagramas.PAQUETE_SESION)
        {

            /// descifrar el autenticador del paquete recibido
           /// instanciar el objeto cifrador con la llave de cifrado
            this.cifradorDES = new AlgoritmoDES(llave.getBytes());

            /// descifrar el autenticador
            texto_cifrado = Datagramas.getAutenticador(this.datagramaRecibido);

            /// validar que venga el autenticador
            if (texto_cifrado == null)
                throw new Exception("¡ERROR!, autenticador incorrecto");

            /// descifrar el autenticador
            this.texto_descifrado = this.cifradorDES.descifraDES(texto_cifrado);

            /// validar el texto descifrado
            if (texto_descifrado == null)
                throw new Exception("¡ERROR!, autenticaciònfallida");

            /// hacer una consulta para buscar la direccion mac
```

```
        direccion_mac = consultas.buscaMacXLogin(login_temporal);

        /// validar que exista la Direcciòn mac
        if (direccion_mac == null)
            throw new Exception("¡ERROR!, no existe la Direcciòn mac");
    }/// fin del condicional que revisa el tipo de paquete para validaciòn

/// condicional que revisar el codigo del paquete para la respuesta
switch (this.datosRecibidos[0])
{
////----------------------------------------------------------\\\\\\
////   caso cuando el paquete es PETICION DE AUTENTICACION     \\\\\\
////----------------------------------------------------------\\\\\\
case Datagramas.PETICION_AUTENTICACION:

        /// verificar el estado de la transaccion, debe estar en MAC_OK
        System.out.print("   el estado de la transaccion es: "+
            (ListaAccesos.getLista().getEstado(direccion_mac)));

        if ((ListaAccesos.getLista().getEstado(direccion_mac)) != ConsultaUsuario.MAC_OK )
            throw new Exception("¡ERROR!, no ha validado la tarjeta MAC");

        System.out.println("  autenticador descifrado  "+this.texto_descifrado+"\n");

        /// ahora generar el autenticador del servidor y concatenarlo al autenticador
        /// del usuario, el separador de los dos autenticadores es el pipe '|'
        /// generar un numero aleatorio
        this.generador = new GeneraNumeroAleatorio();

        /// cifrar el numero con la llave de cifrado del cliente
        this.autenticador = this.generador.getNumeroEnString();

        /// este autenticador hay que guardarlo en la ListaAccesos
        ListaAccesos.getLista().insertaAutenticador(login_temporal, autenticador);

        /// cifrar los dos autenticadores...
        this.texto_cifrado = this.cifradorDES.cifraDES(this.texto_descifrado+"|"+this.autenticador);
        System.out.println("El autenticador generado es: "+this.autenticador+
            "  y el autenticador cifrado es: "+this.texto_cifrado);

        /// construir el datagrama PAQUETE DE RESPUESTA DEL SERVIDOR AAA
        this.datosEnviar = Datagramas.generaPaquete(Datagramas.RESPUESTA_AAA,
              this.texto_cifrado,null,llave);

        /// enviar el paquete
        enviaDatagrama();

        /// en este momento hay que cambiar el estado a LOGIN_OK
        ListaAccesos.getLista().setEstado(direccion_mac, ConsultaUsuario.LOGIN_OK);

 break;/// fin de la opciòn paquete tipo PETICION_AUTENTICACION
 ////----------------------------------------------------------\\\\\\
//// caso en el que el paquete es del tipo RESPUESTA_USUARIO   \\\\\\
 ////----------------------------------------------------------\\\\\\
 case Datagramas.RESPUESTA_USUARIO:

    /// leer el autenticador guardado en la lista de accesos
    this.autenticador = ListaAccesos.getLista().getAutenticador(login_temporal);
```

```
            /// validar el autenticador
            if (this.autenticador == null)
                throw new Exception("¡ERROR!, no existe el autenticador");

              ///  validar el autenticador recibido
              if (validacion(this.texto_descifrado))
              {
                  /// cuando la la autenticaciònes positiva, hay que poner la transaccion en el estado OK
                  ListaAccesos.getLista().setEstado(direccion_mac, ConsultaUsuario.OK);
                 System.out.println("AUTENTICACION POSITIVA con la mac: "+direccion_mac);
              }/// fin del condicional de autenticacion positiva
              else /// cuando la autenticacion es negativa, poner el estado RECHAZO
              {
                  ListaAccesos.getLista().setEstado(direccion_mac, ConsultaUsuario.RECHAZO);
                  System.out.println("AUTENTICACION NEGATIVA");
              }/// fin del condicional de la autenticacion negativa

              /// verificar el estado de la transaccion
              System.out.print("   el estado de la transaccion es: "+(ListaAccesos.getLista().getEstado(direccion_mac)));

              /// quitar de la tabla de autenticadores, este elemento
              ListaAccesos.getLista().eliminaAutenticador(login_temporal);

         break;/// fin de la opcion paquete de respuesta del usuario
        ////----------------------------------------------------------\\\\\\
        ////   caso cuando el tipo de paquete es PAQUETE DE SESION     \\\\\\
        ////----------------------------------------------------------\\\\\\
        case Datagramas.PAQUETE_SESION:
        /// guardar en la lista de sesiones este tiempo
            System.out.println("Actualizando el tiempo para: "+login_temporal);
            ListaAccesos.getLista().actualizaTiempo(login_temporal, System.currentTimeMillis());
            break;
         ////----------------------------------------------------------\\\\\\
         //// caso en el que el paquete es de un tipo desconocido       \\\\\\
        ////----------------------------------------------------------\\\\\\
         default:   /// obviamente es un caso de error, lanzar una excepciÃ²n
             throw new Exception("¡ERROR!, paquete invÃlido");
         }/// fin del condicional que revisa el tipo del paquete
    }catch(Exception ex)
    {
        ex.printStackTrace();
        System.out.println(ex.getMessage());
    }finally
    {   /// una vez regresado o no el mensaje, destruir este hilo
        this.interrupt();
        this.consultas.desconectaBD();
    }
}/// fin del metodo run()

