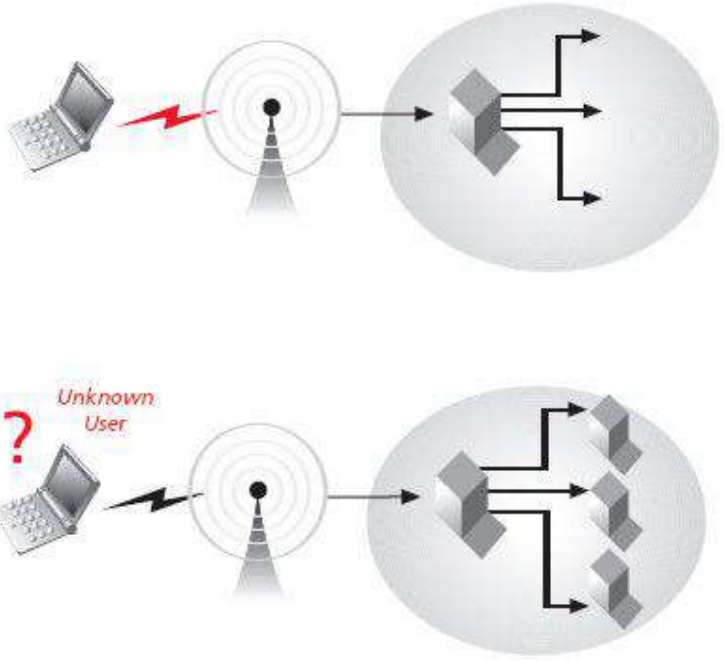
Universidad Nacional de Mar del Plata

PROYECTO FINAL

Facultad de Ingeniería

INCORPORACIÓN DEL ALGORITMO ADVANCED ENCRIPTION STANDARD EN REDES INALÁMBRICAS



Del Gesso, Leandro Martín

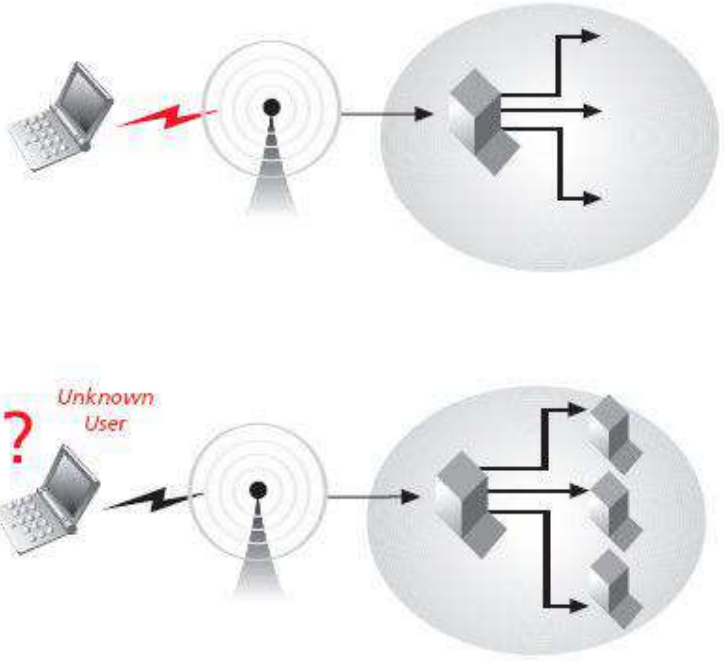
2007

Universidad Nacional de Mar del Plata

PROYECTO FINAL

Facultad de Ingeniería

INCORPORACIÓN DEL ALGORITMO ADVANCED ENCRIPTION STANDARD EN REDES INALÁMBRICAS



Del Gesso, Leandro Martín

2007

Agradecimientos

Principalmente quisiera agradecer a mi familia que me ha brindado la posibilidad de estudiar una carrera la cual ha cumplido con mis expectativas en su totalidad, y me dieron todo lo necesario para llevarla a cabo sin que nunca me falte nada.

Quiero agradecer a Dios por haberme hecho la persona que soy, por la familia que me ha dado, por los amigos y conocidos, por el entorno en el que vivo y por haber podido incorporar no solo la enseñanza sino también el trato humano aportado por la Facultad de Ingeniería a través de profesores y compañeros.

También mis más sinceros agradecimientos a: La Universidad Nacional de Mar del Plata a la cual le debo muchísimo y tengo como objetivo devolvérselo a lo largo de toda mi vida, y a La Facultad de Ingeniería por la inestimable formación que me brindó.

Finalmente, un agradecimiento especial al Laboratorio de Comunicaciones, fundamentalmente a mis tutores Juan Carlos Bonadero y Mónica Liberatori por tantas veces que los molesté e insistí, y en quienes siempre encontré una muy buena predisposición para atender mis preguntas.

INDICE

RESUMEN

Se presenta un algoritmo de encriptación Advanced Encryption Standard (AES) en lenguaje C, y su incorporación a una placa para red inalámbrica de propósitos generales DWL-G520+ provista por D-Link.

La primer parte consistió en el desarrollo de tres archivos ejecutables con sus respectivos códigos fuente los cuales consistieron en:

1. *El algoritmo de encriptación:* pide los datos a encriptar, la clave y muestra o no (según desee el usuario) todas las operaciones que se efectúan sobre los datos iniciales para finalmente llegar a los datos encriptados que son mostrados en pantalla.

2. *El algoritmo de desencriptación:* pide los datos a desencriptar, la clave y muestra o no (según desee el usuario) todas las operaciones que se efectúan sobre los datos de entrada para finalmente llegar a los datos desencriptados que son volcados en pantalla.

3. *Los algoritmos de encriptación y desencriptación juntos*: pide los datos a encriptar, a desencriptar y la clave. El ejecutable muestra o no las operaciones y como opera el sistema en su totalidad desde el comienzo con la encriptación hasta la desencriptación la cual obviamente arrojará los mismos datos de entrada.

Los resultados obtenidos fueron satisfactorios debido a que permitieron encriptar y desencriptar los datos binarios introducidos por teclado.

En una segunda etapa, se incorporó el algoritmo AES al driver Linux del adaptador inalámbrico DWL-G520+, que sólo proporciona el sistema de

encriptación elemental denominado Wired Equivalent Privacy (WEP), a fin de que sea compatible con cualquier red inalámbrica que utilice AES. De esta forma se puede implementar el AES en placas de cualquier marca y serie en forma sencilla, conociendo el formato de los drivers.

Con el soporte de un Punto de Acceso DWL-7100 que posee el Laboratorio de Comunicaciones, el cual soporta AES, se implementó el algoritmo en el driver de Linux (slackware 11.0) agregando archivos de encabezamiento y líneas de código adicionales al código primitivo del driver.

Fue incluida en el driver la programación adicional para soportar WPA/PSK – AES (Wi-Fi Protected Access/Pre-shared-key), que permite compatibilizar la placa con dicho sistema.

INTRODUCCIÓN

Seguridad en transmisiones inalámbricas

Los sistemas de encriptación ocupan un lugar muy importante en el mundo de las redes de datos.

Éstos deben su existencia a la necesidad de transmisión de datos con carácter privado, en medios donde integrantes de la red, quienes no son destinatarios del mensaje, pueden interceptar la señal (como en las redes inalámbricas por ejemplo) con lo que la transmisión no sería privada. Ésta es la fundamental razón de la seguridad en transmisiones.

Hay diversos beneficios que puede aportar la seguridad en transmisiones inalámbricas.

En las transmisiones militares fue donde primero se advirtió que sería muy perjudicial, que el enemigo captara las transmisiones del aire y así conociera acerca de las acciones a tomar. Se observa que es esencial asegurar de alguna forma la transmisión.

Yendo a lugares no tan lejanos o no tan extremos, el caso de hoteles o cybers que ofrecen el servicio de Wi-Fi con su respectivo abono, sería muy fácil ubicarse en cercanías de la señal de Wi-Fi y tener acceso a Internet sin pago alguno si la transmisión no estuviera encriptada.

Se pueden citar muchos ejemplos en los cuales que es necesario cifrar los datos para que solo puedan ser entendidos por el destino deseado y así poder establecer un link seguro.

Los algoritmos de encriptación modifican los datos a trasmitir mediante un proceso de criptografía y el agregado de una clave que sólo conocen los

dos extremos de la conexión; de esta forma pueden encriptar y desencriptar los datos, no siendo ésto posible a usuarios desconocidos.

Breve historia de AES

El algoritmo de criptografía AES, también conocido como *Rijndael*, es un esquema de cifrado por bloques adoptado como un estándar de encriptación por el gobierno de los Estados Unidos. Se espera que sea usado en el mundo entero y analizado exhaustivamente. AES fue anunciado por el *Instituto Nacional de Estandares y Tecnología* (NIST) como *FIPS PUB 197* de los Estados Unidos (FIPS 197) el 26 de noviembre de 2001 después de un proceso de estandarización que duró 5 años. Se transformó en un estándar efectivo el 26 de mayo de 2002. Desde 2006, el AES es uno de los algoritmos más populares usados en criptografía simétrica por su gran robustez criptológica.

El cifrado fue desarrollado por dos criptólogos belgas, Joan Daemen y Vincent Rijmen, ambos estudiantes de la *Katholieke Universiteit Leuven*.

Básicamente el AES encripta bloques de 128 bits (16 bytes) los cuales son distribuidos en una matríz de 4 x 4 llamada *state*. Luego efectúa operaciones sobre el state con el agregado de la clave de 128 bits para finalmente entregar 128 bits de datos de salida encriptados.

En una red inalámbrica el AES funciona junto con el algoritmo conocido como Counter Mode + cipher block chaining-message authentication code Protocol.

Cipher block chaining – message authtentication code se conoce por las siglas CBC-MAC y a todo el protocolo mencionado se lo denomina CCMP.

El AES no se ingresa tan sencillamente en las redes inalámbricas como se pudiera pensar en principio, pues se debe cumplir con una serie de pasos

para lograr la transferencia satisfactoria de datos entre el adaptador y el punto de acceso.

Evolución de 802.11

IEEE ha ido evolucionando en concepto de redes inalámbricas comenzando con el estándar IEEE 802.11 y pasando por IEEE 802.11a, IEEE 802.11b, IEEE 802.11g y el último publicado en el 2004, IEEE 802.11i.

En el primer estándar se proporcionaba la forma de encriptado de datos WEP (Wired Equivalent Privacy), el cual se ha demostrado muy probre en términos criptográficos, además no se establecía método de autenticación (que sucede luego de la asociación y antes de la transferencia neta de datos encriptados).

WEP, TKIP y AES

WEP ofrece claves de 40 bits o de 104 bits de longitud pero presenta diversas debilidades (para más detalles ver el ítem 2 de la bibliografía, capítulo 6).

La prueba salta a la vista con solo poner las palabreas claves "WEP" y "crack" en un buscador de Internet para observar los cientos de programas que luego de recoger una gran cantidad de paquetes transferidos por la interface inalámbrica calculan mediante estadística la clave usada y así se concreta el craqueado del sistema. Una vez sabida la clave, se puede introducir en la red inalámbrica con total facilidad.

Para solucionar el problema de WEP, se inventaron 2 sistemas de seguridad a lo largo de la evolución denominados Temporal Key Integrity Protocol (TKIP) y Advanced Encryption Standard (AES). Véase capítulos 8, 10, 11 y 12 del ítem 2 para una explicación detallada.

Brevemente TKIP utiliza el algoritmo RC4 (también utilizado por WEP) para poder implementarlo en los adaptadores que soportan solo WEP, y elimina las debilidades encontradas en WEP.

CCMP/AES es un método totalmente nuevo y al desarrollarlo, se buscó una solución con alta seguridad.

Además, IEEE 802.11i incorpora la autenticación para impedir el ingreso a la red de cualquiera que no conozca la clave master.

Para llevar a cabo la segunda etapa del Proyecto se tuvo que recurrir al sistema operativo Linux, que a diferencia de Windows, proporciona código abierto, y se debió interiorizar en programación en C, dado que Linux se maneja con ese lenguaje.

El adaptador DWL-G520+ y el Punto de Acceso DWL-7100 son ambos de la marca D-LINK.

Posibles proyectos Finales

Éste proyecto podría ser el origen de otros proyectos finales tales como:

- Desarrollar la misma idea pero bajo Windows, con la dificultad de conseguir el código fuente del driver.
- Implementar el sistema TKIP para la mima placa inalámbrica y sobre el mismo driver modificado para CCMP/AES, para tener total compatibilidad con cualquier sistema de encriptación.
- Por último, dado que en éste proyecto se implementó WPA-PSK/AES, podría implementarse otra configuración existente que es la WPA-EAP/ AES o TKIP (ésta implementa servidor radius sobre la red cableada), el cual involucra protocolo de capas superiores.

PROYECTO

Primera etapa:

Aes en detalle

La primer etapa del Proyecto consistió en el desarrollo de los programas de encriptación y desencriptación mencionados anteriormente. Para ello se debió estudiar en detalle el funcionamiento del algoritmo AES, el cual se pasa a describir.

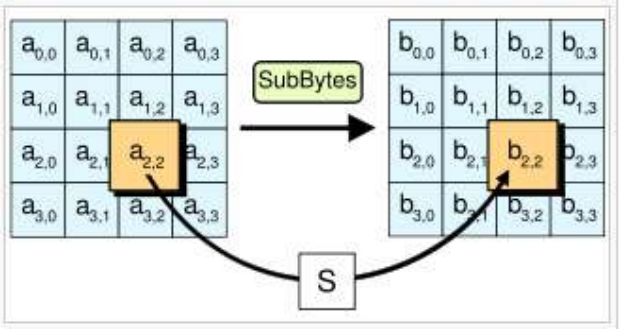
El AES está desarrollado y justificado en el documento Fips 197, inclusive posee ejemplos para poder comprobar el funcionamiento. Los tres programas de la primer parte del proyecto se basan en dicho documento.

Como ya se mencionó anteriormente, los tres programas se desarrollaron según el siguiente esquema: primero se ingresan los datos que conforman el *state*, luego la clave, consulta si se desean visualizar las operaciones sobre el *state* en pantalla y por último muestra el *state* resultante del algoritmo correspondiente.

Aquí se presenta una explicación sintética del funcionamiento del AES.

AES hace 4 tipos distintos de operaciones sobre el *state,* y las repite 10 veces (10 rondas). Previamente, toma la clave de 128 bits y se desarrollan 10 claves nuevas para emplear en cada una de las 10 rondas. La primera (y original) se agrega antes de ejecutar las rondas.

La primera operación dentro de las rondas es "*Sub_Bytes*" y básicamente consiste en la sustitución de cada byte del *state* por otro contenido en una tabla llamada S-box, la cual deriva del Inverso multiplicativo sobre GF(2^8). SubBytes provee buenas propiedades de no-linealidad.

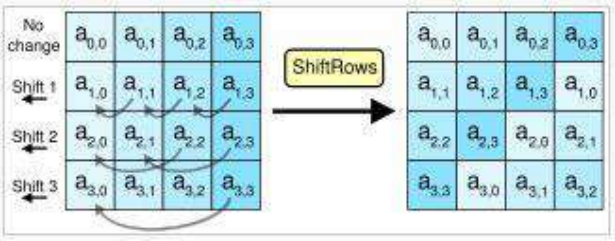


Sub_bytes

S-box es una tabla de 256 valores con lo cual, el valor del byte correspondiente del *state* ingresa a la *S-box* como un subíndice que indica la posición dentro de ella, y el valor de la tabla en esa posición es el que reemplazará al byte de entrada.

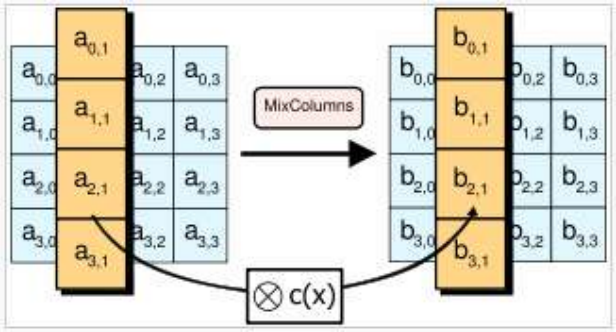
Los valores de *S-box* se encuentran en el documento fips 197 al igual que la tabla inversa de la *S-box* utilizada en la desencriptación.

Se continúa con "*Shift_Rows*". Como se ve en la figura 2, si se recorren i filas con i desde 0 hasta 3, la operación efectúa i corrimientos de los bytes de la fila i hacia la izquierda.



Shift_rows

La siguiente operación es "*Mix_Columns*", y consiste en tomar cada columna del *state* y hacer operaciones matemáticas con todas las filas de cada correspondiente columna. Las operaciones son en aritmética finita. En la última ronda esta operación no se realiza.



Mix_coloumns

Las operaciones matemáticas son:

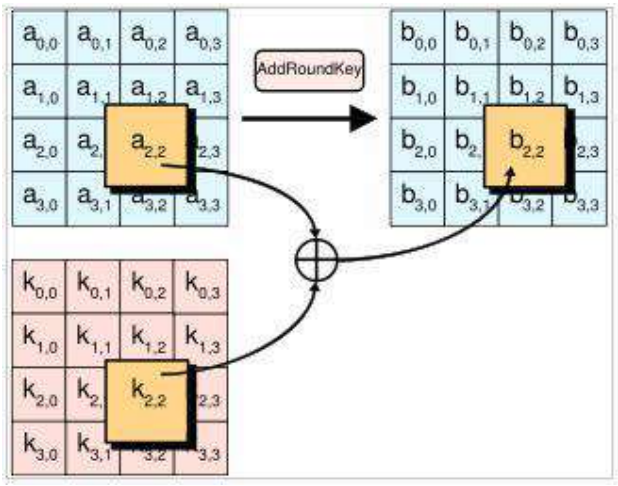
$$\begin{bmatrix} S'_{0,c} \\ S'_{1,c} \\ S'_{2,c} \\ S'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} S_{0,c} \\ S_{1,c} \\ S_{2,c} \\ S_{3,c} \end{bmatrix}$$

Donde $S_{i,c}$ son los bytes de cada columna y $S'_{i,c}$ son los nuevos valores con $0 <= i <= 3$.

Por último se agrega la correspondiente clave con "*Add_Round_key*".

En esta última ronda, se suma la clave de la ronda al *state* mediante la operación x-or (or exclusiva). La clave se ordena en forma de matriz, de la misma dimensión que el *state* para que el resultado tenga la misma forma.

La expansión de claves es preferible observarla en Fips 197.



Add_round_key

La desencriptación obviamente consiste en el proceso inverso al anterior por lo que es fundamental el conocimiento de la clave que se usó al encriptar. Además las operaciones son inversas a las anteriores: en vez de *sub_bytes* se reemplaza por *inv_sub_bytes* la cual utiliza una tabla inversa de la *S-Box*, *shift_rows* cambia por *inv_shift_rows* la cual hace rotaciones hacia la derecha, *mix_columns* es cambiada por *inv_mix_columns*, que utiliza la siguiente expresión:

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix}$$

La función inversa de *Add_round_key* es la misma que en el encriptado, ya que si se efectúa x-or dos veces se llega al valor inicial.

Se puede ver que AES utiliza como centro de sus cálculos la matriz *state* de 4 x 4, dos tablas S-box e inversa de S-box de 256 valores cada una, un arreglo de 44 elementos de 32 bits que almacena las claves de cada ronda y

por último un arreglo (rcon) de 10 componentes utilizado en la expansión de claves.

A continuación se muestran los diagramas de flujo de los procesos de encriptación y desencriptación separadamente.