/*****************************************************************
 * método que copia el datagrama recibido almacenado en el
 * arreglo datosRecibidos, al arreglo datagramaRecibido
 *****************************************************************/
private void copiaDatagrama() throws Exception
{
    switch (this.datosRecibidos[0])
    {
        /// para este caso el tamaño depende de la cantidad de atributos
        /// y el tamaño de sus valores, unicamente considera un solo
```

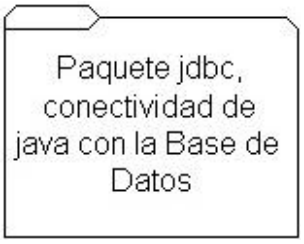
```
            /// atributo, en el caso de haber mas modificar esta secciÃ²n
            case Datagramas.RESPUESTA_USUARIO:
            case Datagramas.PETICION_AUTENTICACION:
                this.datagramaRecibido = new byte[41+this.datosRecibidos[42]];
                break;
            case Datagramas.RESPUESTA_AAA:
                this.datagramaRecibido = new byte[49];
                break;
            case Datagramas.PAQUETE_RECHAZO:
                break;
            case Datagramas.PAQUETE_SESION:
                this.datagramaRecibido = new byte[17+this.datosRecibidos[18]];
                break;
            default:
                throw new Exception("¡ERROR!, el datagrama es invÃlido");
        }/// fin del condicional que revisa el codigo del datagrama
        for (int i=0;i<this.datagramaRecibido.length;i++)
            this.datagramaRecibido[i] = this.datosRecibidos[i];

    }/// fin del método copiaDatagrama()
    /**********************************************************************
     * metodo que envia un datagrama ya creado previamente
     **********************************************************************/
    private void enviaDatagrama()
    {
        try{
            /// crear el nuevo datagrama que se va a enviar
            this.paquete_enviar = new DatagramPacket(this.datosEnviar,this.datosEnviar.length,
                this.IPAddress,this.puerto);
            /// enviar el datagrama
            this.socket1815.send(this.paquete_enviar);
            System.out.println();
            System.out.print("Datagrama enviado al Access Point en el instante: "
                +((java.util.Calendar.getInstance()).get(java.util.Calendar.MINUTE))
                +" : "+((java.util.Calendar.getInstance()).get(java.util.Calendar.SECOND)));
            Datagramas.despliegaPaquete(this.datosEnviar);
        }catch(java.io.IOException ex)
        {
            System.out.println(ex.getMessage());
        }catch(Exception ex)
        {
            System.out.println(ex.getMessage());
        }
    }/// fin del metodo enviaDatagrama()
    /**********************************************************************
     * método que valida que el nÃ¹nico generado por el servidor sea
     * igual al valor regresado por el cliente.
     * El valor del autenticador esta guardado en la ListaAccesos
     **********************************************************************/
    private boolean validacion(String valor)
    {
        if (valor.equals(this.autenticador))
            return true;
        else
            return false;
    }

}//// fin de la clase PeticionProtocolo
```

Clase PeticionRadius

Clase ServidorRadius



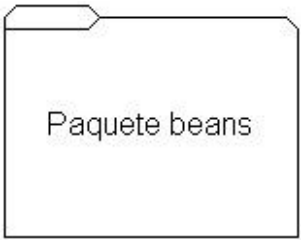
Este paquete contiene clases que permiten la conectividad con la base de datos.

Clase ConsultasBD

Clase EliminacionesBD

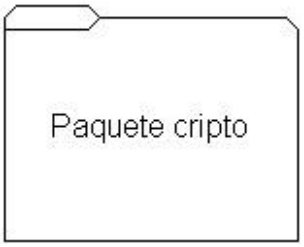
Clase InsercionesBD

Clase TablasBD



Este paquete contiene clases del tipo java beans.

Clase Usuario



Este paquete contiene clases que implementan los algoritmos de cifrado criptogràfico.

Clase AlgoritmoDES

Clase MD5

Clase Sha01

```
package cripto;
import java.security.*;

public final class Sha01
```

```
{

  /**************************************************
   * This method generates and returns a digest
   * for an incoming array of bytes using Sun's
   * SHA message digest algorithm..
   **************************************************/
  public static byte[] digestIt(byte[] dataIn)
  {
    byte[] theDigest = null;
    try{
      //Create a MessageDigest object
      // implementing the SHA algorithm, as
      // supplied by SUN
      MessageDigest messageDigest = MessageDigest.getInstance("SHA", "SUN");
      //Feed the byte array to the digester.  Can
      // accommodate multiple calls if needed
      messageDigest.update(dataIn);
      //Complete the digestion and save the
      // result
      theDigest = messageDigest.digest();
    }catch(Exception e){System.out.println(e);}

    //Return the digest value to the calling
    // method as an array of bytes.
    return theDigest;
  }//end digestIt()


  /**************************************************************
   *This method converts an incoming array of
   * bytes into a string that represents each of
   * the bytes as two hex characters.
   **************************************************************/
  public static String byteArrayToHexStr(byte[] data)
  {
    String output = "";
    String tempStr = "";
    int tempInt  = 0;
    for(int cnt = 0;cnt < data.length;cnt++){
      //Deposit a byte into the 8 lsb of an int.
      tempInt = data[cnt]&0xFF;
      //Get hex representation of the int as a
      // string.
      tempStr = Integer.toHexString(tempInt);
      //Append a leading 0 if necessary so that
      // each hex string will contain two
      // characters.
      if(tempStr.length() == 1)
                      tempStr = "0" + tempStr;
      //Concatenate the two characters to the
      // output string.
      output = output + tempStr;
    }//end for loop
    return output.toUpperCase();
  }//end byteArrayToHexStr

  /**************************************************
   * mÃ¨todo que cifra con SHA-1
```

```
   ************************************************/
  public static byte[] cifrar(String texto) throws Exception
  {
      java.security.MessageDigest digest = java.security.MessageDigest.getInstance("SHA-1","SUN");
      digest.reset();
      digest.update(texto.getBytes());
      return digest.digest();
  }/// fi n del mÃ¨todo cifrar()

}//end class Sha01
```

Clase AES

```
/*****************************************************************
 * ARCHIVO : AES.java
 * DESCRIPCION : Clase
 * Created on 2 de agosto de 2005, 02:20 PM
 *****************************************************************/
    package cripto;
    import java.security.*;
    import javax.crypto.*;
    import javax.crypto.spec.*;
    import java.io.*;

 /*****************************************************************************
  * This program generates a AES key, retrieves its raw bytes, and
  * then reinstantiates a AES key from the key bytes.
  * The reinstantiated key is used to initialize a AES cipher for
  * encryption and decryption.
  * @author Juan Francisco GarcÃ¬a Navarro
  *****************************************************************************/
public class AES
{
     /*****************************************************************
     * convierte un arreglo debytes a una cadena de caracteres
     * @param buf       Array of bytes to convert to hex string
     * @return          Generated hex string
     *****************************************************************/
     public static String asHex (byte buf[])
     {
      StringBuffer strbuf = new StringBuffer(buf.length * 2);
      int i;

      for (i = 0; i < buf.length; i++)
      {
       if (((int) buf[i] & 0xff) < 0x10)
                 strbuf.append("0");

        strbuf.append(Long.toString((int) buf[i] & 0xff, 16));
      }

      return strbuf.toString();
     }/// fin del metodo asHex()

  /************************************************************
   *This method converts an incoming array of
   * bytes into a string that represents each of
   * the bytes as two hex characters.
   ************************************************************/
```

```
public static String byteArrayToHexStr(byte[] data)
{
  String output = "";
  String tempStr = "";
  int tempInt = 0;
  for(int cnt = 0;cnt < data.length;cnt++){
    //Deposit a byte into the 8 lsb of an int.
    tempInt = data[cnt]&0xFF;
    //Get hex representation of the int as a
    // string.
    tempStr = Integer.toHexString(tempInt);
    //Append a leading 0 if necessary so that
    // each hex string will contain two
    // characters.
    if(tempStr.length() == 1)
                    tempStr = "0" + tempStr;
    //Concatenate the two characters to the
    // output string.
    output = output + tempStr;
  }//end for loop
  return output.toUpperCase();
}//end byteArrayToHexStr

/*****************************************************************
 * mÃ¨todo que cifra usando AES como algoritmo
 * @param String texto.- es el mensaje a cifrar
 * @returns el arreglo de bytes cifrado
 *****************************************************************/
public static byte[] cifrarAES(String texto,byte[] llave)
{
    byte[] encrypted = null;
    try{
        SecretKeySpec skeySpec = new SecretKeySpec(llave, "AES");

        // obtener la referencia del objeto cifrador
        Cipher cipher = Cipher.getInstance("AES");

        /// inicializar el cifrador
        cipher.init(Cipher.ENCRYPT_MODE, skeySpec);

        /// cifrar el arreglo de bytes
        encrypted = cipher.doFinal(texto.getBytes());
    }catch(java.security.NoSuchAlgorithmException ex){
        ex.printStackTrace();
    }catch(java.security.InvalidKeyException ex){
        ex.printStackTrace();
    }catch(javax.crypto.IllegalBlockSizeException ex){
        ex.printStackTrace();
    }catch(javax.crypto.NoSuchPaddingException ex){
        ex.printStackTrace();
    }catch(javax.crypto.BadPaddingException ex){
        ex.printStackTrace();
    }
     return encrypted;
}/// fin del metodo cifrarAES()
/*****************************************************************
 * mÃ¨todo que cifra usando AES como algoritmo
 * @param String texto.- es el mensaje a cifrar
 * @returns el arreglo de bytes cifrado
```

```
 ****************************************************************/
  public static byte[] descifrarAES(byte[] texto,byte[] llave)
  {
      byte[] encrypted = null;
      try{
           SecretKeySpec skeySpec = new SecretKeySpec(llave, "AES");

           // obtener la referencia del objeto cifrador
           Cipher cipher = Cipher.getInstance("AES");

           /// inicializar el cifrador
           cipher.init(Cipher.DECRYPT_MODE, skeySpec);

           /// cifrar el arreglo de bytes
           encrypted = cipher.doFinal(texto);
      }catch(java.security.NoSuchAlgorithmException ex){
          ex.printStackTrace();
      }catch(java.security.InvalidKeyException ex){
          ex.printStackTrace();
      }catch(javax.crypto.IllegalBlockSizeException ex){
          ex.printStackTrace();
      }catch(javax.crypto.NoSuchPaddingException ex){
          ex.printStackTrace();
      }catch(javax.crypto.BadPaddingException ex){
          ex.printStackTrace();
      }
       return encrypted;
  }/// fin del metodo descifrarAES()

}/// fin de la clase AES
```

BIBLIOGRAFÍA Y REFERENCIAS ELECTRÓNICAS

[1] RFC 2865 - Remote Authentication Dial In User Service (RADIUS).- http://www.faqs.org/rfcs/rfc2865.html

[2] Jonathan Hassell, "RADIUS", Ed. O'Reilly, Octubre 2002

[3] Andrew S. Tanenbaum, "Redes de Computadoras", cuarta edición, Ed. Prentice may, 2003

[4] sitio oficial del consorcio IEEE, www.ieee.org

[5] http://grouper.ieee.org/groups/802/11/

[6] sitio oficial del consorcio IETF, www.ietf.org

[7] RFC 1321 – The MD5 Message-Digest Algorithm.- www.faqs.org/rfcs/rfc1321.html

[8] RFC 1829.- The ESP DES-CBC Transform.- www.faqs.org/rfcs/rfc1829.html

[9] sitio oficial de java, http://java.sun.com

[10] sitio del NIST, www.nist.gov

[11] 802.11 Security, Bob Fleck, Bruce Potter, O'Reilly, december 2002

[12] Laboratorios RSA, www.rsasecurity.com

[13] Bruce Schneier, Applied Cryptography, Second Edition, John Wiley & Sons, Inc.

[14] Intercepting Mobile Communications: The Insecurity of 802.11, Nikita Borisov, Ian Goldberg, David Wagner, www.isaac.cs.berkeley.edu/isaac/mobicom.pdf

[15] R.L Rivest. The RC4 Encryption Algorithm. RSA Data Security, Inc., Mar. 12, 1992 (Propietary)

[16] L.M.S.C. of the IEEE Computer Society. Wirelless LAN medium access control (MAC) and physical layer (PHY) specifications. IEEE Standard 802.11, 1999 Edition, 1999.

[17] Artículo denominado ***"WiFi Attack Vectors"*** publicado en la revista ***"COMMUNICATIONS of the ACM "*** de fecha agosto del 2005, volumen 48, número 8

[18] IEEE Standard for Local and Metropolitan Area Network, Port-Based Network Access Control, IEEE Std 802.1X-2001, http://standards.ieee.org/getieee802/download/802.1X-2001.pdf

[19] RFC 2284, PPP Extensible Authentication Protocol (EAP).- http://www.faqs.org/rfcs/rfc2284.html

[20] RFC 3174, US Secure Hash Algorithm 1 (SHA1).- http://www.faqs.org/rfcs/rfc3174.html

[21] FIPS 197, Specification for the ADVANCED ENCRYPTION STANDARD (AES), November 26, 2001, http://csrc.nist.gov/publications/fips/fips 197/fips-197.pdf

[22] Artículo "Securing Wi-Fi Networks", publicado en la revista Computer, de fecha Julio del 2005, autores Kjell J., Erlend Dyrnes, Per Thorsheim

ANEXOS

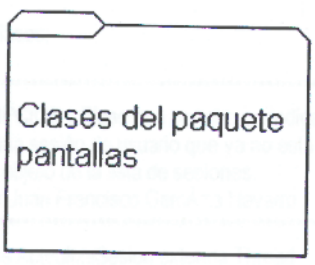
ANEXO.- En este anexo se incluyen algunos códigos fuente del proyecto.

Clase Inicio.- Esta clase permite iniciar la ejecución del servidor de autenticación.