Diagrama de flujo de la encriptación

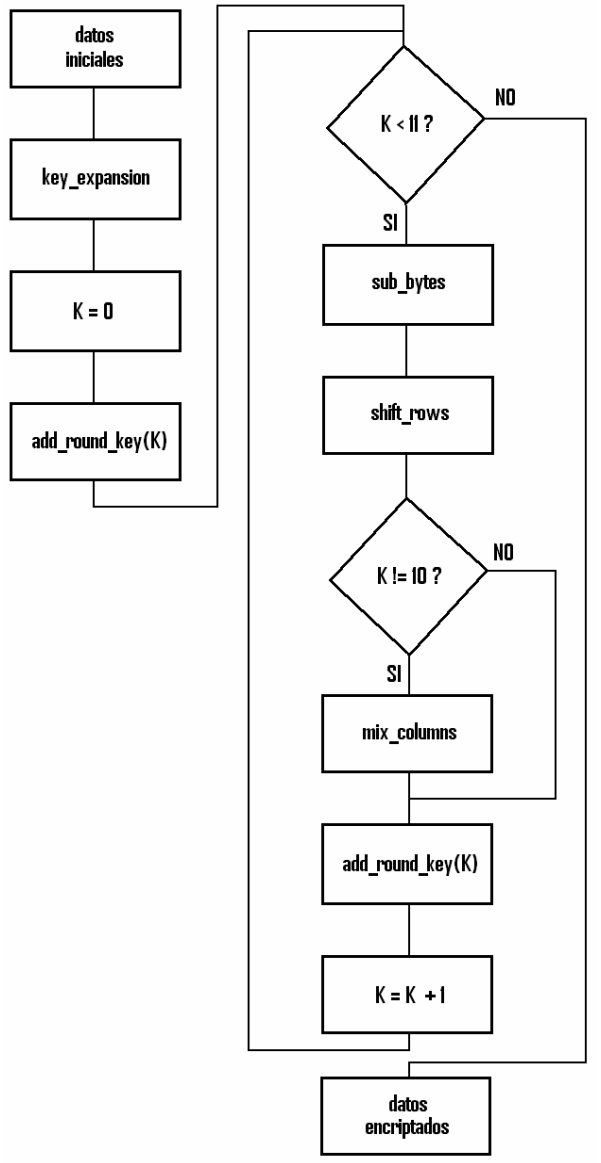
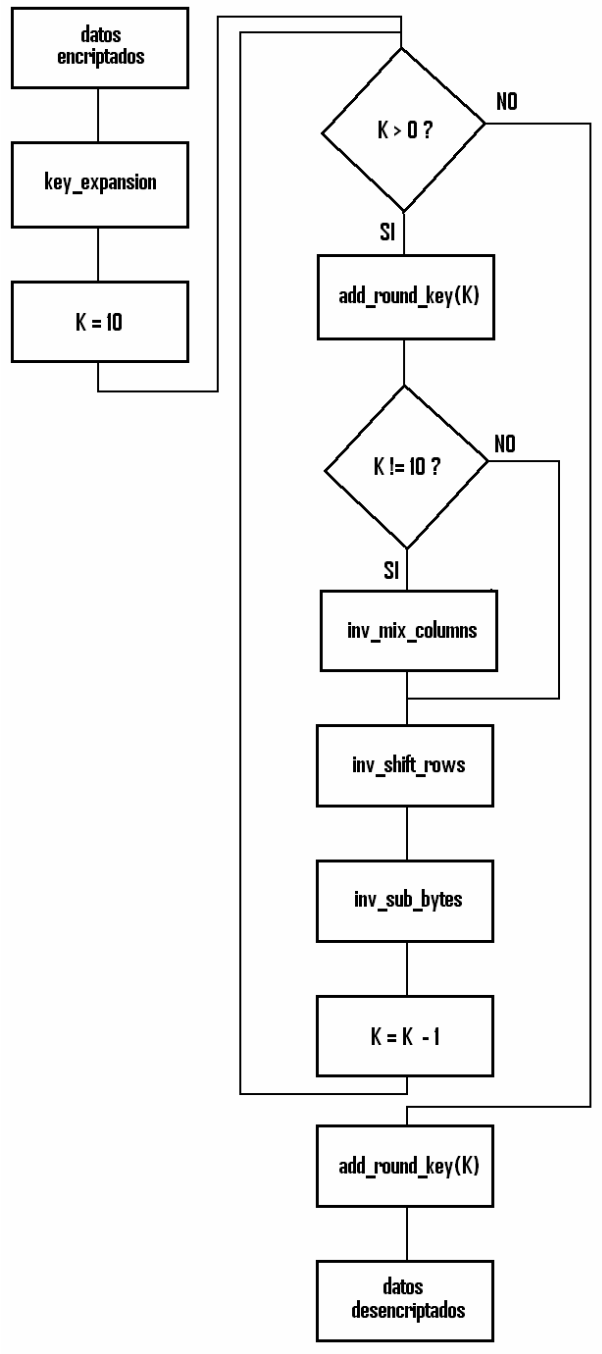


Diagrama de flujo de la desencriptación

El diagrama de flujo de desencriptación es el siguiente:



El tercer programa hace los 2 procesos, uno seguido del otro.

Yendo al código fuente de cada uno de los programas, en el aestx.c se tienen como variables globales: s_box[256], state[4][4], key_array[44] y rcon[10]. La decisión que motivó que estas variables sean globales fue para adquirir velocidad en el alogoritmo. La misma razón por la cual existen las variables globales de los otros dos programas.

Se incluyeron funciones para el ingreso de datos por teclado y la muestra de mensajes en pantalla. Éstas son *init* y *show_state*.

El programa comienza haciendo una breve presentación en pantalla, y luego llama a la función init para ingresar los datos a encriptar. Luego llama a *copy_data_to_state* que copia los datos ingresados desde una variable de tipo array de 16 valores al *state* y a partir de aquí trabajar sobre el *state*.

A continuación se llama a la función *get_master_key* para ingresar la clave que será usada en la encriptación.

Luego se llama a la función *key_expansion* para llevar a cabo la expansión de claves para cada ronda. *Key_expansion* utiliza a su vez otras funciones: *sub_word*, *rot_word* que se encuentran definidas en el FIPS 197.

Finalizada la carga de datos, clave y desarrollo de las claves de ronda se comienza con el algoritmo de encriptación no sin antes de preguntarle al usuario si desea ver las operaciones en pantalla sobre el *state*.

Luego del algoritmo, se llama a la función *copy_state_to_data*, que hace el proceso inverso a *copy_state_to data*, finalmente los datos encriptados son mostrados en pantalla.

Las funciones esenciales para el funcionamiento del algoritmo son: *sub_bytes*, *shift_rows*, *mix_columns*, *mult* y *add_round_key*.

Para el programa aesrx.c es más fácil la explicación por que el código es similar al aestx.c pero con el simple detalle de que se realiza la desencriptación.

Una diferencia es que tiene la variable *inv_s_box* necesaria para la desencriptación. Además, *s_box* se mantiene, ya que se utiliza en el desarrollo de claves en los dos extremos: el transmisor y el receptor.

Finalmente en el algoritmo de desencriptado están las funciones inversas antes mencionadas: *inv_s_box*, *inv_shift_rows* e *inv_mix_columns*. Como ya se mencionó, add_round_key es simplemente una operación de or exclusiva entre dos matrices.

Luego de la explicación de los dos primeros programas, el tercero es simplemente la concatenación de ellos, por lo que aes.c lleva a cabo el encriptado y el desencriptado mediante los mecanismos expuestos anteriormente.

Aes.c encripta y desencripta y muestra los datos encriptados y desencriptados. Éstos últimos obviamente son los mismos que los ingresados en el principio del programa, pero sirven para la verificación del funcionamiento del programa.

Código fuente de los tres programas

aestx.c

Código del primer programa (aestx.c)

```
#include <stdio.h>
#include <conio.h>

/*Variables globales*/

unsigned char state[4][4];

unsigned char s_box[] = {
    0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67, 0x2b, 0xfe, 0xd7,
0xab, 0x76,
    0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2, 0xaf, 0x9c, 0xa4,
0x72, 0xc0,
    0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5, 0xf1, 0x71, 0xd8, 0x31,
0x15,
    0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12, 0x80, 0xe2, 0xeb, 0x27,
0xb2, 0x75,
    0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b, 0xd6, 0xb3, 0x29, 0xe3,
0x2f, 0x84,
    0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb, 0xbe, 0x39, 0x4a, 0x4c,
0x58, 0xcf,
    0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9, 0x02, 0x7f, 0x50, 0x3c, 0x9f,
0xa8,
    0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6, 0xda, 0x21, 0x10, 0xff, 0xf3,
0xd2,
    0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7, 0x7e, 0x3d, 0x64, 0x5d,
0x19, 0x73,
    0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8, 0x14, 0xde, 0x5e,
0x0b, 0xdb,
    0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3, 0xac, 0x62, 0x91, 0x95,
0xe4, 0x79,
    0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4, 0xea, 0x65, 0x7a,
0xae, 0x08,
    0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd, 0x74, 0x1f, 0x4b, 0xbd,
0x8b, 0x8a,
    0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35, 0x57, 0xb9, 0x86, 0xc1,
0x1d, 0x9e,
    0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e, 0x87, 0xe9, 0xce, 0x55,
0x28, 0xdf,
    0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d, 0x0f, 0xb0, 0x54,
0xbb, 0x16};

unsigned long key_array[44];
unsigned long  rcon[] = {
   0x01000000, 0x02000000, 0x04000000, 0x08000000, 0x10000000,
   0x20000000, 0x40000000, 0x80000000, 0x1b000000, 0x36000000};

/*Prototipos*/
```

```
void init(unsigned char data_tx[16]);
void show_state(void);

/*funciones de creación del arreglo de claves (ejecutadas una sola vez)*/

unsigned long get_master_key(unsigned char i);
void key_expansion(unsigned long master_key[4]);
unsigned long sub_word(unsigned long temp);
unsigned long rot_word(unsigned long temp);

/*funciones esenciales para el algoritmo, llamadas cada vez q se desea encriptar datos*/

void copy_data_to_state(unsigned char data_tx[16]);

void sub_bytes(void);
void shift_rows(void);
void mix_columns(void);
unsigned char mult(unsigned char n, unsigned char m);
void add_round_key(unsigned char round);

void copy_state_to_data(unsigned char data_tx[16]);


/*******************Definiciones******************/

void copy_data_to_state(unsigned char data_tx[16]) {
   unsigned char i,j;

   for(j=0;j<4;j++){
            for(i=0;i<4;i++) {
                     state[i][j] = data_tx[4*j + i];
            }
   }
}

void copy_state_to_data(unsigned char data_tx[16]) {
   unsigned char i,j;

   for(j=0;j<4;j++){
            for(i=0;i<4;i++) {
                     data_tx[4*j + i] = state[i][j];
            }
   }
}

unsigned long get_master_key(unsigned char i){        /*Pide la Master Key*/
   unsigned long key;

   printf(" Ingrese la Master_key[%d] en hexadecimal(32 bits, 4 bytes): ",i);
   scanf("%x",&key);
   return key;
}

void key_expansion(unsigned long master_key[4]){
```

```
    const char Nk = 4, Nb = 4, Nr = 10;
    unsigned char i,j;
    unsigned long temp;

    for(i = 0;i < Nk;i++) key_array[i] = master_key[i];

    i = Nk;

    while (i < (Nb * (Nr + 1))){
            temp = key_array[i - 1];
            if ((i % Nk) == 0) {
                    j = (unsigned char) i/Nk;

                    temp = rot_word(temp);

                    temp = sub_word(temp) ^ rcon[j - 1];

            }
            else {
                    if ((Nk > 0x6) && ((i % Nk) == 0x4)) temp = sub_word(temp);
            }
            key_array[i] = key_array[i - Nk] ^ temp;
            i++;
    }
}

unsigned long rot_word(unsigned long temp){
    unsigned char i;

    for(i = 0;i < 8;i++){
            if ((temp & 0x80000000) != 0) temp = ((temp << 1) ^ 0x1);
            else temp = (temp << 1);
    }
    return temp;
}

unsigned long sub_word(unsigned long temp){
    unsigned char temp1, temp2, temp3, temp4;
    unsigned long temp5, temp6, temp7, temp8;

    temp1 = (unsigned char) temp;
    temp1 = s_box[temp1];

    temp2 = (unsigned char) (temp >> 8);
    temp2 = s_box[temp2];

    temp3 = (unsigned char) (temp >> 16);
    temp3 = s_box[temp3];

    temp4 = (unsigned char) (temp >> 24);
    temp4 = s_box[temp4];

    temp5 = (unsigned long) temp1;
    temp6 = (unsigned long) temp2;
    temp7 = (unsigned long) temp3;
```

```
    temp8 = (unsigned long) temp4;

    temp5 = temp5 + (temp6 << 8) + (temp7 << 16) + (temp8 << 24);

    return temp5;
}

void sub_bytes(void){
    unsigned char i,j;
    const unsigned char c = 0x63;

    for(i=0;i<4;i++){

            for(j=0;j<4;j++) {
                    state[i][j] = s_box[state[i][j]];
            }
    }
}

void shift_rows(void){
    unsigned char i,j,a[3];

    for(i = 1;i < 4;i++){
            j = 0;
            while (j < i) {
                    a[j] = state[i][j];
                    j++;
            }
            j = 0;
            while (j < (4 - i)){
                    state[i][j] = state[i][j+i];
                    j++;
            }
            j = 0;
            while (j < i){
                    state[i][3 - j] = a[i - 1 - j];
                    j++;
            }
    }
}

void mix_columns(void){
    unsigned char j, vect[4];

    for(j = 0;j < 4;j++){
vect[0] = (mult(0x2,state[0][j])) ^ (mult(0x3,state[1][j])) ^ state[2][j] ^ state[3][j];
vect[1] = state[0][j] ^ (mult(0x2,state[1][j])) ^ (mult(0x3,state[2][j])) ^ state[3][j];
vect[2] = state[0][j] ^ state[1][j] ^ (mult(0x2,state[2][j])) ^ (mult(0x3,state[3][j]));
vect[3] = (mult(0x3,state[0][j])) ^ state[1][j] ^ state[2][j] ^ (mult(0x2,state[3][j]));
            state[0][j] = vect[0];
            state[1][j] = vect[1];
            state[2][j] = vect[2];
            state[3][j] = vect[3];
    }
}
```

```
unsigned char mult(unsigned char m, unsigned char n){
   unsigned int s;

   n = (unsigned int) n;
   if (m == 0x2){ /*multiplicación x 2*/
            s = n << 1;
/*si luego de la mult x 2 supera 255 entonces hay aplicar módulo x8 + x4 + x3 + x + 1*/
            if (s > 255) s = (s & 0xff) ^ (0x1b);
   }
   else{
            s = (n << 1) ^ n; /*multiplicación x 3*/
/*si luego de la mult x 3 supera 255 entonces hay que aplicar módulo x8 + x4 + x3 + x
+ 1*/
            if (s > 255) s = (s & 0xff) ^ (0x1b); /*descarte de b8 comenzando desde b0*/
   }
   s = (unsigned char) s;
   return s;
}

void add_round_key(unsigned char round){
   unsigned char i,j, key[4][4];

   for(i = 0;i < 4;i++){
            key[3][i] = (unsigned char) key_array[(4 * round) + i];
            key[2][i] = (unsigned char) (key_array[(4 * round) + i] >> 8);
            key[1][i] = (unsigned char) (key_array[(4 * round) + i] >> 16);
            key[0][i] = (unsigned char) (key_array[(4 * round) + i] >> 24);
   }
   for(i = 0;i < 4;i++){
            for(j = 0;j < 4;j++) state[i][j] = state[i][j] ^ key[i][j];
   }
}

void init(unsigned char data_tx[16]) {
   unsigned char i, ch;

   for(i=0;i<16;i++) {

            printf(" Ingrese el valor data_rx[%d]: ",i);

            scanf("%x", &ch);

            data_tx[i] = ch;

   }
}

void show_state(void){
   unsigned char i,j;

   for(i=0;i<4;i++){
            printf("\n");
            for(j=0;j<4;j++) {
                     printf(" State[%d][%d] es: %x",i,j,state[i][j]);
```

```
            }
    }
}


void main(void) {

    unsigned char data_tx[16], k;
    int c;
    unsigned long master_key[4];

    printf("                                Proyecto Final\n\n");
    printf("Presentacion del Sistema de Encriptacion AES en lenguaje C \n\n");
    printf("                                Integrante\n\n");
    printf("                          Leandro Martin Del Gesso\n\n");
    printf("                                Tutores\n\n");
    printf("              Juan Carlos Bonadero, Monica Liberatori\n\n");

    printf("               Aqui se observara la encriptacion\n\n");
    printf("A continuacion se le pedira ingresar los datos a encriptar\n");
    printf("(16 bytes) que conformaran el State\n\n");

    init(data_tx);
    copy_data_to_state(data_tx);

    printf("\n                                State inicial");
    show_state();

    printf("\n\n    A continuacion se le pedira ingresar la clave master\n");
    printf("(128 bits) en 4 partes de 32 bits (4 bytes)\n\n");
    for(k = 0;k < 4;k++) master_key[k] = get_master_key(k);
    key_expansion(master_key);

    printf("\n\n Desea ver todas las operaciones sobre el state? (s/n, default: n)\n ");
    scanf("%s",&c);
    c = (unsigned char) c;

    /*Luego de inicializar todo, comienza la encriptación*/

    k = 0;

    add_round_key(k);

    if (c == 's') {
            printf("\n                                State luego de primer add_round_key");

            show_state();
    }

    for(k = 1;k < 11;k++){

            sub_bytes();

            if (c == 's') {
printf("\n                        State luego de Sub_bytes ronda %d",k);
```

```
                show_state();
        }

        shift_rows();

        if (c == 's') {
printf("\n                State luego de shift_rows ronda %d",k);
                show_state();
        }

        if (k != 10) {
                mix_columns();

                if (c == 's') {
printf("\n                State luego de mix columns ronda %d",k);
                show_state();
                }

        }

        add_round_key(k);

        if (c == 's') {
printf("\n                State luego de add_round_key ronda %d",k);
                show_state();
        }

   }

   copy_state_to_data(data_tx);

   printf("\n\nLos datos encriptados son: ");
   for(k = 0;k < 16;k++) printf("%x ",data_tx[k]);
   printf("\n\n");
   getch();
}
```