```
/**********************************************************************
 * ARCHIVO : Inicio.java
 * DESCRIPCION : Clase que comienza la ejecución del proyecto
 * Created on 25 de abril de 2005, 10:15 AM
 **********************************************************************/
import clases.pantallas.*;
/**********************************************************************
 * @notas :
 * @author Juan Francisco García Navarro
 **********************************************************************/
public class Inicio
{

  /*************************************************
   * @param args the command line arguments
   * @notas:
   *************************************************/
  public static void main(String[] args)
  {
    try{
      PantallaPrincipal pantallaPrincipal = new PantallaPrincipal();
      /*clases.AuthenticationServer authServer = new clases.AuthenticationServer();
      clases.ServidorRadius radiusServer = new clases.ServidorRadius();
      clases.ActualizaSesion update = new clases.ActualizaSesion();
      radiusServer.start();
      authServer.start();
      update.start();*/
    }catch(Exception ex)
    {
      ex.printStackTrace();
      System.exit(1);
    }
  }/// fin del metodo main

}/// fin de la clase Inicio
```



Este paquete contiene las clases para la manipulación e interacción gráfica con el usuario. Por simplicidad se omite el código fuente de estas clases, numeradas en la sección 4.5.

Clase PantallaPrincipal

Clase PantallaLogin

Clase PantallaListaUsuarios

Clase PantallaListaTarjetas

Clase PantallaListaClientes

Clase PantallaBorraUsuario

Clase PantallaBorraMac

Clase PantallaBorraCliente

Clase PantallaAsociaciones

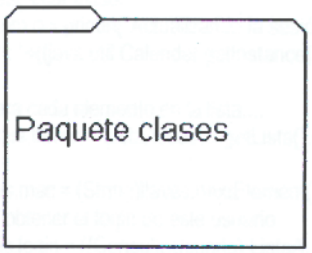
Clase Mensaje

Clase IngresaUsuario

Clase IngresaMac

Clase IngresaCliente

Clase Confirmacion



Este paquete contiene clases de propósito general.

Clase ActualizaSesion

```
/*************************************************************
 * ARCHIVO : ActualizaSesion.java
 *
 * Created on 3 de mayo de 2005, 05:13 PM
 *************************************************************/

package clases;

/*************************************************************
 * DESCRIPCION : Clase que permite periodicamente revisar si
 * hay alguna sesiòn de usuario que ya no està activa, para
 * sacar al objeto de la lista de sesiones.
 * @author Juan Francisco GarcÃ¬a Navarro
 *************************************************************/
public class ActualizaSesion extends Thread
{
   /** constante que define el tiempo que este hilo se "duerme" **/
   private static final int TIEMPO_SESION =  (1000 * 60) * 3;/// 3 minutos
   /** objeto Enumeration para recuperar todas las llaves de la tabla **/
   private java.util.Enumeration llaves = null;
   /** objeto String para almacenar el login temporal **/
```

```
    private String login = null;
    /** variable que permite almacenar temporalmente el valor del tiempo **/
    private long tiempo = 0;
    /** objeto que permite almacenar temporalmente la Direcciòn mac **/
    private String mac = null;
    /** objeto para almacenar el tiempo actual **/
    private long tiempo_actual = 0;

    /*********************************************************
     * CONSTRUCTOR :
     *********************************************************/
    public ActualizaSesion()
    {
    }/// fin del constructor()
    /*************************************************
     * inicia la ejecuciòn del hilo
     *************************************************/
    public void run()
    {
      try{
        while(true)
        {
          Thread.sleep(TIEMPO_SESION);/// esperar un tiempo

          /// desplegar aviso:
          System.out.println("Actualizando la sesión: "+((java.util.Calendar.getInstance()).get(java.util.Calendar.MINUTE))
            +" : "+((java.util.Calendar.getInstance()).get(java.util.Calendar.SECOND)));

          /// para cada elemento en la lista....
          for (this.llaves = ListaAccesos.getLista().getLlaves(); llaves.hasMoreElements(); )
          {
            this.mac = (String)llaves.nextElement();/// obtener la llave
            /// obtener el login de este usuario
            this.login = ((ConsultaUsuario) ListaAccesos.getLista().getConsultaUsuario(this.mac)).getLogin();
            this.tiempo = ListaAccesos.getLista().getTiempo(this.login);/// tiempo del ultimo mensaje

            System.out.println("revisando para: "+login+"  "+mac+" tiempo: "+tiempo);
            /// revisar si ese tiempo excede al intervalo de tiempo permitido
            /// caso asirmativo eliminar el registro de la sesión
            tiempo_actual = System.currentTimeMillis();
            System.out.println("tiempo actual "+tiempo_actual+" diferencia "+ (tiempo_actual - tiempo));
            if (tiempo_actual - this.tiempo > TIEMPO_SESION)
              ListaAccesos.getLista().terminaSesion(this.mac);
          }/// fin del ciclo que recorre todos los elementos de la lista
        }/// fin del cilco que otera mientras este hilo se encuentre activo
      }catch(InterruptedException ex)
      {
        this.interrupt();
      }catch(Exception ex)
      {
        ex.printStackTrace();
        System.out.println(ex.getMessage());
        this.interrupt();
      }
    }/// fin del metodo run()

}/// fin de la clase ActualizaSesion
```

Clase AuthenticationServer

```
/*************************************************************
 * CLASE : AuthenticationServer.java
 *
 * Created on 28 de abril de 2005, 04:00 PM
 *************************************************************/
  package clases;
  import java.io.*;
  import java.net.*;

/*****************************************************************
 * DESCRIPCION : Clase que implementa un servidor de autenticaciòn
 * multihilo, cada petición genera un nuevo hilo para atenderla.
 * @author  Juan Francisco GarcÃ¬a Navarro
 *****************************************************************/
public class AuthenticationServer extends Thread
{
  ////////////////////////////////////////////
  /////     DECLARACION DE ATRIBUTOS        //////
  ////////////////////////////////////////////
  /** constante del puerto para leer los datagramas del access point **/
  private static final int PUERTO_1815 = 1815;
  /** escuchar en el puerto 1815 los paquetes que vienen del access point **/
  private DatagramSocket socket1815 = null;
  /** arreglo para guardar los bytes recibidos **/
  private byte[] datosRecibidos = new byte[128];
  /** datagrama que se recibe **/
  private DatagramPacket paquete_recibido = null;

  ////////////////////////////////////////////
  /////     DECLARACION DE ATRIBUTOS        //////
  ////////////////////////////////////////////

  /*************************************************************
   * CONSTRUCTOR : Instancia los objetos socket y DatagramPacket
   *************************************************************/
  public AuthenticationServer()
  {
    try{
       this.socket1815 = new DatagramSocket(this.PUERTO_1815);
       System.out.println("Servidor de autenticaciònescuchando en el puerto 1815");
       this.paquete_recibido = new DatagramPacket(this.datosRecibidos,this.datosRecibidos.length);
    }catch(java.io.IOException ex){
        ex.printStackTrace();
    }
    catch(Exception ex){
       ex.printStackTrace();
    }
}/// fin del constructor()

/*************************************************************
 * metodo que comienza la ejecuciòn del hilo
 *************************************************************/
public void run()
{
  while(true)/// ciclo que itera indefinidamente
  {
     try{
```

```
            /// esperar a recibir algÃ¹n datagrama
            this.socket1815.receive(this.paquete_recibido);

            /// generar un hilo que atienda la petición
            (new clases.PeticionProtocolo(this.paquete_recibido.getAddress(),
              this.paquete_recibido.getPort(),this.datosRecibidos,this.socket1815)).run();

            }catch(java.io.IOException ex)
            {
              System.out.println(ex.getMessage());
            }catch(Exception ex)
            {
              System.out.println(ex.getMessage());
            }
          }/// fin del ciclo que itera indefinidamente
        }/// fin del metodo run()

        /**************************************************************
         * CUERPO PRINCIPAL para ejecutar pruebas
         **************************************************************/
        public static void main(String args[])
        {
          AuthenticationServer authServer = new AuthenticationServer();
          authServer.run();
        }
      }/// fin de la clase AuthenticationServer
```

Clase ConsultaUsuario

Clase Datagramas

Clase GeneraNumeroAleatorio

Clase ListaAccesos

Clase PeticionProtocolo

```
/**************************************************
 * ARCHIVO : PeticionProtocolo.java
 *
 * Created on 28 de abril de 2005, 04:06 PM
 **************************************************/

package clases;
import java.io.*;
import java.net.*;
import clases.jdbc.ConsultasBD;
import clases.beans.Usuario;
import cripto.*;

/**************************************************
 * DESCRIPCION : Clase que atiende las peticiones del
 * servidor de autenticaciòn.
 * @author  Juan Francisco GarcÃ¬a Navarro
 **************************************************/
public class PeticionProtocolo extends Thread
{
```

```
////////////////////////////////////
///// DECLARACION DE ATRIBUTOS    /////
////////////////////////////////////
/** objeto que guarda la Dirección IP del usuario para respuesta **/
private InetAddress IPAddress = null;
/** variable que permite almacenar el puerto del usuario **/
private int puerto = 0;
/** arreglo que permite almacenar localmente los bytes recibidos **/
private byte[] datosRecibidos = null;
/** arreglo para guardar los bytes para enviar **/
private byte[] datosEnviar = null;
/** datagrama que se envÃ¬a **/
private DatagramPacket paquete_enviar  = null;
/** objeto para realizar las consultas a la base de datos  **/
private ConsultasBD consultas = null;
/** escuchar en el puerto 1815 los paquetes que vienen del access point **/
private DatagramSocket socket1815 = null;
/** objeto que permite guardar el tipo del paquete recibido,
para hacer algunas validaciones **/
private byte tipoPaquete = 0;
/** valor para guardar el texto cifrado **/
private String texto_cifrado = "";
/** valor para almacenar el texto ya descifrado **/
private String texto_descifrado = "";
/** objeto para cifrar usando el algoritmo DES **/
private AlgoritmoDES cifradorDES = null;
/** objeto que genera un nÃ'mero aleatorio **/
private GeneraNumeroAleatorio generador = null;
/** autenticador que se desea cifrar **/
private String autenticador = null;
/** arreglo de bytes para hacer una copia local de los bytes del datagrama
con el tamaño justo del datagrama **/
private byte[] datagramaRecibido = null;

/*************************************************************
 * CONSTRUCTOR : Inicializa valores
 * @param InetAddress IPAddress .- la Direcciòn IP del usuario
 * @param int puerto .- puerto de comunicaciones
 * @param byte[] datosRecibidos .- es el datagrama recibido
 * @param DatagramSocket socket .- el objeto para contestar
 *************************************************************/
public PeticionProtocolo(InetAddress IPAddress,int puerto,byte[] datosRecibidos,DatagramSocket socket)
{
  this.IPAddress = IPAddress;
  this.puerto = puerto;
  this.datosRecibidos = datosRecibidos;
  this.socket1815 = socket;
  /// instanciar el objeto que realiza las consultas a la base de datos
  this.consultas = new ConsultasBD();