aesrx.c

Código del segundo programa (aesrx.c):

```
#include <stdio.h>
#include <conio.h>

/*Variables globales*/

unsigned char state[4][4];
unsigned char inv_s_box[] ={
0x52, 0x09, 0x6a, 0xd5, 0x30, 0x36, 0xa5, 0x38, 0xbf, 0x40, 0xa3, 0x9e, 0x81, 0xf3,
0xd7, 0xfb,
0x7c, 0xe3, 0x39, 0x82, 0x9b, 0x2f, 0xff, 0x87, 0x34, 0x8e, 0x43, 0x44, 0xc4, 0xde, 0xe9,
0xcb,
0x54, 0x7b, 0x94, 0x32, 0xa6, 0xc2, 0x23, 0x3d, 0xee, 0x4c, 0x95, 0x0b, 0x42, 0xfa,
0xc3, 0x4e,
```

```
    0x08, 0x2e, 0xa1, 0x66, 0x28, 0xd9, 0x24, 0xb2, 0x76, 0x5b, 0xa2, 0x49, 0x6d, 0x8b,
0xd1, 0x25,
    0x72, 0xf8, 0xf6, 0x64, 0x86, 0x68, 0x98, 0x16, 0xd4, 0xa4, 0x5c, 0xcc, 0x5d, 0x65,
0xb6, 0x92,
    0x6c, 0x70, 0x48, 0x50, 0xfd, 0xed, 0xb9, 0xda, 0x5e, 0x15, 0x46, 0x57, 0xa7, 0x8d,
0x9d, 0x84,
    0x90, 0xd8, 0xab, 0x00, 0x8c, 0xbc, 0xd3, 0x0a, 0xf7, 0xe4, 0x58, 0x05, 0xb8, 0xb3,
0x45, 0x06,
    0xd0, 0x2c, 0x1e, 0x8f, 0xca, 0x3f, 0x0f, 0x02, 0xc1, 0xaf, 0xbd, 0x03, 0x01, 0x13, 0x8a,
0x6b,
    0x3a, 0x91, 0x11, 0x41, 0x4f, 0x67, 0xdc, 0xea, 0x97, 0xf2, 0xcf, 0xce, 0xf0, 0xb4, 0xe6,
0x73,
    0x96, 0xac, 0x74, 0x22, 0xe7, 0xad, 0x35, 0x85, 0xe2, 0xf9, 0x37, 0xe8, 0x1c, 0x75,
0xdf, 0x6e,
    0x47, 0xf1, 0x1a, 0x71, 0x1d, 0x29, 0xc5, 0x89, 0x6f, 0xb7, 0x62, 0x0e, 0xaa, 0x18,
0xbe, 0x1b,
    0xfc, 0x56, 0x3e, 0x4b, 0xc6, 0xd2, 0x79, 0x20, 0x9a, 0xdb, 0xc0, 0xfe, 0x78, 0xcd,
0x5a, 0xf4,
    0x1f, 0xdd, 0xa8, 0x33, 0x88, 0x07, 0xc7, 0x31, 0xb1, 0x12, 0x10, 0x59, 0x27, 0x80,
0xec, 0x5f,
    0x60, 0x51, 0x7f, 0xa9, 0x19, 0xb5, 0x4a, 0x0d, 0x2d, 0xe5, 0x7a, 0x9f, 0x93, 0xc9,
0x9c, 0xef,
    0xa0, 0xe0, 0x3b, 0x4d, 0xae, 0x2a, 0xf5, 0xb0, 0xc8, 0xeb, 0xbb, 0x3c, 0x83, 0x53,
0x99, 0x61,
    0x17, 0x2b, 0x04, 0x7e, 0xba, 0x77, 0xd6, 0x26, 0xe1, 0x69, 0x14, 0x63, 0x55, 0x21,
0x0c, 0x7d};

    unsigned char s_box[] = {
    0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67, 0x2b, 0xfe, 0xd7,
0xab, 0x76,
    0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2, 0xaf, 0x9c, 0xa4,
0x72, 0xc0,
    0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5, 0xf1, 0x71, 0xd8, 0x31,
0x15,
    0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12, 0x80, 0xe2, 0xeb, 0x27,
0xb2, 0x75,
    0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b, 0xd6, 0xb3, 0x29, 0xe3,
0x2f, 0x84,
    0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb, 0xbe, 0x39, 0x4a, 0x4c,
0x58, 0xcf,
    0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9, 0x02, 0x7f, 0x50, 0x3c, 0x9f,
0xa8,
    0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6, 0xda, 0x21, 0x10, 0xff, 0xf3,
0xd2,
    0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7, 0x7e, 0x3d, 0x64, 0x5d,
0x19, 0x73,
    0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8, 0x14, 0xde, 0x5e,
0x0b, 0xdb,
    0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3, 0xac, 0x62, 0x91, 0x95,
0xe4, 0x79,
    0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4, 0xea, 0x65, 0x7a,
0xae, 0x08,
    0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd, 0x74, 0x1f, 0x4b, 0xbd,
0x8b, 0x8a,
```

```
    0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35, 0x57, 0xb9, 0x86, 0xc1,
0x1d, 0x9e,
    0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e, 0x87, 0xe9, 0xce, 0x55,
0x28, 0xdf,
    0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d, 0x0f, 0xb0, 0x54,
0xbb, 0x16};

    unsigned long key_array[44];
    unsigned long rcon[] = {
       0x01000000, 0x02000000, 0x04000000, 0x08000000, 0x10000000,
       0x20000000,     0x40000000, 0x80000000, 0x1b000000, 0x36000000};


    /*Prototipos*/

    void init(unsigned char data_rx[16]);
    void show_state(void);

    /*funciones de creación de arreglo de claves (se ejecuta una sola vez)*/

    unsigned long get_master_key(unsigned char i);
    void key_expansion(unsigned long master_key[4]);
    unsigned long rot_word(unsigned long temp);
    unsigned long sub_word(unsigned long temp);

    /*funciones que son llamadas cada vez q se quiera ejecutar el algoritmo*/

    void copy_data_to_state(unsigned char data_rx[16]);

    void add_round_key(unsigned char round);
    void inv_mix_columns(void);
    unsigned char mult(unsigned char m, unsigned char n);
    void inv_sub_bytes(void);
    void inv_shift_rows(void);

    void copy_state_to_data(unsigned char data_rx[16]);


    /*Definiciones*/


    unsigned long get_master_key(unsigned char i){          /*Pide la Master Key*/
       unsigned long key;

       printf(" Ingrese la Master_key[%d] en hexadecimal(32 bits, 4 bytes): ",i);
       scanf("%x",&key);
       return key;
    }

    void key_expansion(unsigned long master_key[4]){
       const char Nk = 4, Nb = 4, Nr = 10;
       unsigned char i,j;
       unsigned long temp;

       for(i = 0;i < Nk;i++) key_array[i] = master_key[i];
```

```
    i = Nk;
    while (i < (Nb * (Nr + 1))){
            temp = key_array[i - 1];
            if ((i % Nk) == 0) {
                    (unsigned char) j = i/Nk;
                    temp = rot_word(temp);
                    temp = sub_word(temp) ^ rcon[j - 1];
            }
            else {
                    if ((Nk > 0x6) && ((i % Nk) == 0x4)) temp = sub_word(temp);
            }
            key_array[i] = key_array[i - Nk] ^ temp;
            i++;
    }
}

unsigned long rot_word(unsigned long temp){
    unsigned char i;

    for(i = 0;i < 8;i++){
            if ((temp & 0x80000000) != 0) temp = ((temp << 1) ^ 0x1);
            else temp = (temp << 1);
    }
    return temp;
}

unsigned long sub_word(unsigned long temp){
    unsigned char temp1, temp2, temp3, temp4;
    unsigned long temp5, temp6, temp7, temp8;

    temp1 = (unsigned char) temp;
    temp1 = s_box[temp1];

    temp2 = (unsigned char) (temp >> 8);
    temp2 = s_box[temp2];

    temp3 = (unsigned char) (temp >> 16);
    temp3 = s_box[temp3];

    temp4 = (unsigned char) (temp >> 24);
    temp4 = s_box[temp4];

    temp5 = (unsigned long) temp1;
    temp6 = (unsigned long) temp2;
    temp7 = (unsigned long) temp3;
    temp8 = (unsigned long) temp4;

    temp5 = temp5 + (temp6 << 8) + (temp7 << 16) + (temp8 << 24);
    return temp5;
}

void copy_data_to_state(unsigned char data_rx[16]) {
    unsigned char i,j;
    for(j = 0;j < 4;j++){
            for(i=0;i<4;i++) {
```

```
                        state[i][j] = data_rx[4*j + i];
                }
        }
}

void copy_state_to_data(unsigned char data_rx[16]) {
        unsigned char i,j;
        for(j=0;j<4;j++){
                for(i=0;i<4;i++) {
                        data_rx[4*j + i] = state[i][j];
                }
        }
}

void inv_shift_rows(void){
        unsigned char i,j,a[3];
        for(i = 1;i < 4;i++){
                j = 3;
                while (j > (3 - i)) {
                        a[3 - j] = state[i][7 - j - i];
                        j--;
                }
                j = 3;
                while (j > (i - 1)){
                        state[i][j] = state[i][j - i];
                        j--;
                }
                j = 3;
                while (j > (3 - i)){
                        state[i][3 - j] = a[3 - j];
                        j--;
                }
        }
}

void inv_sub_bytes(void){
        unsigned char i,j;

        for(i=0;i<4;i++){
                for(j=0;j<4;j++) {
                        state[i][j] = inv_s_box[state[i][j]];
                }
        }
}
void inv_mix_columns(void){
        unsigned char j, vect[4];

        for(j = 0;j < 4;j++){
        vect[0] = (mult(0xe,state[0][j])) ^ (mult(0xb,state[1][j])) ^ (mult(0xd,state[2][j])) ^
(mult(0x9,state[3][j]));
        vect[1] = (mult(0x9,state[0][j])) ^ (mult(0xe,state[1][j])) ^ (mult(0xb,state[2][j])) ^
(mult(0xd,state[3][j]));
        vect[2] = (mult(0xd,state[0][j])) ^ (mult(0x9,state[1][j])) ^ (mult(0xe,state[2][j])) ^
(mult(0xb,state[3][j]));
```

```
	vect[3] = (mult(0xb,state[0][j])) ^ (mult(0xd,state[1][j])) ^ (mult(0x9,state[2][j])) ^
(mult(0xe,state[3][j]));
			state[0][j] = vect[0];
			state[1][j] = vect[1];
			state[2][j] = vect[2];
			state[3][j] = vect[3];
		}
	}
	unsigned char mult(unsigned char m, unsigned char n){
		unsigned int s = 0;

		n = (unsigned int) n;
		if (m & 0x1) s = n;
		if (m & 0x2) s = (n << 1) ^ s;
		if (m & 0x4) s = (n << 2) ^ s;
		if (m & 0x8) s = (n << 3) ^ s;
		/*si luego de la mult supera 255 entonces hay aplicar módulo (x8 + x4 + x3 + x + 1)
(polinomio irreducible) */
		if (s > 1023) s = (s & 0x3ff) ^ (0x6c); /*0x1b << 2, descarte de b10 comenzado con b0
*/
		if (s > 511) s = (s & 0x1ff) ^ (0x36); /*0x1b << 1, descarte de b9 */
		if (s > 255) s = (s & 0xff) ^ (0x1b); /*descarte de b8*/
		s = (unsigned char) s;
		return s;
	}


	void add_round_key(unsigned char round){
		unsigned char i,j, key[4][4];
		unsigned long key_aux;

		for(i = 0;i < 4;i++){
			key_aux =  key_array[4 * round + i];
			key[3][i] = (unsigned char) key_aux;
			key[2][i] = (unsigned char) (key_aux >> 8);
			key[1][i] = (unsigned char) (key_aux >> 16);
			key[0][i] = (unsigned char) (key_aux >> 24);

		}

		for(i = 0;i < 4;i++){
			for(j = 0;j < 4;j++) state[i][j] = state[i][j] ^ key[i][j];
		}
	}

	void init(unsigned char data_rx[16]) {
		unsigned char i, ch;
		for(i=0;i<16;i++) {

			printf(" Ingrese el valor data_rx[%d]: ",i);
			scanf("%x", &ch);
			data_rx[i] = ch;

		}
	}
```

```
void show_state(void){ /*auxiliar para ver el funcionamiento*/
   unsigned char i,j;
   for(i=0;i<4;i++){
            printf("\n");
            for(j=0;j<4;j++) {
                     printf(" State[%d][%d] es: %x",i,j,state[i][j]);
            }
   }
}

void main(void){
   unsigned char data_rx[16];
   int c;
   unsigned long master_key[4];
   char k;

   printf("                              Proyecto Final\n\n");
   printf("Presentacion del Sistema de Encriptacion AES en lenguaje C \n\n");
   printf("                              Integrante\n\n");
   printf("                        Leandro Martin Del Gesso\n\n");
   printf("                              Tutores\n\n");
   printf("            Juan Carlos Bonadero, Monica Liberatori\n\n");

   printf("              Aqui se observara la desencriptacion\n\n");
   printf("A continuacion se le pedira ingresar los datos a encriptar\n");
   printf("(16 bytes) que conformaran el State\n\n");

   init(data_rx);
   copy_data_to_state(data_rx);

   printf("\n                                   State inicial");
   show_state();

   printf("\n\n    A continuacion se le pedira ingresar la clave master\n");
   printf("(128 bits) en 4 partes de 32 bits (4 bytes)\n\n");

   for(k = 0;k < 4;k++) master_key[k] = get_master_key(k); /*estos sera distinto en el
driver*/
   key_expansion(master_key);

   printf("\n\n Desea ver todas las operaciones sobre el state? (s/n, default: n)\n ");
   scanf("%s",&c);
   c = (unsigned char) c;

   /*Luego de inicializar comienza el algoritmo de desencriptación*/


   for(k = 10;k > 0;k--) {

            add_round_key(k);

            if (c == 's') {
printf("\n            State luego de add_round_key, ronda %d",k);
                     show_state();
```

```
            }

            if (k != 10) {
                        inv_mix_columns();

                        if (c == 's') {
printf("\n            State luego de inv_mix_columns, ronda %d",k);
                                    show_state();
                        }
            }

            inv_shift_rows();

            if (c == 's') {
printf("\n            State luego de Inv_mix_columns, ronda %d",k);
                        show_state();
            }

            inv_sub_bytes();

            if (c == 's') {
                        printf("\n                    State luego de inv_sub_bytes, ronda %d",k);
                        show_state();
            }

   }

   k = 0;

   add_round_key(k);

   if (c == 's') {
            printf("\n                    State luego de add_round_key, ronda %d",k);
            show_state();
   }

   copy_state_to_data(data_rx);

   printf("\n\n Los datos desencriptados son: ");
   for(k = 0;k < 16;k++) printf("%x ",data_rx[k]);
   printf("\n\n");
   getch();
}
```

aes.c

Código del tercer programa (aes.c):

```
#include <stdio.h>
#include <conio.h>

/*----------------------------------------Variables globales----------------------------------------*/

unsigned char state[4][4];
unsigned char inv_s_box[] = {
```

```
    0x52, 0x09, 0x6a, 0xd5, 0x30, 0x36, 0xa5, 0x38, 0xbf, 0x40, 0xa3, 0x9e, 0x81, 0xf3,
0xd7, 0xfb,
    0x7c, 0xe3, 0x39, 0x82, 0x9b, 0x2f, 0xff, 0x87, 0x34, 0x8e, 0x43, 0x44, 0xc4, 0xde, 0xe9,
0xcb,
    0x54, 0x7b, 0x94, 0x32, 0xa6, 0xc2, 0x23, 0x3d, 0xee, 0x4c, 0x95, 0x0b, 0x42, 0xfa,
0xc3, 0x4e,
    0x08, 0x2e, 0xa1, 0x66, 0x28, 0xd9, 0x24, 0xb2, 0x76, 0x5b, 0xa2, 0x49, 0x6d, 0x8b,
0xd1, 0x25,
    0x72, 0xf8, 0xf6, 0x64, 0x86, 0x68, 0x98, 0x16, 0xd4, 0xa4, 0x5c, 0xcc, 0x5d, 0x65,
0xb6, 0x92,
    0x6c, 0x70, 0x48, 0x50, 0xfd, 0xed, 0xb9, 0xda, 0x5e, 0x15, 0x46, 0x57, 0xa7, 0x8d,
0x9d, 0x84,
    0x90, 0xd8, 0xab, 0x00, 0x8c, 0xbc, 0xd3, 0x0a, 0xf7, 0xe4, 0x58, 0x05, 0xb8, 0xb3,
0x45, 0x06,
    0xd0, 0x2c, 0x1e, 0x8f, 0xca, 0x3f, 0x0f, 0x02, 0xc1, 0xaf, 0xbd, 0x03, 0x01, 0x13, 0x8a,
0x6b,
    0x3a, 0x91, 0x11, 0x41, 0x4f, 0x67, 0xdc, 0xea, 0x97, 0xf2, 0xcf, 0xce, 0xf0, 0xb4, 0xe6,
0x73,
    0x96, 0xac, 0x74, 0x22, 0xe7, 0xad, 0x35, 0x85, 0xe2, 0xf9, 0x37, 0xe8, 0x1c, 0x75,
0xdf, 0x6e,
    0x47, 0xf1, 0x1a, 0x71, 0x1d, 0x29, 0xc5, 0x89, 0x6f, 0xb7, 0x62, 0x0e, 0xaa, 0x18,
0xbe, 0x1b,
    0xfc, 0x56, 0x3e, 0x4b, 0xc6, 0xd2, 0x79, 0x20, 0x9a, 0xdb, 0xc0, 0xfe, 0x78, 0xcd,
0x5a, 0xf4,
    0x1f, 0xdd, 0xa8, 0x33, 0x88, 0x07, 0xc7, 0x31, 0xb1, 0x12, 0x10, 0x59, 0x27, 0x80,
0xec, 0x5f,
    0x60, 0x51, 0x7f, 0xa9, 0x19, 0xb5, 0x4a, 0x0d, 0x2d, 0xe5, 0x7a, 0x9f, 0x93, 0xc9,
0x9c, 0xef,
    0xa0, 0xe0, 0x3b, 0x4d, 0xae, 0x2a, 0xf5, 0xb0, 0xc8, 0xeb, 0xbb, 0x3c, 0x83, 0x53,
0x99, 0x61,
    0x17, 0x2b, 0x04, 0x7e, 0xba, 0x77, 0xd6, 0x26, 0xe1, 0x69, 0x14, 0x63, 0x55, 0x21,
0x0c, 0x7d};


    unsigned char s_box[] = {
    0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67, 0x2b, 0xfe, 0xd7,
0xab, 0x76,
    0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2, 0xaf, 0x9c, 0xa4,
0x72, 0xc0,
    0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5, 0xf1, 0x71, 0xd8, 0x31,
0x15,
    0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12, 0x80, 0xe2, 0xeb, 0x27,
0xb2, 0x75,
    0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b, 0xd6, 0xb3, 0x29, 0xe3,
0x2f, 0x84,
    0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb, 0xbe, 0x39, 0x4a, 0x4c,
0x58, 0xcf,
    0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9, 0x02, 0x7f, 0x50, 0x3c, 0x9f,
0xa8,
    0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6, 0xda, 0x21, 0x10, 0xff, 0xf3,
0xd2,
    0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7, 0x7e, 0x3d, 0x64, 0x5d,
0x19, 0x73,
    0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8, 0x14, 0xde, 0x5e,
0x0b, 0xdb,
```

```
    0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3, 0xac, 0x62, 0x91, 0x95,
0xe4, 0x79,
    0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4, 0xea, 0x65, 0x7a,
0xae, 0x08,
    0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd, 0x74, 0x1f, 0x4b, 0xbd,
0x8b, 0x8a,
    0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35, 0x57, 0xb9, 0x86, 0xc1,
0x1d, 0x9e,
    0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e, 0x87, 0xe9, 0xce, 0x55,
0x28, 0xdf,
    0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d, 0x0f, 0xb0, 0x54,
0xbb, 0x16};

    unsigned long key_array[44];

    unsigned long rcon[] = {
       0x01000000, 0x02000000, 0x04000000, 0x08000000, 0x10000000,
       0x20000000,     0x40000000, 0x80000000, 0x1b000000, 0x36000000};


    /*----------------------------------------Prototipos----------------------------------------------*/

    void init(unsigned char data_tx[16]);
    void show_state(void);

    /*funciones de creacion del arreglo de claves (ejecutadas una sola vez)*/

    unsigned long get_master_key(unsigned char i);
    void key_expansion(unsigned long master_key[4]);
    unsigned long sub_word(unsigned long temp);
    unsigned long rot_word(unsigned long temp);

    /*funciones esenciales para el algoritmo de encriptación, llamadas cada vez q se desea
encriptar datos*/

    void sub_bytes(void);
    void shift_rows(void);
    void mix_columns(void);
    unsigned char mult(unsigned char n, unsigned char m);
    void add_round_key(unsigned char round);

    /*funciones esenciales para el algoritmo de desencritación, llamadas cada vez q se
desea desencriptar datos*/

    void add_round_key(unsigned char round);
    void inv_mix_columns(void);
    unsigned char mult2(unsigned char m, unsigned char n);
    void inv_sub_bytes(void);
    void inv_shift_rows(void);

    /*funciones que usa tanto la encriptacion como la desencriptacion*/

    void copy_data_to_state(unsigned char data[16]);
    void copy_state_to_data(unsigned char data[16]);
    void show_state(void);
```

```
/*----------------------------------------Definiciones----------------------------------------*/


void copy_data_to_state(unsigned char data[16]) {
   unsigned char i,j;

   for(j=0;j<4;j++){
            for(i=0;i<4;i++) {
                     state[i][j] = data[4*j + i];
            }
   }
}

void copy_state_to_data(unsigned char data[16]) {
   unsigned char i,j;

   for(j=0;j<4;j++){
            for(i=0;i<4;i++) {
                     data[4*j + i] = state[i][j];
            }
   }
}

unsigned long get_master_key(unsigned char i){
   unsigned long key;

   printf(" Ingrese la Master_key[%d] en hexadecimal(32 bits, 4 bytes): ",i);
   scanf("%x",&key);
   return key;
}

void key_expansion(unsigned long master_key[4]){
   const char Nk = 4, Nb = 4, Nr = 10;
   unsigned char i,j;
   unsigned long temp;

   for(i = 0;i < Nk;i++) key_array[i] = master_key[i];

   i = Nk;

   while (i < (Nb * (Nr + 1))){
            temp = key_array[i - 1];
            if ((i % Nk) == 0) {
                     j = (unsigned char) i/Nk;

                     temp = rot_word(temp);

                     temp = sub_word(temp) ^ rcon[j - 1];

            }
            else {
                     if ((Nk > 0x6) && ((i % Nk) == 0x4)) temp = sub_word(temp);
            }
```

```
            key_array[i] = key_array[i - Nk] ^ temp;
            i++;
    }
}

unsigned long rot_word(unsigned long temp){
    unsigned char i;

    for(i = 0;i < 8;i++){
            if ((temp & 0x80000000) != 0) temp = ((temp << 1) ^ 0x1);
            else temp = (temp << 1);
    }
    return temp;
}

unsigned long sub_word(unsigned long temp){
    unsigned char temp1, temp2, temp3, temp4;
    unsigned long temp5, temp6, temp7, temp8;

    temp1 = (unsigned char) temp;
    temp1 = s_box[temp1];

    temp2 = (unsigned char) (temp >> 8);
    temp2 = s_box[temp2];

    temp3 = (unsigned char) (temp >> 16);
    temp3 = s_box[temp3];

    temp4 = (unsigned char) (temp >> 24);
    temp4 = s_box[temp4];

    temp5 = (unsigned long) temp1;
    temp6 = (unsigned long) temp2;
    temp7 = (unsigned long) temp3;
    temp8 = (unsigned long) temp4;

    temp5 = temp5 + (temp6 << 8) + (temp7 << 16) + (temp8 << 24);

    return temp5;
}

void sub_bytes(void){
    unsigned char i,j;
    const unsigned char c = 0x63;

    for(i=0;i<4;i++){

            for(j=0;j<4;j++) {
                    state[i][j] = s_box[state[i][j]];
            }
    }
}

void shift_rows(void){
    unsigned char i,j,a[3];
```

```
    for(i = 1;i < 4;i++){
        j = 0;
        while (j < i) {
            a[j] = state[i][j];
            j++;
        }
        j = 0;
        while (j < (4 - i)){
            state[i][j] = state[i][j+i];
            j++;
        }
        j = 0;
        while (j < i){
            state[i][3 - j] = a[i - 1 - j];
            j++;
        }
    }
}

void mix_columns(void){
    unsigned char j, vect[4];

    for(j = 0;j < 4;j++){
vect[0] = (mult(0x2,state[0][j])) ^ (mult(0x3,state[1][j])) ^ state[2][j] ^ state[3][j];
vect[1] = state[0][j] ^ (mult(0x2,state[1][j])) ^ (mult(0x3,state[2][j])) ^ state[3][j];
vect[2] = state[0][j] ^ state[1][j] ^ (mult(0x2,state[2][j])) ^ (mult(0x3,state[3][j]));
vect[3] = (mult(0x3,state[0][j])) ^ state[1][j] ^ state[2][j] ^ (mult(0x2,state[3][j]));
        state[0][j] = vect[0];
        state[1][j] = vect[1];
        state[2][j] = vect[2];
        state[3][j] = vect[3];
    }
}

unsigned char mult(unsigned char m, unsigned char n){
    unsigned int s;

    n = (unsigned int) n;
    if (m == 0x2){ /*multiplicación x 2*/
        s = n << 1;
        /*si luego de la mult x 2 supera 255 entonces hay aplicar módulo x8 + x4 +
x3 + x + 1*/
        if (s > 255) s = (s & 0xff) ^ (0x1b);
    }
    else{
        s = (n << 1) ^ n; /*multiplicación x 3*/
        /*si luego de la mult x 3 supera 255 entonces hay que aplicar módulo x8 + x4
+ x3 + x + 1*/
        if (s > 255) s = (s & 0xff) ^ (0x1b); /*descarte de b8 comenzando desde b0*/
    }
    s = (unsigned char) s;
    return s;
}
```

```
void add_round_key(unsigned char round){
   unsigned char i,j, key[4][4];

   for(i = 0;i < 4;i++){
            key[3][i] = (unsigned char) key_array[(4 * round) + i];
            key[2][i] = (unsigned char) (key_array[(4 * round) + i] >> 8);
            key[1][i] = (unsigned char) (key_array[(4 * round) + i] >> 16);
            key[0][i] = (unsigned char) (key_array[(4 * round) + i] >> 24);
   }
   for(i = 0;i < 4;i++){
            for(j = 0;j < 4;j++) state[i][j] = state[i][j] ^ key[i][j];
   }
}

void inv_shift_rows(void){
   unsigned char i,j,a[3];
   for(i = 1;i < 4;i++){
            j = 3;
            while (j > (3 - i)) {
                     a[3 - j] = state[i][7 - j - i];
                     j--;
            }
            j = 3;
            while (j > (i - 1)){
                     state[i][j] = state[i][j - i];
                     j--;
            }
            j = 3;
            while (j > (3 - i)){
                     state[i][3 - j] = a[3 - j];
                     j--;
            }
   }
}

void inv_sub_bytes(void){
   unsigned char i,j;

   for(i=0;i<4;i++){
            for(j=0;j<4;j++) {
                     state[i][j] = inv_s_box[state[i][j]];
            }
   }
}

void inv_mix_columns(void){
   unsigned char j, vect[4];

   for(j = 0;j < 4;j++){
            vect[0]  =  (mult2(0xe,state[0][j]))  ^  (mult2(0xb,state[1][j]))  ^
(mult2(0xd,state[2][j])) ^ (mult2(0x9,state[3][j]));
            vect[1]  =  (mult2(0x9,state[0][j]))  ^  (mult2(0xe,state[1][j]))  ^
(mult2(0xb,state[2][j])) ^ (mult2(0xd,state[3][j]));
            vect[2]  =  (mult2(0xd,state[0][j]))  ^  (mult2(0x9,state[1][j]))  ^
(mult2(0xe,state[2][j])) ^ (mult2(0xb,state[3][j]));
```

```
                vect[3] = (mult2(0xb,state[0][j])) ^ (mult2(0xd,state[1][j])) ^
(mult2(0x9,state[2][j])) ^ (mult2(0xe,state[3][j]));
                state[0][j] = vect[0];
                state[1][j] = vect[1];
                state[2][j] = vect[2];
                state[3][j] = vect[3];
        }
    }

    unsigned char mult2(unsigned char m, unsigned char n){
        unsigned int s = 0;

        n = (unsigned int) n;
        if (m & 0x1) s = n;
        if (m & 0x2) s = (n << 1) ^ s;
        if (m & 0x4) s = (n << 2) ^ s;
        if (m & 0x8) s = (n << 3) ^ s;
        /*si luego de la mult supera 255 entonces hay aplicar módulo (x8 + x4 + x3 + x + 1)
(polinomio irreducible) */
        if (s > 1023) s = (s & 0x3ff) ^ (0x6c); /*0x1b << 2, descarte de b10 comenzado con b0
*/
        if (s > 511) s = (s & 0x1ff) ^ (0x36); /*0x1b << 1, descarte de b9 */
        if (s > 255) s = (s & 0xff) ^ (0x1b); /*descarte de b8*/
        s = (unsigned char) s;
        return s;
    }

    void init(unsigned char data_tx[16]) {
        unsigned char i, ch;

        for(i=0;i<16;i++) {

                printf(" Ingrese data_rx[%d] en hexadecimal (1 byte): ",i);

                scanf("%x", &ch);

                data_tx[i] = ch;

        }
    }

    void show_state(void){
        unsigned char i,j;

        for(i=0;i<4;i++){
                printf("\n");
                for(j=0;j<4;j++) {
                        printf(" State[%d][%d] es: %x",i,j,state[i][j]);
                }
        }
    }


    void main(void) {
```

```
	unsigned char data_tx[16], k, data_rx[16];
	int c;
	unsigned long master_key[4];

	printf("                              Proyecto Final\n\n");
	printf("Presentacion del Sistema de Encriptacion AES en lenguaje C \n\n");
	printf("                              Integrante\n\n");
	printf("                        Leandro Martin Del Gesso\n\n");
	printf("                              Tutores\n\n");
	printf("            Juan Carlos Bonadero, Monica Liberatori\n\n");

	printf("A continuacion se le pedira ingresar los datos a encriptar\n");
	printf("(16 bytes) que conformaran el State\n\n");
	init(data_tx);
	copy_data_to_state(data_tx);

	printf("\n                                    State inicial");
	show_state();

    /*key_expacnsion es una misma operación tanto para la encriptacion como para la
desencriptacion*/


	printf("\n\n    A continuacion se le pedira ingresar la clave master\n");
	printf("(128 bits) en 4 partes de 32 bits (4 bytes)\n\n");

	for(k = 0;k < 4;k++) master_key[k] = get_master_key(k); /*en el driver esto se hara
de otra forma*/
	key_expansion(master_key);

	printf("\n\n Desea ver todas las operaciones sobre el state? (s/n, default: n)\n ");
	scanf("%s",&c);
	c = (unsigned char) c;

	k = 0;


	/*Comienza la encriptacion*/


	add_round_key(k);

	if (c == 's') {
		printf("\n                State luego de add_round_key ronda %d",k);
		show_state();
	}

	for(k = 1;k < 11;k++){

		sub_bytes();

		if (c == 's') {
			printf("\n                State luego de Sub_bytes ronda %d",k);
			show_state();
		}
```

```
                shift_rows();

                if (c == 's') {
                        printf("\n               State luego de shift_rows ronda %d",k);
                        show_state();

                }

                if (k != 10) {
                        mix_columns();

                        if (c == 's') {
                                printf("\n               State luego de mix columns ronda %d",k);
                                show_state();
                        }

                }

                add_round_key(k);

                if (c == 's') {
                        printf("\n               State luego de add_round_key ronda %d",k);

                        show_state();

                }

        }

        copy_state_to_data(data_tx);

        printf("\n\n Los datos encriptados son: ");
        for(k = 0;k < 16;k++) printf("%x ",data_tx[k]);
        printf("\n\n");


    /*-------------------------------------------------------------------------
    *
    *  acá temrmina la encriptación y comienza la desencriptacion
    *
    *-------------------------------------------------------------------------*/


    if (c == 's') printf("\n               Fin de la encriptacion, comienza la desencriptacion\n\n");

        for(k = 0;k <16;k++) data_rx[k] = data_tx[k];

        copy_data_to_state(data_rx);

        /*comienza la desencriptacion*/

        if (c == 's') {
```

```
            printf("\n                                    State inicial");
            show_state();
            printf("\n");
    }

    for(k = 10;k > 0;k--) {

            add_round_key(k);

            if (c == 's') {
                    printf("\n                State luego de add_round_key, ronda %d",k);
                    show_state();
            }

            if (k != 10) {
                    inv_mix_columns();

                    if (c == 's') {
printf("\n                State luego de inv_mix_columns, ronda %d",k);
                            show_state();
                    }
            }

            inv_shift_rows();

            if (c == 's') {
printf("\n                State luego de Inv_mix_columns, ronda %d",k);
                    show_state();
            }

            inv_sub_bytes();

            if (c == 's') {
                    printf("\n                State luego de inv_sub_bytes, ronda %d",k);
                    show_state();
            }

    }

    k = 0;

    add_round_key(k);

    if (c == 's') {
            printf("\n                State luego de add_round_key, ronda %d",k);
            show_state();
    }

    copy_state_to_data(data_rx);

    /*en el driver hay q copiar data_rx a una variable externa*/

    printf("\n\n Los datos desencriptados son: ");
    for(k = 0;k < 16;k++) printf("%x ",data_rx[k]);
    printf("\n\n");
```

```
    getch();
}
```