}/// fin del constructor()

/*****************************************************
 * metodo que inicia la ejecuciòn del hilo
 *****************************************************/
public void run()
{
    /// variables temporales
 String texto_cifrado = "";
```

```
String login_temporal = "";
String llave = null;
String direccion_mac = null;
byte[] login = null;

try{
     /// copiar a otro arreglo de bytes, el datagrama recibido con el tamaño original
     copiaDatagrama();/// lanzar una excepciÃ²n

     /// desplegar el paquete recibido
     System.out.println("\n\n Datagrama recibido \n");
     Datagramas.despliegaPaquete(this.datagramaRecibido);

     /// obtener el valor del login dentro del paquete recibido
     login = Datagramas.getAtributo(datagramaRecibido,Datagramas.ATRIBUTO_LOGIN);
     if (login == null)
        throw new Exception("¡ERROR!, el paquete no contiene atributo login");

     login_temporal = new String(login);

     /// hacer una consulta a la base de datos para traer la llave del usuario
     llave = consultas.buscaLlave(login_temporal);

     /// validar que la llave exista, sino existe regresa un valor null
     if (llave == null)
        throw new Exception("¡ERROR!, esa llave para el cliente no existe");
     else
        System.out.println("llave: "+llave+"  longitÃ'd: "+llave.length());

     /// validar el datagrama bien formado, y que el hash este correcto
     if (!Datagramas.validaDatagrama(datagramaRecibido,llave))
        throw new Exception("¡ERROR!, el datagrama es invÃ lido");

     /// el tipo de paquete sesión no contiene autenticador
     /// pero los restantes si, por lo que hay que validarlo
     this.tipoPaquete = this.datagramaRecibido[0];

     if (this.tipoPaquete != Datagramas.PAQUETE_SESION)
     {

        /// descifrar el autenticador del paquete recibido
        /// instanciar el objeto cifrador con la llave de cifrado
        this.cifradorDES = new AlgoritmoDES(llave.getBytes());

        /// descifrar el autenticador
        texto_cifrado = Datagramas.getAutenticador(this.datagramaRecibido);

        /// validar que venga el autenticador
        if (texto_cifrado == null)
           throw new Exception("¡ERROR!, autenticador incorrecto");

        /// descifrar el autenticador
        this.texto_descifrado = this.cifradorDES.descifraDES(texto_cifrado);

        /// validar el texto descifrado
        if (texto_descifrado == null)
           throw new Exception("¡ERROR!, autenticaciònfallida");

     /// hacer una consulta para buscar la direccion mac
```

```
        direccion_mac = consultas.buscaMacXLogin(login_temporal);

        /// validar que exista la Direcciòn mac
        if (direccion_mac == null)
          throw new Exception("¡ERROR!, no existe la Direcciòn mac");
      }/// fin del condicional que revisa el tipo de paquete para validaciòn

    /// condicional que revisar el codigo del paquete para la respuesta
    switch (this.datosRecibidos[0])
    {
    ////-------------------------------------------\\\\\\
    ////  caso cuando el paquete es PETICION DE AUTENTICACION  \\\\\\
    ////-------------------------------------------\\\\\\
    case Datagramas.PETICION_AUTENTICACION:

        /// verificar el estado de la transaccion, debe estar en MAC_OK
        System.out.print("  el estado de la transaccion es: "+
          (ListaAccesos.getLista().getEstado(direccion_mac)));

        if ((ListaAccesos.getLista().getEstado(direccion_mac)) != ConsultaUsuario.MAC_OK )
          throw new Exception("¡ERROR!, no ha validado la tarjeta MAC");

        System.out.println("  autenticador descifrado  "+this.texto_descifrado+"\n");

        /// ahora generar el autenticador del servidor y concatenarlo al autenticador
        /// del usuario, el separador de los dos autenticadores es el pipe '|'
        /// generar un numero aleatorio
        this.generador = new GeneraNumeroAleatorio();

        /// cifrar el numero con la llave de cifrado del cliente
        this.autenticador = this.generador.getNumeroEnString();

        /// este autenticador hay que guardarlo en la ListaAccesos
        ListaAccesos.getLista().insertaAutenticador(login_temporal, autenticador);

        /// cifrar los dos autenticadores...
        this.texto_cifrado = this.cifradorDES.cifraDES(this.texto_descifrado+"|"+this.autenticador);
        System.out.println("El autenticador generado es: "+this.autenticador+
          "  y el autenticador cifrado es: "+this.texto_cifrado);

        /// construir el datagrama PAQUETE DE RESPUESTA DEL SERVIDOR AAA
        this.datosEnviar = Datagramas.generaPaquete(Datagramas.RESPUESTA_AAA,
            this.texto_cifrado,null,llave);

        /// enviar el paquete
        enviaDatagrama();

        /// en este momento hay que cambiar el estado a LOGIN_OK
        ListaAccesos.getLista().setEstado(direccion_mac, ConsultaUsuario.LOGIN_OK);

    break;/// fin de la opciòn paquete tipo PETICION_AUTENTICACION
    ////-------------------------------------------\\\\\\
    //// caso en el que el paquete es del tipo RESPUESTA_USUARIO  \\\\\\
    ////-------------------------------------------\\\\\\
    case Datagramas.RESPUESTA_USUARIO:

      /// leer el autenticador guardado en la lista de accesos
      this.autenticador = ListaAccesos.getLista().getAutenticador(login_temporal);
```

```
            /// validar el autenticador
            if (this.autenticador == null)
              throw new Exception("¡ERROR!, no existe el autenticador");

             /// validar el autenticador recibido
             if (validacion(this.texto_descifrado))
             {
               /// cuando la la autenticaciònes positiva, hay que poner la transaccion en el estado OK
               ListaAccesos.getLista().setEstado(direccion_mac, ConsultaUsuario.OK);
               System.out.println("AUTENTICACION POSITIVA con la mac: "+direccion_mac);
             }/// fin del condicional de autenticacion positiva
             else /// cuando la autenticacion es negativa, poner el estado RECHAZO
             {
               ListaAccesos.getLista().setEstado(direccion_mac, ConsultaUsuario.RECHAZO);
               System.out.println("AUTENTICACION NEGATIVA");
             }/// fin del condicional de la autenticacion negativa

             /// verificar el estado de la transaccion
             System.out.print("  el estado de la transaccion es: "+(ListaAccesos.getLista().getEstado(direccion_mac)));

             /// quitar de la tabla de autenticadores, este elemento
             ListaAccesos.getLista().eliminaAutenticador(login_temporal);

          break;/// fin de la opcion paquete de respuesta del usuario
          ////------------------------------------------\\\\\\
          ////  caso cuando el tipo de paquete es PAQUETE DE SESION    \\\\\\
          ////------------------------------------------\\\\\\
          case Datagramas.PAQUETE_SESION:
          /// guardar en la lista de sesiones este tiempo
            System.out.println("Actualizando el tiempo para: "+login_temporal);
            ListaAccesos.getLista().actualizaTiempo(login_temporal, System.currentTimeMillis());
            break;
          ////------------------------------------------\\\\\\
          //// caso en el que el paquete es de un tipo desconocido    \\\\\\
          ////------------------------------------------\\\\\\
          default:  /// obviamente es un caso de error, lanzar una excepciÃ²n
            throw new Exception("¡ERROR!, paquete invÃ lido");
          }/// fin del condicional que revisa el tipo del paquete
      }catch(Exception ex)
      {
        ex.printStackTrace();
        System.out.println(ex.getMessage());
      }finally
      {  /// una vez regresado o no el mensaje, destruir este hilo
        this.interrupt();
        this.consultas.desconectaBD();
      }
    }/// fin del metodo run()

    /**************************************************************
     * método que copia el datagrama recibido almacenado en el
     * arreglo datosRecibidos, al arreglo datagramaRecibido
     **************************************************************/
    private void copiaDatagrama() throws Exception
    {
      switch (this.datosRecibidos[0])
      {
        /// para este caso el tamaño depende de la cantidad de atributos
        /// y el tamaño de sus valores, unicamente considera un solo
```

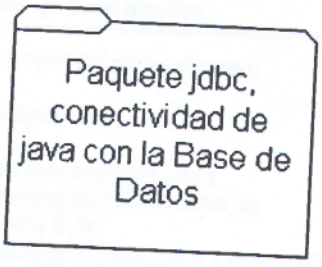
```
            /// atributo, en el caso de haber mas modificar esta secciÃ²n
            case Datagramas.RESPUESTA_USUARIO:
            case Datagramas.PETICION_AUTENTICACION:
              this.datagramaRecibido = new byte[41+this.datosRecibidos[42]];
              break;
            case Datagramas.RESPUESTA_AAA:
              this.datagramaRecibido = new byte[49];
              break;
            case Datagramas.PAQUETE_RECHAZO:
              break;
            case Datagramas.PAQUETE_SESION:
              this.datagramaRecibido = new byte[17+this.datosRecibidos[18]];
              break;
            default:
              throw new Exception("¡ERROR!, el datagrama es invÃ lido");
          }/// fin del condicional que revisa el codigo del datagrama
          for (int i=0;i<this.datagramaRecibido.length;i++)
            this.datagramaRecibido[i] = this.datosRecibidos[i];

        }/// fin del método copiaDatagrama()
        /*****************************************************************
         * metodo que envia un datagrama ya creado previamente
         *****************************************************************/
        private void enviaDatagrama()
        {
          try{
            /// crear el nuevo datagrama que se va a enviar
            this.paquete_enviar = new DatagramPacket(this.datosEnviar,this.datosEnviar.length,
              this.IPAddress,this.puerto);
            /// enviar el datagrama
            this.socket1815.send(this.paquete_enviar);
            System.out.println();
            System.out.print("Datagrama enviado al Access Point en el instante: "
              +((java.util.Calendar.getInstance()).get(java.util.Calendar.MINUTE))
              +" : "+((java.util.Calendar.getInstance()).get(java.util.Calendar.SECOND)));
            Datagramas.despliegaPaquete(this.datosEnviar);
          }catch(java.io.IOException ex)
          {
            System.out.println(ex.getMessage());
          }catch(Exception ex)
          {
            System.out.println(ex.getMessage());
          }
        }/// fin del metodo enviaDatagrama()
        /*****************************************************************
         * método que valida que el nÃºnico generado por el servidor sea
         * igual al valor regresado por el cliente.
         * El valor del autenticador esta guardado en la ListaAccesos
         *****************************************************************/
        private boolean validacion(String valor)
        {
          if (valor.equals(this.autenticador))
            return true;
          else
            return false;
        }

}//// fin de la clase PeticionProtocolo
```

Clase PeticionRadius

Clase ServidorRadius



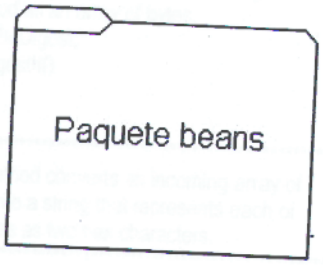
Este paquete contiene clases que permiten la conectividad con la base de datos.

Clase ConsultasBD

Clase EliminacionesBD

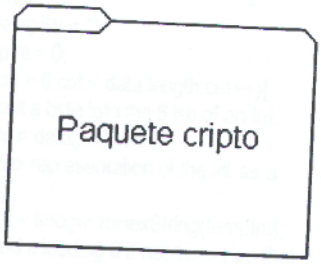
Clase InsercionesBD

Clase TablasBD



Este paquete contiene clases del tipo java beans.

Clase Usuario



Este paquete contiene clases que implementan los algoritmos de cifrado criptográfico.

Clase AlgoritmoDES

Clase MD5

Clase Sha01

```
package cripto;
import java.security.*;

public final class Sha01
```

```
{

 /**************************************************
  * This method generates and returns a digest
  * for an incoming array of bytes using Sun's
  * SHA message digest algorithm..
  **************************************************/
 public static byte[] digestIt(byte[] dataIn)
 {
  byte[] theDigest = null;
  try{
   //Create a MessageDigest object
   // implementing the SHA algorithm, as
   // supplied by SUN
   MessageDigest messageDigest = MessageDigest.getInstance("SHA", "SUN");
   //Feed the byte array to the digester.  Can
   // accommodate multiple calls if needed
   messageDigest.update(dataIn);
   //Complete the digestion and save the
   // result
   theDigest = messageDigest.digest();
  }catch(Exception e){System.out.println(e);}