Segunda etapa:

Obtención del driver

En principio, se necesita contar con el driver de la placa DWL-G520+ que se puede bajar de la página www.acx100.sourceforge.net para la distribución Slackware y versión de kernel 2.4.x. Dado que hay diversos drivers hechos en fechas diferentes, se testearon varios y se llegó a la conclusión de que el **acx100-0.2.0pre8_plus_fixes_48** es uno de los que mejor funciona. Todo lo realizado a continuación fue hecho sobre ese driver.

Se comenzó examinando el driver para poder comprender su estructura; la forma en que transmite tramas hacia el medio, que arriban desde el sistema operativo; y la manera de recepción de tramas desde el medio y su administración para pasarlas al sistema operativo.

El código del driver se encuentra organizado en dos directorios en los cuales se encuentra todo el código: uno es "src" y aloja el código fuente propiamente dicho (archivos *.c) y el otro es "include" y posee los encabezados usados por los archivos en "scr" (archivos *.h).

El en el proyecto se incorporaron varias funciones que se agregaron en archivos cabeceras en el directorio "include" y a los archivos fuentes les fueron incorporadas líneas de código para poder usar las funciones anteriores. Todo ésto se detalla seguidamente a la explicación del funcionamiento.

Una vez bajado el driver, hay que descomprimirlo, compilarlo y luego puede ser cargado mediante un script que se encuentra en el directorio "scripts" y se denomina "start_net"; del mismo modo existe otro script en el mismo lugar llamado "stop_net" que baja el módulo.

Debug proporcionado por el driver

Una característica de mucha utilidad en el driver es que el "Start_net" se puede editar, y así se configuran los parámetros de red tales como dirección IP, GateWay, subred mask, etc. Adicionalmente, posee una variable llamada "DEBUG" que se utiliza proporcionar diversos mensajes del driver y así observar su funcionamiento si se lo desea.

"DEBUG" debe ser igualada a alguno de los siguientes valores:

- ✓ 0x00 para debug nulo.
- ✓ 0xb para debug medio.
- ✓ 0xffff para máximo debug.

Las dos últimas posibilidades le comunican al driver que provea diversos mensajes los cuales son almacenados en /var/log/syslog. Con la opción 0xffff imprime hasta las tramas recibidas y transmitidas. Ésto ha sido de gran ayuda a lo largo de todo el Proyecto puesto que actúa como un sniffer.

Configuración del Punto de Acceso

El punto de acceso posibilita ser configurada de dos maneras: una llamada "WPA/EAP" (también conocida como WPA enterprice) que necesita un servidor radius sobre la red cableada y se utiliza para redes de gran envergadura; la otra es "WPA/PSK" (Pre Shared Key) o "WPA/ Personal" y es utilizada en redes de menor desarrollo que la anterior.

Si se observan las diferencias entre una y otra, se puede ver que el servidor radius proporciona EAP (extended autentication method) que implica la utilización de upper-layer authentication (autenticación de capas superiores), a manera de ejemplo se puede citar TLS (Transport layer Security). Luego de finalizado el proceso EAP es establecida la PMK (Pairwise Master Key, 256 bits).

Consideraciones con WPA/PSK - AES

En WPA/PSK la PMK se obtiene introduciendo una PassPhrase que consiste en una frase de 8 a 63 caracteres la cual es expandida a 256 bits (PMK), ésto se debe a que sería complicado recordar la PMK directamente por su longitud de 256 bits.

Las dos opciones utilizan un protocolo llamado 4-way-handshake como autenticación y sirve para probar el conocimiento de la PMK entre el punto de acceso y el adaptador mutuamente. La diferencia es que al final del EAP se inicia el 4-way-handshake y en WPA/PSK el 4-way-handshake es iniciado por la asociación (no existe ningún protocolo de capa superior a la de enlace entre la asociación y el 4-way-handshake).

Ambas opciones a su vez permiten seleccionar la forma de encriptado (TKIP o AES).

Dado que la opción WPA/EAP involucra gran desarrollo para implementarlo en un proyecto final de una sola persona, se decidió que WPA/PSK - AES sería el objetivo del proyecto.

Básicamente, hay que tener en cuenta 3 cuestiones para que el adaptador funcione con WPA/PSK - AES:

1. Se deben cumplir los requisitos de la asociación.
2. Se debe implementar el 4-way-handshake (autenticación).
3. Hay que introducir el sistema de encriptación con ciertas características adicionales a lo visto en la primera etapa.

La secuencia que se debe seguir en WPA hasta llegar al 4-way-handshake es:

a) Autenticación a sistema abierto.

b) Asociación con el elemento de información WPA (WPA IE).

Todo comienza con la autenticación a sistema abierto y consiste simplemente en dos mensajes, uno desde el adaptador inalámbrico y otro desde el punto de acceso.

Asociación:

Elemento de información WPA

Al igual que en la autenticación a sistema abierto, la asociación consiste en el intercambio de dos mensajes, uno desde el adaptador inalámbrico y otro desde el punto de acceso. Este intercambio se detalla en estándar 802.11, pero en la asociación WPA además de los elementos de información adicionados en una asociación normal, se debe agregar el elemento WPA (WPA-IE) que se detalla a continuación. Para más detalles sobre las tramas de asociación y autenticación consultar el ítem 1 de la bibliografía, capítulo 4.

El WPA-IE posee la siguiente estructura:

```
Unsigned char  CipherWpaPskAes[] = {
        0xDD, 0x16,                     // RSN IE
        0x00, 0x50, 0xf2, 0x01,    // oui
        0x01, 0x00,                     // Version
        0x00, 0x50, 0xf2, 0x04,    // Multicast
        0x01, 0x00,                     // Number of unicast
        0x00, 0x50, 0xf2, 0x04,    // unicast
        0x01, 0x00,                     // number of authentication method
        0x00, 0x50, 0xf2, 0x02     // authentication
        };
Unsigned char CipherWpaPskTkip[] = {
        0xDD, 0x16,                     // RSN IE
        0x00, 0x50, 0xf2, 0x01,    // oui
        0x01, 0x00,                     // Version
        0x00, 0x50, 0xf2, 0x02,    // Multicast
        0x01, 0x00,                     // Number of unicast
```

```
0x00, 0x50, 0xf2, 0x02,    // unicast
0x01, 0x00,                // number of authentication method
0x00, 0x50, 0xf2, 0x02     // authentication
};
```

Esto es lo que el punto de acceso envía en sus beacons y probe-response como elemento de información adicional a los usuales como por ejemplo: supported rates, ssid, etc. Al ponerlo en el Association Request se le informa al punto de acceso que el adaptador soporta wpa y está configurado de la misma forma que ella, a lo que el punto de acceso devolverá un Association Response exitoso.

En el caso del Proyecto final, el primer elemento de información mostrado fue el usado dado que es el corresponde a AES.

Para llevar a cabo ésto, se implementó una variable global llamada WPA_IE incluida en el archivo cabecera *"globals.h"*.

Cada una de las variables globales utilizadas se irán desarrollando oportunamente.

Los códigos fuente de cada archivo mencionado en el desarrollo del funcionamiento del programa se detallan al final de ésta sección

En el archivo *acx100_helper2.c* se encuentra la función que transmite la association request, que debió ser modificada para que incluyera WPA_IE en sus elementos de información.

Se debe copiar el elemento completo en la posición de memoria en que están siendo almacenados los elementos de información para finalmente ser enviados al medio junto con toda la trama.

Esto se logra llamando a la función *memcpy* con las direcciones de destino e inicio y la cantidad de bytes a ser copiados. El inicio es la dirección

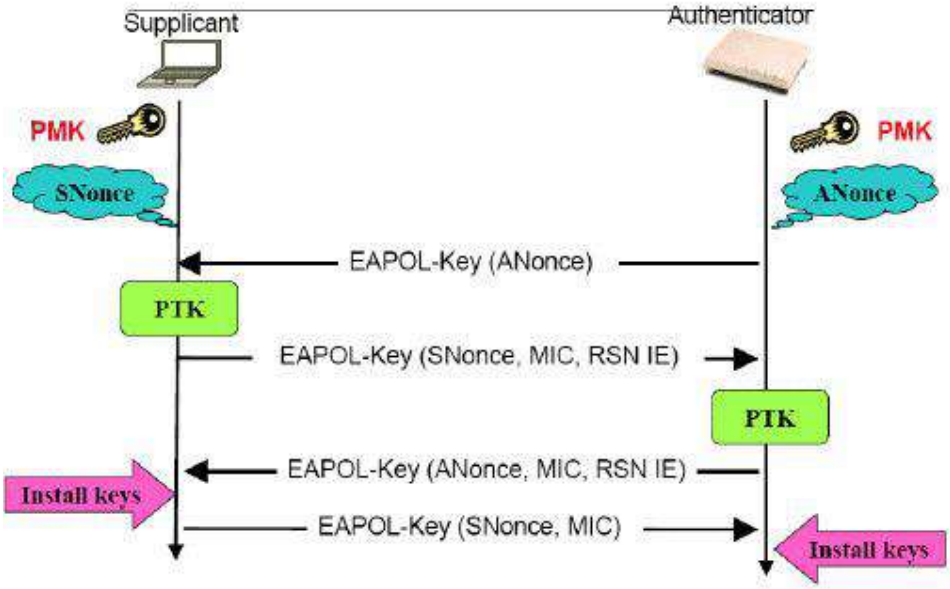
de memoria del WPA-IE, el destino es la siguiente dirección de memoria del último valor del último elemento de información copiado.

Autenticación:

Luego de la asociación, viene el 4-way-handshake el cual es transmitido en tramas de tipo dato (en el campo type (dos bits) del encabezado 802.11 se coloca un 10).

4-way-handshake

La siguiente figura ilustra el protocolo:



Para los 4 mensajes se utilizan *mensajes EAPOL-key* más un encabezado *EAPOL* (EAP over lan) que tiene 4 bytes y que son: la versión (0x01), el tipo (0x03), la longitud del cuerpo (body_lenght) que ocupa dos bytes y que contiene el total en bytes del EAPOL-key frame a partir del campo body_length excluido éste.

EAPOL key frame

La trama EAPOL-key tiene la siguiente estructura:

Descriptor Type – 1 octet	
Key Information – 2 octets	Key Length – 2 octets
Key Replay Counter – 8 octets	
Key Nonce – 32 octets	
EAPOL-Key IV – 16 octets	
Key RSC – 8 octets	
Reserved - 8 octets	
Key MIC – 16 octets	
Key Data Length – 2 octets	Key Data – n octets

EAPOL-Key frame

Un campo clave en este mensaje es el *Key Information* que informa lo que lleva el mensaje, es decir, es usado como campo de instrucción para que el otro extremo sepa que es lo que se envió. *Key Information* se detalla a continuación:

B0 B1 B2	B3	B4 B5	B6	B7	B8	B9	B10	B11	B12 B13 B14 B15
Key Descriptor Version	Key Type	Key Index	Install	Ack Bit	MIC Bit	Secure	Error	Request	Reserved

Key Information

Para mayor detalle consultar capítulo 10 del ítem [2].

A continuación se detallan las funciones necesarias para la ejecución del 4-way-handshake.

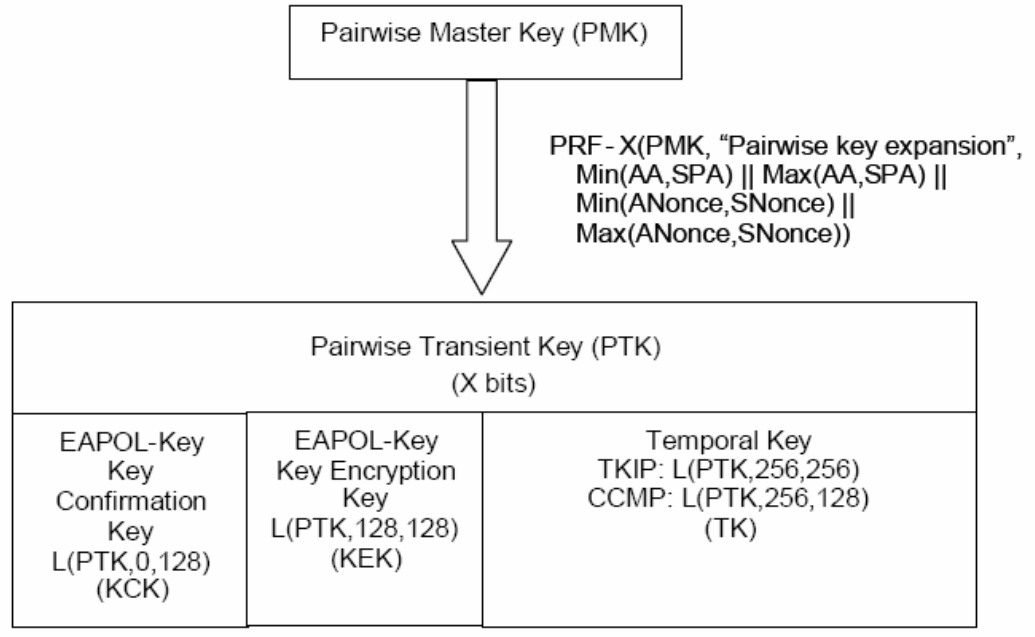
En el primer mensaje del 4-way-handshake, el punto de acceso envía un nonce llamado anonce, que consiste en una secuencia pseudoaleatoria para poder comprobar el conocimiento de la Pairwise Master Key (PMK) de 256

bits en ambos extremos del link. En el segundo mensaje, el adaptador envía su nonce (snonce)

Una vez recibido el primer mensaje, el adaptador computa la clave PTK (Pairwise Temporal Key) a partir de la PMK. En el caso que se use AES en la encriptación, la PTK es de 48 bytes y en caso de que TKIP sea usado, La PTK será 64 bytes.

Expansión de claves

Se puede observar gráficamente en la siguiente figura el desarrollo de las claves temporales a partir de la PMK. Para mayor detalle sobre la jerarquía de las claves consultar el capitulo 10 de Real 802.11 Security (item 2 de la bibliografía).



Pairwise key hierarchy

La clave *KCK* se usa para calcular el mic (Message Integrity Control) sobre todo el mensaje EAPOL (desde *EAPOL version* inclusive hasta el final de la trama).

La TK es la clave para la encriptación.

La KEK sirve para encriptar menajes EAPOL pero no es usada.

Password Hash

En WPA/PSK la PMK es configurada desde una *"passphrase"* de 8 a 63 caracteres ASCII que luego son expandidos a 256 bits para conformar la PMK. La expansión desde la *passphrase* a la PMK se lleva a cabo mediante una función llamada *PasswordHash*. Esta función está sugerida en el documento *ieee802.11i.pdf* que puede ser obtenido desde www.ieee.org. En el driver se implementó en el archivo *Passwordhash.h*.

Para hacer la expansión de claves se necesitan, además de los anonce y snonce, las direcciones MAC de la AP y del adaptador. El anonce es enviado en el primer mensaje del 4-way-handshake y el snonce lo construye el adaptador, de esta forma el adaptador inalámbrico computa las claves temporales antes que el punto de acceso. Luego el adaptador envía el segundo mensaje con su snonce para que el punto de acceso compute las claves y verifique que el adaptador también conoce la PMK a través del mic que el adaptador incluye en el mensaje.

Tanto para el desarrollo de la PTK como para crear el snonce en el adaptador, se utiliza una función generadora de secuencias pseudoaleatorias conocida como PRF (pseudorandom function) a la cual se le deben ingresar parámetros y devolverá como resultado de su operación la cantidad de bytes que se le indique. La función PRF se definió en un archivo cabecera llamado *PRF.h*.

Hashing

Tanto *PRF* como *PasswordHash* utilizan una operación de hashing. El Hashing es bastante usado en criptografía para combinar dos o más entradas y producir un resultado impredecible; basta con modificar un solo bit de la entrada para que se modifique completamente la salida impidiendo obtener conclusiones a partir de ésta. La única forma de que el algoritmo dé la misma salida es ejecutando la función con exactamente los mismos parámetros de entrada.

Los algoritmos de hashing usados en WPA se conocen como hmac_md5 y hmac_sha1. Éstos dos métodos están estandarizados en las regla RFC 2104. Además también se encuentran en *ieee802.11i.pdf*.

Las funciones hmac_sha1 y hmac_md5 fueron implementadas en dos archivos cabeceras *hmac_sha1.h y hmac_md5.h* respectivamente.

hmac_md5 arroja una salida de 16 bytes y usa el algoritmo de hash llamado md5.

hmac_sha1 arroja una salida de 20 bytes y usa el algoritmo de hash conocido como sha1.

En cuanto a los argumentos pasados a estas funciones son los siguientes: la dirección de la *clave*, longitud de clave, la dirección de *datos*, longitud de *datos* y la dirección de la *salida*. Los datos dependen de la aplicación de las funciones, que se detallarán más adelante.

Md5 y sha1 fueron implementadas en los encabezados *"md5.h"* y "*sha1.h"* respectivamente.

Una vez presentadas las funciones auxiliares usadas en la autenticación, se procede a explicar cómo y cuando son usadas.

Para llevar a cabo el 4-way-handshake se agregó un archivo cabecera llamado *4-way.h* en el archivo *acx100_helper2.h,* el cual incluye los siete archivos descriptos anteriormente.

Dentro de *4-way.h* hay cinco funciones que son la esencia de la autenticación. Una función es *acx_4_way* la cual es llamada cuando llega un mensaje del 4-way-handshake. Ésta primero debe obtener un descriptor de la estructura de transmisión para poder transmitir, luego examina si el mensaje recibido es el primero o el tercero del 4-way-handshake. Después de ésta decisión, llama a dos funciones: *four_way_2* o *four_way_4* las cuales contestan a los respectivos mensajes recibidos.

La cuarta función: *auxiliar* es llamada durante la autenticación a sistema abierto para acelerar el proceso de 4-way-handshake.

Finalmente, la quinta función es *init_WPA_frame* y se explica más adelante.

Función auxiliar

Según el orden de llamada, la función *auxiliar* es la primera.

El motivo de existencia de *auxiliar* es que hay operaciones que se pueden realizar antes de llegar a la etapa del 4-way-handshake y de esta forma acelerar el proceso.

Estas operaciones son la *PasswordHash,* la construcción del snonce y la inicialización de la estructura *WPA_frame* que está definida en *globals.h* y aloja la estructura del *EAPOL key message*. Todo esto toma un tiempo mayor a **100 ms** lo que justifica que se lleve a cabo antes de recibir el primer mensaje del 4-way-handshake para no demorar la repuesta a los mensajes del punto de acceso.

Para la construcción del snonce se usa la función PRF de la siguiente forma:

PRF(Random number, 32, "Init Counter", 12, MAC || Time, 10, snonce) (pag. 225 [2])

Tanto para la construcción de *Random Number* como para el conocimiento de *Time* se utiliza una llamada al sistema: *rdtscl(d).* Ésta actúa sobre el argumento *d* que debe ser *unsigned long* (u32) y le asigna a la misma el tiempo desde que la computadora comenzó a funcionar con una exactitud de 0,001 us.

Para hacer el *Random Number* se llama a PRF con el parámetro *time* (u32). La salida, *Random Number,* posee una longitud de 256 bits y debe ser el mejor número aleatorio que la computadora puede hacer.

Init Counter es literalmente un *string*.

El quinto parámetro es la concatenación de la dirección MAC del adaptador y el tiempo.

La función *init_WPA_frame* pone todos los campos en cero menos los dos primeros bytes del encabezado EAPOL, el campo *descriptor_type* y el *replay_counter*.

Como ya se dijo, *auxiliar* es llamada durante la autenticación a sistema abierto para que cuando llegue el primer mensaje del 4-way-handshake se esté listo para responder con el segundo mensaje teniendo en cuenta que antes de enviar el segundo mensaje habrá que hacer algunas operaciones que tomarán su tiempo.

Derivación de mensajes EAPOL - Key

Para que el driver decida llamar a la función *acx_4_way* cuando arriban mensajes pertenecientes al 4-way-handshake, hay que modificar el archivo *acx100_helper2.c*.

El adaptador al recibir una trama tanto para su dirección MAC como para un broadcast crea una interrupción lo que da lugar a la ejecución de una función que examina que tipo de trama ha llegado (acx_rx_ieee802_11*_frame) y* la deriva a la respectiva función que procesa su contenido si corresponde. De esta forma, en la línea de código que examina

si la trama es "DATA_ONLY" (del campo type del header) se introdujo el código necesario para que reconozca un mensaje EAPOL mediante la observación del campo type, que sigue al encabezado llc, y debe contener el número 0x888e.

Después de la derivación de los mensajes EAPOL a la función *acx_4_way*, se procede a la explicación de las funciones *acx_4_way, four_way_2* y *four_way_4*.

El punto de acceso envía el primer mensaje, con su anonce y el campo *Replay_counter* en uno, y aguarda por su contestación, si no la recibe o recibe una contestación errónea, simplemente repite el primer mensaje luego de transcurrido un segundo desde el primero con la diferencia de que el campo *Replay_Counter* es incrementado en uno (es decir que envía el mismo anonce). Si al cabo de cuatro mensajes desde el punto de acceso no consiguió recibir el segundo mensaje válido, envía un mensaje de *Deauthentication* a la placa inalámbrica. Luego el adaptador deberá comenzar de nuevo desde la autenticación a sistema abierto.

En la estructura *WPA_frame, replay_counter* es inicializado en 0x0000000000000001 y es incrementado con cada mensaje del 4-way-handshake. Lo primero que hace la función *acx_4_way* es fijarse si el *replay_counter* de la estructura interna ha llegado a 0x0000000000000005 lo que significa que el punto de acceso envió cuatro veces el primer mensaje y no pudo entender la respuesta enviada por el adaptador con lo que reinicia el replay_counter al valor inicial.

El programa se optimizó para enviar lo más rápido posible las respuestas al punto de acceso, y en caso de que ésta repita los mensajes, dado que ya se conocen las tramas de los mensajes posteriores al primero antes de ser enviados, se computa la respuesta a los subsiguientes mensajes antes que arriben para responder aún más rápido.

Luego *acx_4_way* compara el *raplay_counter* recibido con el que posee la estructura interna del adaptador y las banderas que deben aparecer en el primer o tercer mensaje ubicadas en el campo *key_information* de los mensajes recibidos. A partir de esta comparación decide si el mensaje recibido es el primero o el tercero y deriva a las correspondientes *four_way_2* o *four_way_4* para procesar los mensajes.

Las variables necesarias para que *acx_4_way* transmita son las primeras definidas en ésta. Todas las funciones y estructuras para la transmisión fueron extraídas del mismo driver, es decir, que no se centrará la explicación en como transmite y recibe el driver sino en que se transmite y que se recibe, que es la parte central del Proyecto.

Luego la función observa el *replay_countar* para ver si está en el valor inicial y así realizar la expansión de claves AES después de transmitir la trama .

Expansión de claves AES en el driver

En cuanto a la expansión de claves para la encriptación, se creó el archivo *key.h* que está incluido en *4-way.h* y lleva a cabo la expansión de claves AES tal como se vio en la primer parte del Proyecto a partir de la clave DATA_enc_key que es una variable global y se encuentra definida en *globals.h*. DATA_enc_key son los primeros 16 bytes de la PTK.

Finalmente *acx_4_way* calcula el mic para la siguiente trama en caso de que el punto de acceso no entienda por alguna razón el mensaje transmitido.

Con respecto a la función *four_way_2*, primero examina si el replay_counter está en uno, lo que significa que se ha recibido el primer mensaje con lo cual copia a la variable que almacena los datos a transmitir, los datos correspondientes al segundo mensaje. Por último calcula el mic sólo para el primer mensaje dado que en caso de repetición de mensajes por

parte del punto de acceso, el mic se calcula al final de *acx_4_way* como ya se mencionó antes.

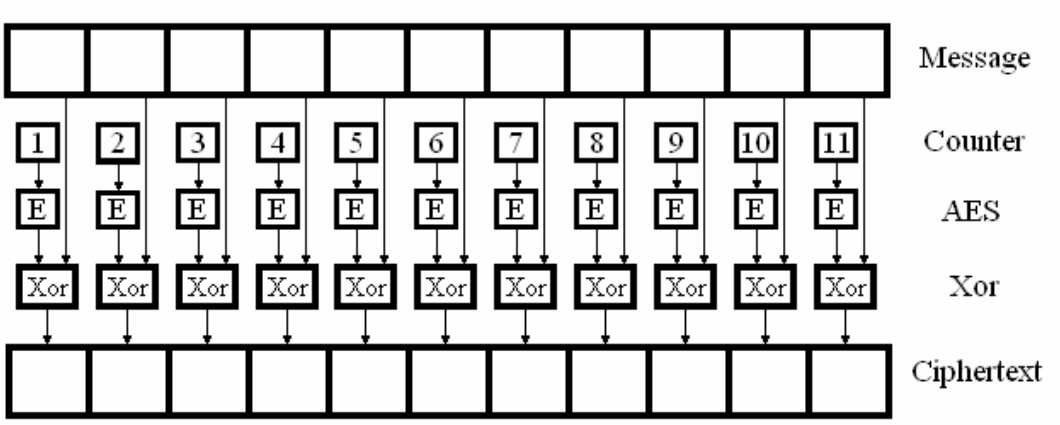
Al final de *four_way_2* se copian los datos a transmitir a la variable global *frame* definida en *globals.h* con el campo *key_mic* en cero y el replay_counter incrementado en uno para poder calcular después el mic del próximo mensaje.

La función *four_way_4* no es muy diferente a *four_way_2,* básicamente hace lo mismo pero alojando en *frame* los campos correspondientes al cuarto mensaje.

Encriptación

CCMP/AES

El corazón de la encriptación consiste en el método *Counter Mode + CBCMAC + Advanced Encryption Standard* (CCMP/AES). Para hacer una breve explicación del funcionamiento del método se recurrirá a la siguiente figura:



Partiendo de un mensaje que arriba a la capa de enlace para ser transmitido al medio, es dividido en múltiples bloques de 16 bytes. A cada bloque se le aplica la operación or exclusiva con el resultado de un contador incrementado para cada bloque y encriptado con AES.

El método permite que la longitud del mensaje no deba ser exactamente un múltiplo de 16 dado que al final se tiene el resultado de la encriptación del contador or exclusivo con la última parte del mensaje (que tiene menos de 16 bytes), con lo cual se lleva acabo la operación con la cantidad de bytes necesarios solamente.

Una ventaja que es importante mencionar acerca de CCMP/AES es que en la desencriptación no se necesita usar el AES en forma inversa como se

podría llegara suponer. Esto se debe a que AES se usa en la encriptación del contador, proceso que es implementado tanto del lado transmisor como del receptor.

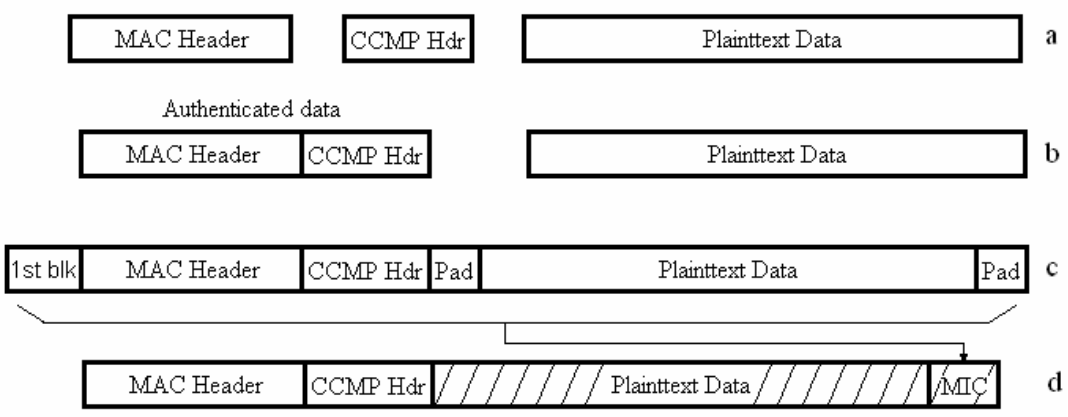
CCMP/AES y el adaptador

A partir de aquí se comenzará a ver la encriptación que se realizó en el driver detalladamente.

Se parte de las MPDUs originadas por la fragmentación de las MSDUs, que llegan desde la capa de red para ser transmitidas al medio. Las MPDUs se alojan en el buffer del socket que en Linux está en una estructura llamada sk_buff. Un puntero a dicha estructura es conocido como skb. La estructura posee a su vez punteros llamados: head, data, tail y end, y una variable int DATA_len con la longitud del paquete. Haed y end apuntan al comienzo y al final respectivamente del buffer, data y end apuntan al principio y final respectivamente del paquete correspondiente. Es decir que data_len = cpu-to-be32(skb->tail – skb->data). La función cpu_to_be32 (o le32) se usa para convertir la distancia entre direcciones de memoria en su correspondiente valor en bytes.

Para mayor conocimiento de la estructura sk_buff consultar el ítem 3 de la bibliografía, capítulo 17.

Básicamente la encriptación sigue los pasos mostrados en la siguiente figura:



Se observa que a la MPDU (o plainext Data) se le suman un bloque llamado primer bloque (1st blk), el encabezado MAC más el encabezado CCMP (CH) y unos bytes en 0 (Pad, cuyo objetivo se discute a continuación) para constituir el MIC (que a partir de ahora se llamará mic_aes para diferenciarlo del mic usado en el 4-way-handshake).

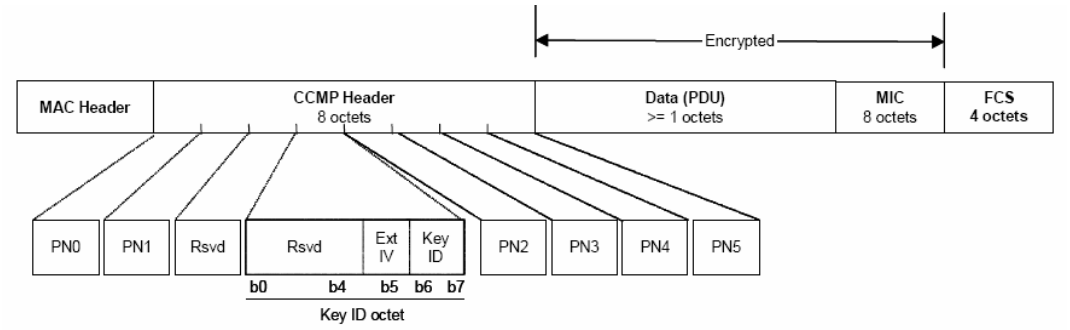
El mic_aes utiliza el método conocido como cbc-mac que sintéticamente consiste en:

- Toma el primer bloque del mensaje (16 bytes) y lo encripta con AES
- Hace la operación XOR del resultado anterior con el siguiente bloque del mensaje y encripta el resultado.
- Realiza la or-exclusiva del resultado anterior y el próximo bloque y así sucesivamente hasta llegar al último bloque.

Instantáneamente se puede ver la razón de ser de los bytes Pad antes mencionados: CBC-MAC necesita sí o sí que la longitud del mensaje a chequear sea múltiplo de 16, de ahí que se deba agregar bytes en cero tanto a *Authenticated data* como a *Plaintext data.*

CCMP header

El encabezado CCMP es detallado aquí:



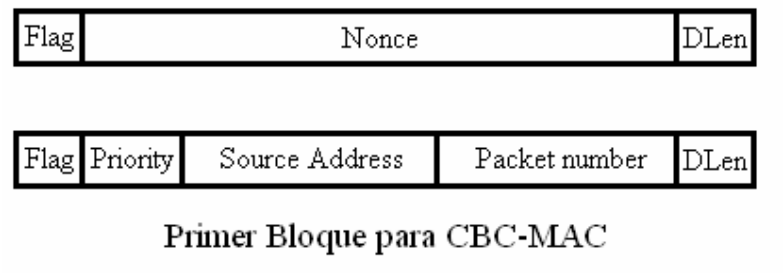
El PN (Packet Number) consta de 48 bits y es utilizado para la operación de encriptado bajo CCMP/AES y desencriptado. El PN es una secuencia pseudoaleatoria con el objetivo de que sea única para en cada pairwise key. PN es incrementado en 1 con cada MPDU.

En el caso de mensajes con pairwise keys, Key ID posee el valor 0, y Ext IV es 1.

First Block

El 1st blk o primer bloque desarrollado para la construcción del mic_aes consiste en un primer byte llamado *flag* seguido de un nonce de 104 bits y por último un byte llamado *DLen.*

El campo *Flag* lleva el valor 0x59 e indica varias cosas, entre otras que



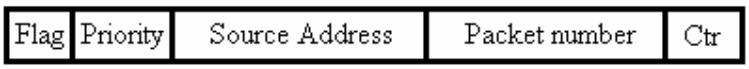
Primer Bloque para CBC-MAC

el valor del mic_aes es de 8 bits. *DLen* indica la longitud de *Plaintext Data.*

Como se puede ver en las figuras, el nonce es armado con un campo llamado *Priority e*l cual fue pensado para futuras ampliaciones y lleva el valor 0x00 actualmente. *Source Address* es la dirección MAC de quien envía el mensaje. Como ya se vio recientemente, el *Packet Number* es el mismo que aparece en el CCMP Header.