  //Return the digest value to the calling
  // method as an array of bytes.
  return theDigest;
 }//end digestIt()


 /**************************************************
  *This method converts an incoming array of
  * bytes into a string that represents each of
  * the bytes as two hex characters.
  **************************************************/
 public static String byteArrayToHexStr(byte[] data)
 {
  String output = "";
  String tempStr = "";
  int tempInt = 0;
  for(int cnt = 0;cnt < data.length;cnt++){
   //Deposit a byte into the 8 lsb of an int.
   tempInt = data[cnt]&0xFF;
   //Get hex representation of the int as a
   // string.
   tempStr = Integer.toHexString(tempInt);
   //Append a leading 0 if necessary so that
   // each hex string will contain two
   // characters.
   if(tempStr.length() == 1)
              tempStr = "0" + tempStr;
   //Concatenate the two characters to the
   // output string.
   output = output + tempStr;
  }//end for loop
  return output.toUpperCase();
 }//end byteArrayToHexStr

/**************************************************
* mÃ¨todo que cifra con SHA-1
```

```
     ************************************************/
    public static byte[] cifrar(String texto) throws Exception
    {
       java.security.MessageDigest digest = java.security.MessageDigest.getInstance("SHA-1","SUN");
       digest.reset();
       digest.update(texto.getBytes());
       return digest.digest();
    }/// fin del mÃ¯todo cifrar()

  }//end class Sha01
```

Clase AES

```
/**********************************************************
 * ARCHIVO : AES.java
 * DESCRIPCION : Clase
 * Created on 2 de agosto de 2005, 02:20 PM
 **********************************************************/
   package cripto;
   import java.security.*;
   import javax.crypto.*;
   import javax.crypto.spec.*;
   import java.io.*;

 /**********************************************************************
  * This program generates a AES key, retrieves its raw bytes, and
  * then reinstantiates a AES key from the key bytes.
  * The reinstantiated key is used to initialize a AES cipher for
  * encryption and decryption.
  * @author Juan Francisco GarcÃ¬a Navarro
  **********************************************************************/
public class AES
{
    /**********************************************************************
     * convierte un arreglo debytes a una cadena de caracteres
     * @param buf       Array of bytes to convert to hex string
     * @return          Generated hex string
     **********************************************************************/
    public static String asHex (byte buf[])
    {
    StringBuffer strbuf = new StringBuffer(buf.length * 2);
    int i;

    for (i = 0; i < buf.length; i++)
    {
     if (((int) buf[i] & 0xff) < 0x10)
               strbuf.append("0");

     strbuf.append(Long.toString((int) buf[i] & 0xff, 16));
    }

   return strbuf.toString();
  }/// fin del metodo asHex()

/*********************************************************
*This method converts an incoming array of
* bytes into a string that represents each of
* the bytes as two hex characters.
*********************************************************/
```

```
    public static String byteArrayToHexStr(byte[] data)
    {
     String output = "";
     String tempStr = "";
     int tempInt = 0;
     for(int cnt = 0;cnt < data.length;cnt++){
      //Deposit a byte into the 8 lsb of an int.
      tempInt = data[cnt]&0xFF;
      //Get hex representation of the int as a
      // string.
      tempStr = Integer.toHexString(tempInt);
      //Append a leading 0 if necessary so that
      // each hex string will contain two
      // characters.
      if(tempStr.length() == 1)
                    tempStr = "0" + tempStr;
      //Concatenate the two characters to the
      // output string.
      output = output + tempStr;
     }//end for loop
     return output.toUpperCase();
    }//end byteArrayToHexStr

    /**************************************************************
     * mÃ¨todo que cifra usando AES como algoritmo
     * @param String texto.- es el mensaje a cifrar
     * @returns el arreglo de bytes cifrado
     **************************************************************/
    public static byte[] cifrarAES(String texto,byte[] llave)
    {
      byte[] encrypted = null;
      try{
          SecretKeySpec skeySpec = new SecretKeySpec(llave, "AES");

          // obtener la referencia del objeto cifrador
          Cipher cipher = Cipher.getInstance("AES");

          /// inicializar el cifrador
          cipher.init(Cipher.ENCRYPT_MODE, skeySpec);

          /// cifrar el arreglo de bytes
          encrypted = cipher.doFinal(texto.getBytes());
      }catch(java.security.NoSuchAlgorithmException ex){
         ex.printStackTrace();
      }catch(java.security.InvalidKeyException ex){
         ex.printStackTrace();
      }catch(javax.crypto.IllegalBlockSizeException ex){
         ex.printStackTrace();
      }catch(javax.crypto.NoSuchPaddingException ex){
         ex.printStackTrace();
      }catch(javax.crypto.BadPaddingException ex){
         ex.printStackTrace();
      }
      return encrypted;
    }/// fin del metodo cifrarAES()
    /**************************************************************
     * mÃ¨todo que cifra usando AES como algoritmo
     * @param String texto.- es el mensaje a cifrar
     * @returns el arreglo de bytes cifrado
```

```
 ************************************************************/
public static byte[] descifrarAES(byte[] texto,byte[] llave)
{
   byte[] encrypted = null;
   try{
       SecretKeySpec skeySpec = new SecretKeySpec(llave, "AES");

       // obtener la referencia del objeto cifrador
       Cipher cipher = Cipher.getInstance("AES");

       /// inicializar el cifrador
       cipher.init(Cipher.DECRYPT_MODE, skeySpec);

       /// cifrar el arreglo de bytes
       encrypted = cipher.doFinal(texto);
   }catch(java.security.NoSuchAlgorithmException ex){
      ex.printStackTrace();
   }catch(java.security.InvalidKeyException ex){
      ex.printStackTrace();
   }catch(javax.crypto.IllegalBlockSizeException ex){
      ex.printStackTrace();
   }catch(javax.crypto.NoSuchPaddingException ex){
      ex.printStackTrace();
   }catch(javax.crypto.BadPaddingException ex){
      ex.printStackTrace();
   }
   return encrypted;
}/// fin del metodo descifrarAES()

}/// fin de la clase AES
```