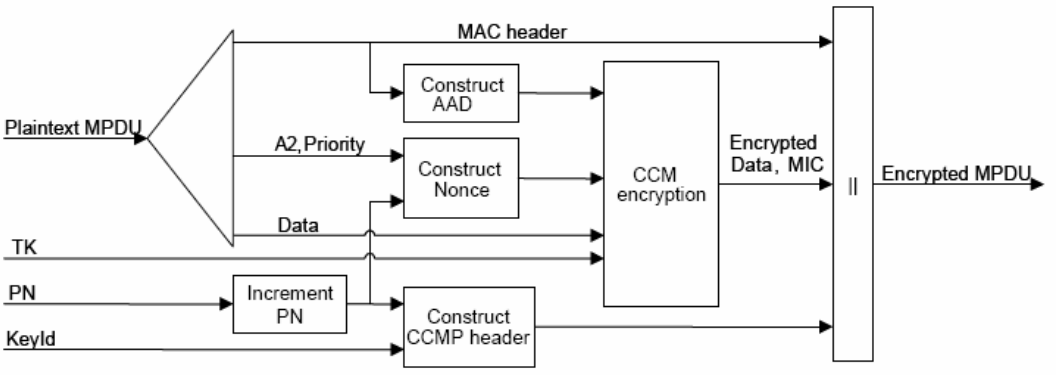
Contador para CCMP

El contador usado para la operación del *Counter Mode* es construido en forma similar al *1st block*, como se observa en la siguiente figura, solo se diferencia en los dos últimos bytes dado que cambia el campo *DLen* por *Ctr* (Counter). El campo *Ctr* es la pieza clave del contador, parte del valor 1 y puede llegar hasta 65535 con lo que puede acomodar la MPDU más larga permitida en 802.11. Los demás campos están para inicializar el contador de forma tal de eliminar el uso del mismo valor inicial dos veces mediante el uso del PN.



Contador para CCMP AES

La siguiente figura resume el proceso de encriptado:



CCMP encapsulation block diagram

Desencriptación

Del lado receptor, arriba una MPDU encriptada como se vio recientemente.

Se puede resumir el proceso de desencriptación en cinco pasos:

- Leer el PN directamente de la trama recibida y compararlo con el recibido en el mensaje anterior, si aquel es menor o igual a éste, la trama debe ser descartada.

- Preparar el contador para el proceso de desencapsulado (decapsulation). La información necesaria para esto se encuentra en la trama recibida*: Source Address, PN*, más los *flags* cuyos valores son conocidos y constantes.

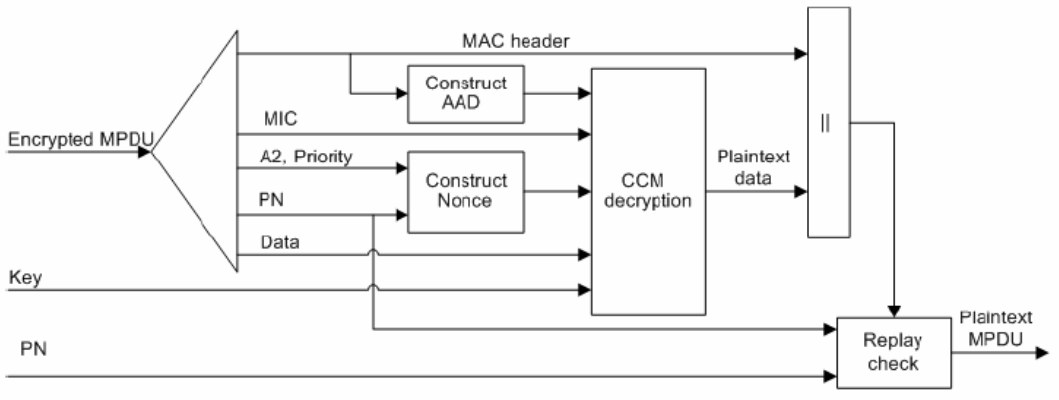
- Desencapsular la trama recibida con CCMP/AES.

- Armar los datos correspondientes para procesar el mic_aes y verificar que el recibido es correcto sino descartar la trama.

- Pasar la MPDU a las capas superiores.

Aquí se representa la desencriptación gráficamente:

Tolo lo referente al paso de tramas entre el buffer del socket y la estructura interna del driver está en el archivo *acx100_conv.c*, por lo tanto



CCMP decapsulation block diagram

para aplicar la encriptación CCMP/AES se desarrollaron archivos cabeceras incluidos en este archivo.

Para llevar a cabo la operación de enriptación hay que insertar una subcapa entre los datos a transmitir y la operación de transmisión que lleve a cabo el proceso deseado.

De la misma forma, para la desencriptación se debe implementar una subcapa entre los datos recibidos y el proceso de los mismos para llevarlos a las capas superiores.

Básicamente se hicieron dos archivos: *ccmp_enc.h* y *ccmp_des_enc.h*. Estos archivos son incluidos en *acx100_conv.c* más otro denominado *aes.h* mediante ccmp_*conv.h* simplemente para la prolijidad del programa.

Aes.h es similar a lo hecho en la primera etapa del proyecto pero con algunas modificaciones dado que aquí es un archivo *include* con definiciones de funciones y por lo tanto no posee *"main()"*.

El archivo *acx100_conv.c* contiene dos funciones las cuales son llamadas para transformar tramas desde formato ethernet a 802.11 y viceversa. Ellas son acx_ether_to_txdesc y acx_rxdesc_to_ether.

La encriptación se debe implementar en la primera mencionada. Ésta función es llamada pasándole como parámetros: un puntero a la estructura privada del dispositivo wlandevice_t, un puntero a la estructura txhostdescriptor y un puntero a la estructura sk_buff.

La función primero clasifica la trama que arribó al buffer del socket y luego pasa los datos a la estructura *txdescriptor* para ser transmitidos al medio. Esa línea de código es reemplazada por la llamada a la función *enc* (definida en *ccmp_enc.h*) la cual devuelve la longitud de los datos encriptados listos para ser enviados al medio y copia los datos encriptados en la variable payload->length.

Las tramas que arriban del medio y deben ser pasadas al sistema operativo a través del buffer del socket lo hacen por medio de la función *acx_rxdesc_to_ether*.

La función hace primero un chequeo por si hay header adicional y luego ya comienza a trabajar con los datos recibidos. Ósea que antes de éstas líneas de código se deben desencriptar los datos llamando a la función *des_enc* definida en *CCMP_des_enc.h*.

La función comienza armando el PN haciendo uso de la función PRF. El PN es usado en el header ccmp, en el primer bloque y en el contador. Luego arma el primer bloque, el header MAC con los bits correspondientes en 0, la trama para computar el mic_aes mediante la función cbc_mac. Después de hecho el *mic_aes* arma la trama con la MPDU para ser encriptada con ccmp/aes no antes de armar el contador. La función devuelve la longitud de los datos a transmitir que como se vio es la longitud de Plaintext más 16 bytes.

La función cbc_mac está definida en *ccmp_des_enc.h*, por eso éste archivo está incluido primero que *ccmp_enc.h* en *ccmp_conv_.h*.

Con respecto a la desencriptación, a la trama recibida hay que extraerles el header ccmp y de ésta formar el PN. Con ésto más la dirección MAC del punto de acceso y el flag 0x59 se arma el contador para comenzar la desencriptación con ccmp/aes.

Luego se arma la trama para chequear el *mic_aes* para finalmente, si todo resultó bien, pasarla al buffer del socket.

Archivos y funciones usados para la segunda etapa

globals.h

```
/***************variables globales agregadas a acx100_helper2.h****************/
/*****************************para 4-way.h************************************/
u8 pmk[32];
u8 ptk[48];
u8 EAPOL_mic_key[16];
u8 snonce[32];
struct WPA_fr {
   u8 EAPOL_HDR[4];
   u8 descriptor_type;
   u8 key_information[2];
   u8 key_length[2];
   u8 replay_counter[8];
   u8 key_nonce[32];
   u8 EAPOL_key_iv[16];
   u8 rsc[8];
   u8 key_identifier[8];
   u8 key_mic[16];
   u8 key_data_length[2];
} WPA_frame;
u8 replay_counter[] = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01};
u8 EAPOL_HDR[] = {0x01, 0x03, 0x00, 0x00};
u8 WPA_IE[] = {0xdd, 0x16, 0x00, 0x50, 0xf2, 0x01, 0x01, 0x00,
        0x00, 0x50, 0xf2, 0x04, 0x01, 0x00, 0x00, 0x50,
        0xf2, 0x04, 0x01, 0x00, 0x00, 0x50, 0xf2, 0x02};
u8 mic[20];
u8 frame[140];
/*****************************************************************************/
/************************variables globales de key.h**************************/
extern u8 s_box[256];
extern u32 key_array[44];
extern u32 rcon[10];
u8 DATA_enc_key[16];
/*****************************************************************************/
```

Como regla general en el Informe del Proyecto, se decidió que en cualquier archivo existente en el driver original en el cual haya que agregar o modificar código, la modificación se marcará con dos renglones de asteriscos entre barras, uno al principio y otro al final de la modificación, y resaltados con letra *negrita* para su rápida ubicación.

acx_transmit_assoc_req

```
/*
* Arguments:
*
* Returns:
*
* Side effects:
*
* Call context:
*
* STATUS:
*
* Comment:
*
*---------------------------------------------------------------*/

/* acx_transmit_assoc_req()
 * STATUS: almost ok, but UNVERIFIED.
 */
static int acx_transmit_assoc_req(wlandevice_t *priv)
{
    unsigned int packet_len;
    struct txdescriptor *tx_desc;
    struct txhostdescriptor *header;
    struct txhostdescriptor *payload;
    TxData *hd;
    u8 *pCurrPos;

    FN_ENTER;
```

```
   acxlog(L_BINSTD | L_ASSOC, "Sending association request, awaiting response!
NOT ASSOCIATED YET.\n");
   if ((tx_desc = acx_get_tx_desc(priv)) == NULL) {
            FN_EXIT(1, NOT_OK);
            return NOT_OK;
   }

   packet_len = WLAN_HDR_A3_LEN;

   header = tx_desc->fixed_size.s.host_desc;      /* hostdescriptor for header */
   payload = tx_desc->fixed_size.s.host_desc + 1; /* hostdescriptor for payload */

   hd = (TxData *)header->data;
   pCurrPos = (u8 *)payload->data;
   hd->frame_control                                                              =
host2ieee16(WLAN_SET_FC_FSTYPE(WLAN_FSTYPE_ASSOCREQ));  /* 0x00 */;
   hd->duration_id = host2ieee16(0x8000);

   MAC_COPY(hd->da, priv->bssid);
   MAC_COPY(hd->sa, priv->dev_addr);
   MAC_COPY(hd->bssid, priv->bssid);

   hd->sequence_control = 0;

   header->length = cpu_to_le16(WLAN_HDR_A3_LEN);
   header->data_offset = 0;

   /* now start filling the AssocReq frame body */

#if BROKEN
   *(u16          *)pCurrPos          =          host2ieee16(priv->capabilities          &
~(WLAN_SET_MGMT_CAP_INFO_IBSS(1)));
#else
   /* FIXME: is it correct that we have to manually patc^H^H^H^Hadjust the
    * Capabilities like that?
    * I'd venture that priv->capabilities
```

```
    * (acx_update_capabilities()) should have set that
    * beforehand maybe...
    * Anyway, now Managed network association works properly
    * without failing.
    */

   /*(u16 *)pCurrPos = host2ieee16((priv->capabilities &
~(WLAN_SET_MGMT_CAP_INFO_IBSS(1))) |
WLAN_SET_MGMT_CAP_INFO_ESS(1));*/

   *(u16 *)pCurrPos = host2ieee16(WLAN_SET_MGMT_CAP_INFO_ESS(1));
   if ((u8)0 != priv->wep_restricted)
           SET_BIT(*(u16 *)pCurrPos,
host2ieee16(WLAN_SET_MGMT_CAP_INFO_PRIVACY(1)));
   /* only ask for short preamble if the peer station supports it */
   if (priv->station_assoc.caps & IEEE802_11_MGMT_CAP_SHORT_PRE)
           SET_BIT(*(u16 *)pCurrPos,
host2ieee16(WLAN_SET_MGMT_CAP_INFO_SHORT(1)));
   /* only ask for PBCC support if the peer station supports it */
   if (priv->station_assoc.caps & IEEE802_11_MGMT_CAP_PBCC)
           SET_BIT(*(u16 *)pCurrPos,
host2ieee16(WLAN_SET_MGMT_CAP_INFO_PBCC(1)));
#endif
   acxlog(L_ASSOC, "association: requesting capabilities 0x%04X\n", *(u16
*)pCurrPos);
   pCurrPos += 2;

   /* add listen interval */
   *(u16 *)pCurrPos = host2ieee16(priv->listen_interval);
   pCurrPos += 2;

   /* add ESSID */
   *(u8 *)pCurrPos = (u8)0; /* Element ID */
   pCurrPos += 1;
   *(u8 *)pCurrPos = (u8)strlen(priv->essid_for_assoc); /* Length */
   memcpy(&pCurrPos[1], priv->essid_for_assoc, pCurrPos[0]);
   pCurrPos += 1 + pCurrPos[0];
```

```
    /* add rates */
    *(u8 *)pCurrPos = (u8)1; /* Element ID */
    pCurrPos += 1;
    *(u8 *)pCurrPos = priv->rate_supported_len; /* Length */
    pCurrPos += 1;
    memcpy(pCurrPos, priv->rate_supported, priv->rate_supported_len);
    pCurrPos += priv->rate_supported_len;
/*******************************************************************************/
    /*add the information element: WPA IE */
    memcpy(pCurrPos, WPA_IE, WPA_IE_len);
    pCurrPos += WPA_IE_len;
/*******************************************************************************/
    /* calculate lengths */
    packet_len += (int)pCurrPos - (int)payload->data;

    payload->length = cpu_to_le16(packet_len - WLAN_HDR_A3_LEN);
    payload->data_offset = 0;

    tx_desc->total_length = cpu_to_le16(packet_len);

    acx_dma_tx_data(priv, tx_desc);
    FN_EXIT(1, OK);
    return OK;
    /* calculate lengths */
    packet_len += (int)pCurrPos - (int)payload->data;

    payload->length = cpu_to_le16(packet_len - WLAN_HDR_A3_LEN);
    payload->data_offset = 0;

    tx_desc->total_length = cpu_to_le16(packet_len);

    acx_dma_tx_data(priv, tx_desc);
    FN_EXIT(1, OK);
    return OK;
}
```

Passwordhash.h

```
u8 F(char *password, unsigned char *ssid,int ssidlength, int iterations,
int count, unsigned char *output);
u8 F(char *password,unsigned char *ssid,int ssidlength, int iterations,
int count, unsigned char *output) {
   unsigned char digest[36], digest1[SHA1_DIGEST_SIZE];
   int i, j;

   /* U1 = PRF(P, S || int(i)) */
   memcpy(digest, ssid, ssidlength);
   digest[ssidlength] = (unsigned char)((count>>24) & 0xff);
   digest[ssidlength+1] = (unsigned char)((count>>16) & 0xff);
   digest[ssidlength+2] = (unsigned char)((count>>8) & 0xff);
   digest[ssidlength+3] = (unsigned char)(count & 0xff);
   if (hmac_sha1(digest, ssidlength + 4,(unsigned char*) password,
   (int) strlen(password), digest1)) return 1;
   /* output = U1 */
   memcpy(output, digest1, SHA1_DIGEST_SIZE);
   for (i = 1; i < iterations; i++) {
      /* Un = PRF(P, Un-1) */
   if (hmac_sha1(digest1, SHA1_DIGEST_SIZE, (unsigned char*) password,
   (int) strlen(password), digest)) return 1;
   memcpy(digest1, digest, SHA1_DIGEST_SIZE);
   /* output = output xor Un */
   for (j = 0; j < SHA1_DIGEST_SIZE; j++) {
      output[j] ^= digest[j];
   }
   }
   return 0;
}
/*
* password - ascii string up to 63 characters in length
* ssid - octet string up to 32 octets
* ssidlength - length of ssid in octets
* output must be 40 octets in length and outputs 256 bits of key
*/
u8 PasswordHash (char *password,unsigned char *ssid,int ssidlength,unsigned char
*output);
u8 PasswordHash (char *password,unsigned char *ssid,int ssidlength,unsigned char
*output) {
   unsigned char i;
   if ((strlen(password) > 63) || (ssidlength > 32) || (strlen(password) < 8)) {
   acxlog(L_BINSTD | L_ASSOC, "Argumentos de Passwordhash no validos");
      return 0;
   }
   for (i = 0; i < strlen(password); i++)
   if ((password[i] < 32) || (password[i] > 126)) return 0;

   if (F(password, ssid, ssidlength, 4096, 1, output)) return 0;

   if (F(password, ssid, ssidlength, 4096, 2,&output[SHA1_DIGEST_SIZE]))
   return 0;
   return 1;
```

```
}
```

sha1.h

```
/*
 *  FIPS-180-1 compliant SHA-1 implementation
 *
 *  Copyright (C) 2001-2003  Christophe Devine
 *
 *  This program is free software; you can redistribute it and/or modify
 *  it under the terms of the GNU General Public License as published by
 *  the Free Software Foundation; either version 2 of the License, or
 *  (at your option) any later version.
 *
 *  This program is distributed in the hope that it will be useful,
 *  but WITHOUT ANY WARRANTY; without even the implied warranty of
 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *  GNU General Public License for more details.
 *
 *  You should have received a copy of the GNU General Public License
 *  along with this program; if not, write to the Free Software
 *  Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
 */

/*#include <string.h>*/

#ifndef uint8
#define uint8 unsigned char
#endif

#ifndef uint32
#define uint32 unsigned int
#endif

typedef struct {
    uint32 total[2];
    uint32 state[5];
    uint8 buffer[64];
} sha1_context;

/* uncomment the following line to run the test suite */

/* #define TEST  */

#define GET_UINT32(n,b,i)                       \
{                                               \
    (n) = ( (uint32) (b)[(i)    ] << 24 )       \
        | ( (uint32) (b)[(i) + 1] << 16 )       \
        | ( (uint32) (b)[(i) + 2] <<  8 )       \
        | ( (uint32) (b)[(i) + 3]       );      \
}

#define PUT_UINT32(n,b,i)                       \
{                                               \
```

```
    (b)[(i)    ] = (uint8) ( (n) >> 24 );       \
    (b)[(i) + 1] = (uint8) ( (n) >> 16 );       \
    (b)[(i) + 2] = (uint8) ( (n) >>  8 );       \
    (b)[(i) + 3] = (uint8) ( (n)       );       \
}

void sha1_init(sha1_context *ctx);
void sha1_init( sha1_context *ctx )
{
    ctx->total[0] = 0;
    ctx->total[1] = 0;

    ctx->state[0] = 0x67452301;
    ctx->state[1] = 0xEFCDAB89;
    ctx->state[2] = 0x98BADCFE;
    ctx->state[3] = 0x10325476;
    ctx->state[4] = 0xC3D2E1F0;
}

void sha1_process( sha1_context *ctx, uint8 data[64]);
void sha1_process( sha1_context *ctx, uint8 data[64])
{
    uint32 temp, W[16], A, B, C, D, E;

    GET_UINT32( W[0],  data,  0 );
    GET_UINT32( W[1],  data,  4 );
    GET_UINT32( W[2],  data,  8 );
    GET_UINT32( W[3],  data, 12 );
    GET_UINT32( W[4],  data, 16 );
    GET_UINT32( W[5],  data, 20 );
    GET_UINT32( W[6],  data, 24 );
    GET_UINT32( W[7],  data, 28 );
    GET_UINT32( W[8],  data, 32 );
    GET_UINT32( W[9],  data, 36 );
    GET_UINT32( W[10], data, 40 );
    GET_UINT32( W[11], data, 44 );
    GET_UINT32( W[12], data, 48 );
    GET_UINT32( W[13], data, 52 );
    GET_UINT32( W[14], data, 56 );
    GET_UINT32( W[15], data, 60 );

#define S(x,n) ((x << n) | ((x & 0xFFFFFFFF) >> (32 - n)))

#define R(t)                                        \
(                                                   \
    temp = W[(t -  3) & 0x0F] ^ W[(t - 8) & 0x0F] ^     \
           W[(t - 14) & 0x0F] ^ W[ t      & 0x0F],      \
    ( W[t & 0x0F] = S(temp,1) )                         \
)

#define P(a,b,c,d,e,x)                              \
{                                                   \
    e += S(a,5) + F(b,c,d) + K + x; b = S(b,30);        \
}
```

```
    A = ctx->state[0];
    B = ctx->state[1];
    C = ctx->state[2];
    D = ctx->state[3];
    E = ctx->state[4];

#define F(x,y,z) (z ^ (x & (y ^ z)))
#define K 0x5A827999

    P( A, B, C, D, E, W[0]  );
    P( E, A, B, C, D, W[1]  );
    P( D, E, A, B, C, W[2]  );
    P( C, D, E, A, B, W[3]  );
    P( B, C, D, E, A, W[4]  );
    P( A, B, C, D, E, W[5]  );
    P( E, A, B, C, D, W[6]  );
    P( D, E, A, B, C, W[7]  );
    P( C, D, E, A, B, W[8]  );
    P( B, C, D, E, A, W[9]  );
    P( A, B, C, D, E, W[10] );
    P( E, A, B, C, D, W[11] );
    P( D, E, A, B, C, W[12] );
    P( C, D, E, A, B, W[13] );
    P( B, C, D, E, A, W[14] );
    P( A, B, C, D, E, W[15] );
    P( E, A, B, C, D, R(16) );
    P( D, E, A, B, C, R(17) );
    P( C, D, E, A, B, R(18) );
    P( B, C, D, E, A, R(19) );

#undef K
#undef F

#define F(x,y,z) (x ^ y ^ z)
#define K 0x6ED9EBA1

    P( A, B, C, D, E, R(20) );
    P( E, A, B, C, D, R(21) );
    P( D, E, A, B, C, R(22) );
    P( C, D, E, A, B, R(23) );
    P( B, C, D, E, A, R(24) );
    P( A, B, C, D, E, R(25) );
    P( E, A, B, C, D, R(26) );
    P( D, E, A, B, C, R(27) );
    P( C, D, E, A, B, R(28) );
    P( B, C, D, E, A, R(29) );
    P( A, B, C, D, E, R(30) );
    P( E, A, B, C, D, R(31) );
    P( D, E, A, B, C, R(32) );
    P( C, D, E, A, B, R(33) );
    P( B, C, D, E, A, R(34) );
    P( A, B, C, D, E, R(35) );
    P( E, A, B, C, D, R(36) );
    P( D, E, A, B, C, R(37) );
    P( C, D, E, A, B, R(38) );
```

```
    P( B, C, D, E, A, R(39) );

#undef K
#undef F

#define F(x,y,z) ((x & y) | (z & (x | y)))
#define K 0x8F1BBCDC

    P( A, B, C, D, E, R(40) );
    P( E, A, B, C, D, R(41) );
    P( D, E, A, B, C, R(42) );
    P( C, D, E, A, B, R(43) );
    P( B, C, D, E, A, R(44) );
    P( A, B, C, D, E, R(45) );
    P( E, A, B, C, D, R(46) );
    P( D, E, A, B, C, R(47) );
    P( C, D, E, A, B, R(48) );
    P( B, C, D, E, A, R(49) );
    P( A, B, C, D, E, R(50) );
    P( E, A, B, C, D, R(51) );
    P( D, E, A, B, C, R(52) );
    P( C, D, E, A, B, R(53) );
    P( B, C, D, E, A, R(54) );
    P( A, B, C, D, E, R(55) );
    P( E, A, B, C, D, R(56) );
    P( D, E, A, B, C, R(57) );
    P( C, D, E, A, B, R(58) );
    P( B, C, D, E, A, R(59) );

#undef K
#undef F

#define F(x,y,z) (x ^ y ^ z)
#define K 0xCA62C1D6

    P( A, B, C, D, E, R(60) );
    P( E, A, B, C, D, R(61) );
    P( D, E, A, B, C, R(62) );
    P( C, D, E, A, B, R(63) );
    P( B, C, D, E, A, R(64) );
    P( A, B, C, D, E, R(65) );
    P( E, A, B, C, D, R(66) );
    P( D, E, A, B, C, R(67) );
    P( C, D, E, A, B, R(68) );
    P( B, C, D, E, A, R(69) );
    P( A, B, C, D, E, R(70) );
    P( E, A, B, C, D, R(71) );
    P( D, E, A, B, C, R(72) );
    P( C, D, E, A, B, R(73) );
    P( B, C, D, E, A, R(74) );
    P( A, B, C, D, E, R(75) );
    P( E, A, B, C, D, R(76) );
    P( D, E, A, B, C, R(77) );
    P( C, D, E, A, B, R(78) );
    P( B, C, D, E, A, R(79) );
```

```
#undef K
#undef F

    ctx->state[0] += A;
    ctx->state[1] += B;
    ctx->state[2] += C;
    ctx->state[3] += D;
    ctx->state[4] += E;
}

void sha1_update(sha1_context *ctx, uint8 *input, uint32 length);
void sha1_update( sha1_context *ctx, uint8 *input, uint32 length )
{
    uint32 left, fill;

    if( ! length ) return;

    left = ctx->total[0] & 0x3F;
    fill = 64 - left;

    ctx->total[0] += length;
    ctx->total[0] &= 0xFFFFFFFF;

    if( ctx->total[0] < length )
        ctx->total[1]++;

    if( left && length >= fill )
    {
        memcpy( (void *) (ctx->buffer + left),
                (void *) input, fill );
        sha1_process( ctx, ctx->buffer );
        length -= fill;
        input  += fill;
        left = 0;
    }

    while( length >= 64 )
    {
        sha1_process( ctx, input );
        length -= 64;
        input  += 64;
    }

    if( length )
    {
        memcpy( (void *) (ctx->buffer + left),
                (void *) input, length );
    }
}

static uint8 sha1_padding[64] =
{
 0x80, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

```
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
};

void sha1_final(sha1_context *ctx, uint8 digest[20]);
void sha1_final(sha1_context *ctx, uint8 digest[20] )
{
    uint32 last, padn;
    uint32 high, low;
    uint8 msglen[8];

    high = ( ctx->total[0] >> 29 )
         | ( ctx->total[1] <<  3 );
    low  = ( ctx->total[0] <<  3 );

    PUT_UINT32( high, msglen, 0 );
    PUT_UINT32( low,  msglen, 4 );

    last = ctx->total[0] & 0x3F;
    padn = ( last < 56 ) ? ( 56 - last ) : ( 120 - last );

    sha1_update( ctx, sha1_padding, padn );
    sha1_update( ctx, msglen, 8 );

    PUT_UINT32( ctx->state[0], digest,  0 );
    PUT_UINT32( ctx->state[1], digest,  4 );
    PUT_UINT32( ctx->state[2], digest,  8 );
    PUT_UINT32( ctx->state[3], digest, 12 );
    PUT_UINT32( ctx->state[4], digest, 16 );
}
```

md5.h

```
/*
 * Cryptographic API.
 *
 * MD5 Message Digest Algorithm (RFC1321).
 *
 * Derived from cryptoapi implementation, originally based on the
 * public domain implementation written by Colin Plumb in 1993.
 *
 * Copyright (c) Cryptoapi developers.
 * Copyright (c) 2002 James Morris <jmorris@intercode.com.au>
 *
 * This program is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by the Free
 * Software Foundation; either version 2 of the License, or (at your option)
 * any later version.
 *
 */
```

```
/*#include <linux/init.h>
*#include <linux/module.h>
*#include <linux/string.h>
*#include <linux/crypto.h>*/
#include <asm/byteorder.h>

#define MD5_DIGEST_SIZE             16
#define MD5_HMAC_BLOCK_SIZE         64
#define MD5_BLOCK_WORDS             16
#define MD5_HASH_WORDS              4

#define u8 unsigned char
#define u16 unsigned short
#define u32 unsigned int
#define u64 unsigned long long

#define F1(x, y, z)         (z ^ (x & (y ^ z)))
#define F2(x, y, z)         F1(z, x, y)
#define F3(x, y, z)         (x ^ y ^ z)
#define F4(x, y, z)         (y ^ (x | ~z))

#define MD5STEP(f, w, x, y, z, in, s) \
   (w += f(x, y, z) + in, w = (w<<s | w>>(32-s)) + x)

struct md5_ctx {
   u32 hash[MD5_HASH_WORDS];
   u32 block[MD5_BLOCK_WORDS];
   u64 byte_count;
};

static void md5_transform(u32 *hash, u32 const *in);
static void md5_transform(u32 *hash, u32 const *in)
{
   u32 a, b, c, d;

   a = hash[0];
   b = hash[1];
   c = hash[2];
   d = hash[3];

   MD5STEP(F1, a, b, c, d, in[0] + 0xd76aa478, 7);
   MD5STEP(F1, d, a, b, c, in[1] + 0xe8c7b756, 12);
   MD5STEP(F1, c, d, a, b, in[2] + 0x242070db, 17);
   MD5STEP(F1, b, c, d, a, in[3] + 0xc1bdceee, 22);
   MD5STEP(F1, a, b, c, d, in[4] + 0xf57c0faf, 7);
   MD5STEP(F1, d, a, b, c, in[5] + 0x4787c62a, 12);
   MD5STEP(F1, c, d, a, b, in[6] + 0xa8304613, 17);
   MD5STEP(F1, b, c, d, a, in[7] + 0xfd469501, 22);
   MD5STEP(F1, a, b, c, d, in[8] + 0x698098d8, 7);
   MD5STEP(F1, d, a, b, c, in[9] + 0x8b44f7af, 12);
   MD5STEP(F1, c, d, a, b, in[10] + 0xffff5bb1, 17);
   MD5STEP(F1, b, c, d, a, in[11] + 0x895cd7be, 22);
   MD5STEP(F1, a, b, c, d, in[12] + 0x6b901122, 7);
   MD5STEP(F1, d, a, b, c, in[13] + 0xfd987193, 12);
   MD5STEP(F1, c, d, a, b, in[14] + 0xa679438e, 17);
```

```
MD5STEP(F1, b, c, d, a, in[15] + 0x49b40821, 22);

MD5STEP(F2, a, b, c, d, in[1] + 0xf61e2562, 5);
MD5STEP(F2, d, a, b, c, in[6] + 0xc040b340, 9);
MD5STEP(F2, c, d, a, b, in[11] + 0x265e5a51, 14);
MD5STEP(F2, b, c, d, a, in[0] + 0xe9b6c7aa, 20);
MD5STEP(F2, a, b, c, d, in[5] + 0xd62f105d, 5);
MD5STEP(F2, d, a, b, c, in[10] + 0x02441453, 9);
MD5STEP(F2, c, d, a, b, in[15] + 0xd8a1e681, 14);
MD5STEP(F2, b, c, d, a, in[4] + 0xe7d3fbc8, 20);
MD5STEP(F2, a, b, c, d, in[9] + 0x21e1cde6, 5);
MD5STEP(F2, d, a, b, c, in[14] + 0xc33707d6, 9);
MD5STEP(F2, c, d, a, b, in[3] + 0xf4d50d87, 14);
MD5STEP(F2, b, c, d, a, in[8] + 0x455a14ed, 20);
MD5STEP(F2, a, b, c, d, in[13] + 0xa9e3e905, 5);
MD5STEP(F2, d, a, b, c, in[2] + 0xfcefa3f8, 9);
MD5STEP(F2, c, d, a, b, in[7] + 0x676f02d9, 14);
MD5STEP(F2, b, c, d, a, in[12] + 0x8d2a4c8a, 20);

MD5STEP(F3, a, b, c, d, in[5] + 0xfffa3942, 4);
MD5STEP(F3, d, a, b, c, in[8] + 0x8771f681, 11);
MD5STEP(F3, c, d, a, b, in[11] + 0x6d9d6122, 16);
MD5STEP(F3, b, c, d, a, in[14] + 0xfde5380c, 23);
MD5STEP(F3, a, b, c, d, in[1] + 0xa4beea44, 4);
MD5STEP(F3, d, a, b, c, in[4] + 0x4bdecfa9, 11);
MD5STEP(F3, c, d, a, b, in[7] + 0xf6bb4b60, 16);
MD5STEP(F3, b, c, d, a, in[10] + 0xbebfbc70, 23);
MD5STEP(F3, a, b, c, d, in[13] + 0x289b7ec6, 4);
MD5STEP(F3, d, a, b, c, in[0] + 0xeaa127fa, 11);
MD5STEP(F3, c, d, a, b, in[3] + 0xd4ef3085, 16);
MD5STEP(F3, b, c, d, a, in[6] + 0x04881d05, 23);
MD5STEP(F3, a, b, c, d, in[9] + 0xd9d4d039, 4);
MD5STEP(F3, d, a, b, c, in[12] + 0xe6db99e5, 11);
MD5STEP(F3, c, d, a, b, in[15] + 0x1fa27cf8, 16);
MD5STEP(F3, b, c, d, a, in[2] + 0xc4ac5665, 23);

MD5STEP(F4, a, b, c, d, in[0] + 0xf4292244, 6);
MD5STEP(F4, d, a, b, c, in[7] + 0x432aff97, 10);
MD5STEP(F4, c, d, a, b, in[14] + 0xab9423a7, 15);
MD5STEP(F4, b, c, d, a, in[5] + 0xfc93a039, 21);
MD5STEP(F4, a, b, c, d, in[12] + 0x655b59c3, 6);
MD5STEP(F4, d, a, b, c, in[3] + 0x8f0ccc92, 10);
MD5STEP(F4, c, d, a, b, in[10] + 0xffeff47d, 15);
MD5STEP(F4, b, c, d, a, in[1] + 0x85845dd1, 21);
MD5STEP(F4, a, b, c, d, in[8] + 0x6fa87e4f, 6);
MD5STEP(F4, d, a, b, c, in[15] + 0xfe2ce6e0, 10);
MD5STEP(F4, c, d, a, b, in[6] + 0xa3014314, 15);
MD5STEP(F4, b, c, d, a, in[13] + 0x4e0811a1, 21);
MD5STEP(F4, a, b, c, d, in[4] + 0xf7537e82, 6);
MD5STEP(F4, d, a, b, c, in[11] + 0xbd3af235, 10);
MD5STEP(F4, c, d, a, b, in[2] + 0x2ad7d2bb, 15);
MD5STEP(F4, b, c, d, a, in[9] + 0xeb86d391, 21);

hash[0] += a;
hash[1] += b;
```

```
    hash[2] += c;
    hash[3] += d;
}

/* XXX: this stuff can be optimized */
static inline void le32_to_cpu_array(u32 *buf, unsigned int words);
static inline void le32_to_cpu_array(u32 *buf, unsigned int words)
{
    while (words--) {
            __le32_to_cpus(buf);
            buf++;
    }
}

static inline void cpu_to_le32_array(u32 *buf, unsigned int words);
static inline void cpu_to_le32_array(u32 *buf, unsigned int words)
{
    while (words--) {
            __cpu_to_le32s(buf);
            buf++;
    }
}

static inline void md5_transform_helper(struct md5_ctx *ctx);
static inline void md5_transform_helper(struct md5_ctx *ctx)
{
    le32_to_cpu_array(ctx->block, sizeof(ctx->block) / sizeof(u32));
    md5_transform(ctx->hash, ctx->block);
}

static void md5_init(void *ctx);
static void md5_init(void *ctx)
{
    struct md5_ctx *mctx = ctx;

    mctx->hash[0] = 0x67452301;
    mctx->hash[1] = 0xefcdab89;
    mctx->hash[2] = 0x98badcfe;
    mctx->hash[3] = 0x10325476;
    mctx->byte_count = 0;
}

static void md5_update(void *ctx, const u8 *data, unsigned int len);
static void md5_update(void *ctx, const u8 *data, unsigned int len)
{
    struct md5_ctx *mctx = ctx;
    const u32 avail = sizeof(mctx->block) - (mctx->byte_count & 0x3f);

    mctx->byte_count += len;

    if (avail > len) {
            memcpy((char *)mctx->block + (sizeof(mctx->block) - avail),
                   data, len);
            return;
    }
```

```
    memcpy((char *)mctx->block + (sizeof(mctx->block) - avail),
          data, avail);

    md5_transform_helper(mctx);
    data += avail;
    len -= avail;

    while (len >= sizeof(mctx->block)) {
            memcpy(mctx->block, data, sizeof(mctx->block));
            md5_transform_helper(mctx);
            data += sizeof(mctx->block);
            len -= sizeof(mctx->block);
    }

    memcpy(mctx->block, data, len);
}

static void md5_final(void *ctx, u8 *out);
static void md5_final(void *ctx, u8 *out)
{
    struct md5_ctx *mctx = ctx;
    const unsigned int offset = mctx->byte_count & 0x3f;
    char *p = (char *)mctx->block + offset;
    int padding = 56 - (offset + 1);

    *p++ = 0x80;
    if (padding < 0) {
            memset(p, 0x00, padding + sizeof (u64));
            md5_transform_helper(mctx);
            p = (char *)mctx->block;
            padding = 56;
    }

    memset(p, 0, padding);
    mctx->block[14] = mctx->byte_count << 3;
    mctx->block[15] = mctx->byte_count >> 29;
    le32_to_cpu_array(mctx->block, (sizeof(mctx->block) -
                sizeof(u64)) / sizeof(u32));
    md5_transform(mctx->hash, mctx->block);
    cpu_to_le32_array(mctx->hash, sizeof(mctx->hash) / sizeof(u32));
    memcpy(out, mctx->hash, sizeof(mctx->hash));
    memset(mctx, 0, sizeof(*mctx));
}
```

hmac_sha1.h

```
#include <sha1.h>
#ifndef SHA1_DIGEST_SIZE
#define SHA1_DIGEST_SIZE 20
#endif
```

```
#ifndef SHA1_HMAC_BLOCK_SIZE
#define SHA1_HMAC_BLOCK_SIZE 64
#endif


void truncate(char *d1/*data to be truncated*/,char *d2/*truncated data*/, int len);

void truncate(char *d1/*data to be truncated*/,char *d2/*truncated data*/, int len) {
   int i;
   for(i = 00;i < len;i++) d2[i] =d1[i];
}

u8 hmac_sha1(unsigned char *text, int text_len, unsigned char *k, int key_len,
unsigned char *digest);

u8 hmac_sha1(unsigned char *text, int text_len, unsigned char *k, int key_len,
unsigned char *digest) {
   sha1_context ictx, octx;/****/
   char isha[SHA1_DIGEST_SIZE], osha[SHA1_DIGEST_SIZE];
   char key[SHA1_DIGEST_SIZE], buf[SHA1_HMAC_BLOCK_SIZE];
   int i, t = SHA1_DIGEST_SIZE;
   /*if key is longer than 64 bytes reset it to key = sha1(key)*/
   if (key_len > SHA1_HMAC_BLOCK_SIZE) {
   sha1_context tctx;/****/

   sha1_init(&tctx);
   sha1_update(&tctx, &k[0], key_len);
   sha1_final(&tctx, &key[0]);

   k = key;
   key_len = SHA1_DIGEST_SIZE;
   }

   /**********************inner Digest************************/
   sha1_init(&ictx);

   /*Pad the key for inner digest*/

   for(i = 0;i < key_len;i++) buf[i] = k[i] ^ 0x36;
   for(i = key_len;i < SHA1_HMAC_BLOCK_SIZE;i++) buf[i] = 0x36;

   sha1_update(&ictx, &buf[0],SHA1_HMAC_BLOCK_SIZE);
   sha1_update(&ictx, &text[0], text_len);

   sha1_final(&ictx, &isha[0]);

   /*Outter digest*/

   sha1_init(&octx);

   /*pad the key for outter digest*/

   for(i = 0;i < key_len;i++) buf[i] = k[i] ^ 0x5c;
   for(i = key_len;i < SHA1_HMAC_BLOCK_SIZE;i++) buf[i] = 0x5c;
```

```
    sha1_update(&octx, &buf[0], SHA1_HMAC_BLOCK_SIZE);
    sha1_update(&octx, &isha[0], SHA1_DIGEST_SIZE);

    sha1_final(&octx, &osha[0]);

    /*truncate*/
    t = t > SHA1_DIGEST_SIZE ? SHA1_DIGEST_SIZE:t;
    truncate(osha, digest, t);
    return 0;
}
```

hmac_md5.h

```
#include <md5.h>
#ifndef MD5_DIGESTSIZE
#define MD5_DIGESTSIZE 16
#endif

#ifndef MD5_BLOCKSIZE
#define MD5_BLOCKSIZE 64
#endif

inline u8 hmac_md5(u8 *d, int dl, u8 *k, int kl, u8 *out);
inline u8 hmac_md5(u8 *d, int dl, u8 *k, int kl, u8 *out) {
    struct md5_ctx context;
    unsigned char k_ipad[65], k_opad[65], tk[16];
    int i;

    FN_ENTER;

    if (kl > MD5_BLOCKSIZE) {
    struct md5_ctx tctx;

    md5_init(&tctx);
    md5_update(&tctx, k, kl);
    md5_final(&tctx, tk);

    k = tk;
    kl = MD5_DIGESTSIZE;
    }
    /**********Inner Digest**********/
    memset(k_ipad, 0x00, sizeof k_ipad);
    memset(k_opad, 0x00, sizeof k_opad);
    memcpy(k_ipad, k, kl);
    memcpy(k_opad, k, kl);

    /*pad the key for inner digest*/
    for(i = 0;i < MD5_BLOCKSIZE;i++) {
    k_ipad[i] ^= 0x36;
    k_opad[i] ^= 0x5c;
    }
```

```
    md5_init(&context);
    md5_update(&context, k_ipad, MD5_BLOCKSIZE);
    md5_update(&context, d, dl);

    md5_final(&context, out);
    /**********Outter Digest**********/

    md5_init(&context);

    md5_update(&context, k_opad, MD5_BLOCKSIZE);
    md5_update(&context, out, MD5_DIGESTSIZE);

    md5_final(&context, out);
    FN_EXIT(0, OK);
    return 0;
}
```

PRF.h

```
#include "hmac_sha1.h"

u8 PRF(unsigned char *key, int key_len, unsigned char *prefix, int prefix_len,
unsigned char *data, int data_len, unsigned char *output, int len);

u8 PRF(unsigned char *key, int key_len, unsigned char *prefix, int prefix_len,
unsigned char *data, int data_len, unsigned char *output, int len) {
  int i;
  unsigned char input[1024]; /*concatenated input*/
   int currentindex = 20;
   int total_len;
   memcpy(input, prefix, prefix_len);
   input[prefix_len] = 0; /*single octet 0*/
   memcpy(&input[prefix_len + 1], data, data_len);
   total_len = prefix_len + 1 + data_len;
   input[total_len] = 0; /*single octet count, starts at 0*/
   total_len++;
   for(i = 0;i < (len + 19)/20; i++) {
   if (hmac_sha1(input, total_len, key, key_len, output)) {
      return 1;
   }
   output += (unsigned char) currentindex;
   input[total_len - 1]++; /*increment octet count*/
   }
   return 0;
}
```

Auxiliar

```
void auxiliar(struct rxhostdescriptor *rxdesc, wlandevice_t *priv) {
   FN_ENTER;
   unsigned long final, inicio;
//    rdtscl(inicio);
```

```
   if ((PasswordHash("AccessPoint", priv->essid_for_assoc, strlen(priv->essid_for_assoc), pmk)))
   acxlog(L_BINSTD | L_ASSOC, "fallo en PasswordHash\n");

    u8 data[17];
    u8  random_number;
    u8 rand_snonce[32];
    u32 time;
   rdtscl(time);
/************************construccion del random number*********************/
   memcpy(data, priv->dev_addr, 6); /*dirección MAC del adaptaor inalámbrico*/
   data[6] = (u8)(time1);
   data[7] = (u8)(time1 >> 8);
   data[8] = (u8)(time1 >> 16);
   data[9] = (u8)(time1 >> 24);

if (PRF(&data[6], 4, "Random Number", 13, “The Best Random Number”, 22, rand_snonce, 32))
   acxlog(L_BINSTD | L_ASSOC, "fallo en PRF de random number\n");

/*************************construccion del snonce*****************************/
   if (PRF(random_number, 32, "Init Counter", 12, data, 17, rand_snonce, 32))
   acxlog(L_BINSTD | L_ASSOC, "fallo en PRF de snonoce\n");

/****************desarrollo de la pmk a partir de la pass_phrase**************/
   if (PRF(rand_snonce, 32, "Init Counter", 12, data, 17, snonce, nonce_len))
   acxlog(L_BINSTD | L_ASSOC, "fallo en PRF de nonce\n");
   init_WPA_frame();

  //  rdtscl(final);
 //   acxlog(L_BINSTD | L_ASSOC, "diferencia auxiliar: %ld us\n",(final - inicio)/100);

 FN_EXIT(0, OK);
}
```

Init_WPA_frame

```
void init_WPA_frame(void) {
   FN_ENTER;
   WPA_frame.EAPOL_HDR[0] = 0x01;
   WPA_frame.EAPOL_HDR[1] = 0x03;
   memset(&WPA_frame.EAPOL_HDR[2], 0x00, 2);
   WPA_frame.descriptor_type = 0xfe;
   memset(WPA_frame.key_information, 0x00, 2);
   WPA_frame.key_length[0] = 0x00;
   WPA_frame.key_length[1] = 0x10;
   memcpy(WPA_frame.key_nonce, snonce, 32);
   memset(WPA_frame.replay_counter, 0x00, 7);
   WPA_frame.replay_counter[7] = 0x01;
   memset(WPA_frame.key_nonce, 0x00, nonce_len);
   memset(WPA_frame.EAPOL_key_iv, 0x00, 16);
   memset(WPA_frame.rsc, 0x00, 8);
   memset(WPA_frame.key_identifier, 0x00, 8);
   memset(WPA_frame.key_mic, 0x00, 16);
```

```
   memset(WPA_frame.key_data_length, 0x00, 2);
   FN_EXIT(0, OK);
}
```

acx_rx_ieee802_11_frame

```
int acx_rx_ieee802_11_frame(wlandevice_t *priv, rxhostdescriptor_t *rxdesc)
{
   unsigned int ftype, fstype;
   const p80211_hdr_t *p80211_hdr;
   int result = NOT_OK;
/*************************************************************************/
    int payload_length;  /*variables necesarias para la decisión*/
    unsigned int payload_offset;
    wlan_llc_t *e_llc;
    wlan_snap_t *e_snap;
    p80211_hdr_t *w_hdr;
   u8 *e_payload;
    payload_length = MAC_CNT_RCVD(rxdesc->data) -
WLAN_HDR_A3_LEN;
    payload_offset = WLAN_HDR_A3_LEN;
/*************************************************************************/
   FN_ENTER;


   p80211_hdr = acx_get_p80211_hdr(priv, rxdesc);
/* printk("Rx_CONFIG_1 = %X\n",priv->rx_config_1 &
RX_CFG1_INCLUDE_ADDIT_HDR); */

   /* see IEEE 802.11-1999.pdf chapter 7 "MAC frame formats" */
   ftype = WLAN_GET_FC_FTYPE(ieee2host16(p80211_hdr->a3.fc));
   fstype = WLAN_GET_FC_FSTYPE(ieee2host16(p80211_hdr->a3.fc));

   switch (ftype) {
           /* check data frames first, for speed */
           case WLAN_FTYPE_DATA:
                   /* binary driver did ftype-1 to appease jump
                    * table layout */
                   if (fstype == WLAN_FSTYPE_DATAONLY)
                   {
/*************************************************************************/
                           w_hdr = (p80211_hdr_t *) &rxdesc->data->buf;
                           e_llc = (wlan_llc_t *) ((u8*) w_hdr +
payload_offset);
e_snap = (wlan_snap_t *) (((u8 *) e_llc) + sizeof(wlan_llc_t));
                           if (e_snap->type == ntohs(0x888e)) {
                              e_payload = ((u8 *) e_snap) +
sizeof(wlan_snap_t);
                              result = acx_4_way(rxdesc, priv, e_payload);
/*************************************************************************/

                           }
```

```
                        else {
                            if (ACX_MODE_3_MANAGED_AP == priv->macmode_joined) {
                                result = acx_process_data_frame_master(rxdesc, priv);
                            } else if (ISTATUS_4_ASSOCIATED == priv->status) {
                                result = acx_process_data_frame_client(rxdesc, priv);
                            }
                        /**/}/**/
                        } else switch (ftype) {
                                case WLAN_FSTYPE_DATA_CFACK:
                                case WLAN_FSTYPE_DATA_CFPOLL:
                                case WLAN_FSTYPE_DATA_CFACK_CFPOLL:
                                case WLAN_FSTYPE_CFPOLL:
                                case WLAN_FSTYPE_CFACK_CFPOLL:
                                /*  see above.
                                  acx_process_class_frame(rxdesc, priv, 3); */
                                        break;
                                case WLAN_FSTYPE_NULL:
                                        acx_process_NULL_frame(rxdesc, priv, 3);
                                /* FIXME: same here, see above */
                                case WLAN_FSTYPE_CFACK:
                                default:
                                        break;
                        }
                        break;
                case WLAN_FTYPE_MGMT:
                        result = acx_process_mgmt_frame(rxdesc, priv);
                        break;
                case WLAN_FTYPE_CTL:
                        if (fstype != WLAN_FSTYPE_PSPOLL)
                                result = NOT_OK;
                        else
                           result = OK;
                        /*  this call is irrelevant, since
                         *  acx_process_class_frame is a stub, so return
                         *  immediately instead.
                         * return acx_process_class_frame(rxdesc, priv, 3); */
                        break;
                default:
                        break;
        }
        FN_EXIT(1, result);
        return result;
}
```

Key_expansion

```
/*******************Archivo para la expansion de claves AES*******************/

/*funciones de creacion del arreglo de claves (ejecutadas una sola vez)*/
```

```
void key_expansion(void);
unsigned long sub_word(unsigned long temp);
unsigned long rot_word(unsigned long temp);
/*****************************************************************************/
/*****************************************************************************/
void key_expansion(void){
   FN_ENTER;
   const char Nk = 4, Nb = 4, Nr = 10;
   unsigned char i,j;
   unsigned long temp;
   unsigned long master_key[4];

   for(i = 0;i < 4; i++){
      master_key[i] = (unsigned long)DATA_enc_key[i*4];
      master_key[i] = (master_key[i] + ((unsigned long)DATA_enc_key[i*4 + 1] << 8));
      master_key[i] = (master_key[i] + ((unsigned long)DATA_enc_key[i*4 + 2] << 16));
      master_key[i] = (master_key[i] + ((unsigned long)DATA_enc_key[i*4 + 3] << 24));
   }
   for(i = 0;i < Nk;i++) key_array[i] = master_key[i];

   i = Nk;

   while (i < (Nb * (Nr + 1))){
            temp = key_array[i - 1];
            if ((i % Nk) == 0) {
                     j = (unsigned char) i/Nk;

                     temp = rot_word(temp);

                     temp = sub_word(temp) ^ rcon[j - 1];

            }
            else {
                     if ((Nk > 0x6) && ((i % Nk) == 0x4)) temp = sub_word(temp);
            }
            key_array[i] = key_array[i - Nk] ^ temp;
            i++;
   }
   FN_EXIT(0, OK);
}
/*****************************************************************************/
/*****************************************************************************/
unsigned long rot_word(unsigned long temp){
   unsigned char i;

   for(i = 0;i < 8;i++){
            if ((temp & 0x80000000) != 0) temp = ((temp << 1) ^ 0x1);
            else temp = (temp << 1);
   }
   return temp;
}
/*****************************************************************************/
/*****************************************************************************/
unsigned long sub_word(unsigned long temp){
   unsigned char temp1, temp2, temp3, temp4;
```

```
    unsigned long temp5, temp6, temp7, temp8;

    temp1 = (unsigned char) temp;
    temp1 = s_box[temp1];

    temp2 = (unsigned char) (temp >> 8);
    temp2 = s_box[temp2];

    temp3 = (unsigned char) (temp >> 16);
    temp3 = s_box[temp3];

    temp4 = (unsigned char) (temp >> 24);
    temp4 = s_box[temp4];

    temp5 = (unsigned long) temp1;
    temp6 = (unsigned long) temp2;
    temp7 = (unsigned long) temp3;
    temp8 = (unsigned long) temp4;

    temp5 = temp5 + (temp6 << 8) + (temp7 << 16) + (temp8 << 24);

    return temp5;
}
```

acx_4_way

```
static int acx_4_way(struct rxhostdescriptor *rxdesc, wlandevice_t *priv, u8 *data_in,
int length){
    u16 fc;
    struct txdescriptor *tx_desc;
    struct txhostdescriptor *payload;
    struct txhostdescriptor *header;
    p80211_hdr_t *w_hdr;
    u8 *pCurrPos;

    struct wlan_llc *e_llc;
    struct wlan_snap *e_snap;

    const u8 *a1 = NULL;
    const u8 *a2 = NULL;
    const u8 *a3 = NULL;

    short value16;

    FN_ENTER;
    unsigned long final, inicio;
    rdtscl(inicio);


    if ((tx_desc = acx_get_tx_desc(priv)) == NULL) {
    FN_EXIT(1, NOT_OK);
```

```
   return NOT_OK;
   }

   header = tx_desc->fixed_size.s.host_desc;          /*hostdescriptor for header*/
   payload = tx_desc->fixed_size.s.host_desc + 1;/*hostdescriptor for payload*/

   header->length = cpu_to_le16(WLAN_HDR_A3_LEN + sizeof(wlan_llc_t) +
sizeof(wlan_snap_t));

   e_llc = (wlan_llc_t *)(header->data + WLAN_HDR_A3_LEN);
   e_snap = (wlan_snap_t *)((u8 *)e_llc + sizeof(wlan_llc_t));

   e_llc->dsap = (u8)0xaa;
   e_llc->ssap = (u8)0xaa;
   e_llc->ctl = (u8)0x03;
   e_snap->type = htons(0x888e);
   e_snap->oui[0] = (u8)0x00;
   e_snap->oui[1] = (u8)0x00;
   e_snap->oui[2] = (u8)0x00;

   pCurrPos = (u8*)payload->data;

   data_in += 0x05; /*posiconamos en key information*/
   value16 = (u16)*data_in;
   data_in += 0x01;
   value16 = (u16)(*data_in << 8) + value16;
   value16 = ntohs(value16);
   data_in += 1 + 2; /*pocisionamos en replay counter recibido*/

   if (WPA_frame.replay_counter[7] == 0x05)
   WPA_frame.replay_counter[7] = 0x01;

   if ((memcmp(data_in, WPA_frame.replay_counter, 8) == 0) &&
   (value16 & WPA_PAIRWISE_KEY) && (value16 & WPA_KEY_ACK_FLAG)){
//              acxlog(L_BINSTD | L_ASSOC, "Primer mensaje del 4-way-handshake
recibido\n");
      data_in += 8;/*posicionamos en nonce_rcvd*/
      pCurrPos = four_way_2(rxdesc, pCurrPos, priv, data_in);
      WPA_frame.replay_counter[7]++;
   }
   else {if ((memcmp(data_in, WPA_frame.replay_counter, 8) == 00) && (value16 &
WPA_PAIRWISE_KEY) &&
   (value16 & WPA_KEY_ACK_FLAG) && (value16 & WPA_KEY_MIC_FLAG)) {
//    acxlog(L_BINSTD | L_ASSOC, "Tercer mensaje del 4-way handshake
recibido\n");
      data_in -= 5 + EAPOL_HDR_len;/*posicionamos en el principio de la trama*/
      if (pCurrPos == four_way_4(rxdesc, pCurrPos, data_in, length))
      WPA_frame.replay_counter[7]++;
      else {
            FN_EXIT(0, OK);
            return OK;
      }
   }
   }
```

```
    payload->length = cpu_to_le16((int)pCurrPos - (int)payload->data);

    acxlog(L_DATA, "la longitud de la trama es %d\n",payload->length);
    payload->data_offset = 0;
    header->data_offset = 0;

    tx_desc->total_length = cpu_to_le16((int)pCurrPos - (int)payload->data);

    w_hdr = (p80211_hdr_t *)header->data;

    fc = host2ieee16(WLAN_SET_FC_FTYPE(WLAN_FTYPE_DATA) |
                     WLAN_SET_FC_FSTYPE(WLAN_FSTYPE_DATAONLY));
    SET_BIT(fc, host2ieee16(WLAN_SET_FC_TODS(1)));

    a1 = priv->bssid;
    a2 = priv->dev_addr;
    a3 = priv->bssid;

    MAC_COPY(w_hdr->a3.a1, a1);
    MAC_COPY(w_hdr->a3.a2, a2);
    MAC_COPY(w_hdr->a3.a3, a3);

    w_hdr->a3.fc = fc;
    w_hdr->a3.dur = 0;
    w_hdr->a3.seq = 0;

    acx_dma_tx_data(priv, tx_desc); /*función que transmite los datos provista por el
driver*/

    rdtscl(final);
    if (WPA_frame.replay_counter[7] == 0x01) {
    key_expansion();
    acxlog(L_DATA, "Claves temporales generadas\n");
    }

    data_in += EAPOL_HDR_len + 1;
    if ((*data_in & 0x03) == 2) {
    /*retrocedemos para calcular el mic del proximo msj*/
    data_in -= EAPOL_HDR_len + 1;
    if (hmac_sha1(frame, length, EAPOL_mic_key, 16, mic))
       acxlog(L_DATA, "fallo en hmac_sha1 del mic\n");
    }
    else { if ((*data_in & 0x03) == 1) {
    data_in -= EAPOL_HDR_len + 1;
    if (hmac_md5(frame, length, EAPOL_mic_key, 16, mic))
       acxlog(L_DATA, "fallo en hmac_sha1 del mic\n");
    }
    }

    acxlog(L_BINSTD | L_ASSOC, "diferencia 4-way: %ld us\n",(final - inicio)/100);
    FN_EXIT(0, OK);
    return OK;
}
```

Todas las banderas que posee el campo *key_infromation* más definiciones de números útiles están definidos en un archivo llamado *defines.h* (se incluye en *4-way.h*) y se muestra a continuación:

/*defines.h*/

```
#define msj2_total_len 123
#define msj4_total_len 99
#define EAPOL_key_frame_sinIE_len 95
#define WPA_IE_len 24
#define EAPOL_HDR_len 4
#define mic_pos 77
#define data_len 76
#define pmk_len 32
#define ptk_len 48
#define nonce_len 32

/*banderas para key_inforation*/

#define WPA_SHA1_IN_MIC 0x0002
#define WPA_MD5_IN_MIC 0x0001
#define WPA_KEYTYPE_MASK 0x0007
#define WPA_PAIRWISE_KEY 0x0008
#define WPA_KEY_INDEX    0x0030
#define WPA_INSTALL_FLAG 0x0040
#define WPA_KEY_ACK_FLAG 0x0080
#define WPA_KEY_MIC_FLAG 0x0100
#define WPA_SECURE_FLAG  0x0200
#define WPA_ERROR_FLAG   0x0400
#define WPA_REQUEST_FLAG 0x0800
```

***four_way_2* :**

```
static u8 *four_way_2(struct rxhostdescriptor *rxdesc, u8 *pCurrPos, wlandevice_t
*priv, u8 *nonce_rcvd) {

   FN_ENTER;

   if (WPA_frame.replay_counter[7] == 0x01) {
   u8 data[77];
   p80211_hdr_t *w_hdr = (p80211_hdr_t *)&rxdesc->data->buf;
   if (memcmp(priv->dev_addr, w_hdr->a3.a3, 6) < 0) {
   if (memcmp(snonce, nonce_rcvd, nonce_len) < 0) {
//     acxlog(L_DATA, "dev_adrr < AP's addr, snonce < anonce \n");
       memcpy(data, priv->dev_addr, 6);
       memcpy(&data[6], w_hdr->a3.a2, 6);
       memcpy(&data[12], snonce, nonce_len);
       memcpy(&data[44], nonce_rcvd, nonce_len);
   }
   else {
//     acxlog(L_DATA, "dev_adrr < AP's addr, anonce < snonce \n");
```

```
        memcpy(data, priv->dev_addr, 6);
        memcpy(&data[6], w_hdr->a3.a2, 6);
        memcpy(&data[12], nonce_rcvd, nonce_len);
        memcpy(&data[44], snonce, nonce_len);
    }
    }
    else {
    if (memcmp(snonce, nonce_rcvd, nonce_len) < 0) {
//      acxlog(L_DATA, "AP's addr < dev_addr, snonce < anonce \n");
        memcpy(data, w_hdr->a3.a3, 6);
        memcpy(&data[6], priv->dev_addr, 6);
        memcpy(&data[12], snonce, nonce_len);
        memcpy(&data[44], nonce_rcvd, nonce_len);
    }
    else {
//      acxlog(L_DATA, "AP's addr < dev_addr, anonce < snonce \n");
        memcpy(data, w_hdr->a3.a3, 6);
        memcpy(&data[6], priv->dev_addr, 6);
        memcpy(&data[12], nonce_rcvd, nonce_len);
        memcpy(&data[44], snonce, nonce_len);
    }
    }

    if (PRF(pmk, pmk_len,"Pairwise key expansion", 22, data, data_len, ptk, ptk_len))
        acxlog(L_BINSTD | L_ASSOC, "fallo en PRF-48\n");
        if (memcpy(EAPOL_mic_key, &ptk[0], 16) != EAPOL_mic_key)
        acxlog(L_BINSTD | L_ASSOC, "fallo memcpy(EAPOL_mic_key, &ptk[0], 16)\n");
    if (memcpy(DATA_enc_key, &ptk[32], 16) != DATA_enc_key)
        acxlog(L_BINSTD | L_ASSOC, "fallo memcpy(DATA_enc_key, &ptk[32], 16)\n");

/***************************mensajes informativos*****************************/
/*    acxlog(L_BINSTD | L_ASSOC, "ptk = 0x ");
    for(i = 0;i < ptk_len; i++) acxlog(L_BINSTD | L_ASSOC, "%x", ptk[i]);
    acxlog(L_BINSTD | L_ASSOC, "\n");

    acxlog(L_BINSTD | L_ASSOC, "DATA_enc_key = 0x ");
    for(i = 0;i < 16; i++) acxlog(L_BINSTD | L_ASSOC, "%x", DATA_enc_key[i]);
    acxlog(L_BINSTD | L_ASSOC, "\n");*/

/*    acxlog(L_BINSTD | L_ASSOC, "EAPOL_mic_key = 0x ");
    for(i = 0;i < 16; i++) acxlog(L_BINSTD | L_ASSOC, "%x", EAPOL_mic_key[i]);
    acxlog(L_BINSTD | L_ASSOC, "\n");*/
/*****************************armado del mensaje 2****************************/
/*********************************EAPOL header**********************************/
    u8 *pCurrPos_init;
    pCurrPos_init = pCurrPos;
    memcpy(pCurrPos, WPA_frame.EAPOL_HDR, 2); pCurrPos += 2;
    *(u8 *)pCurrPos = (u8)((msj2_total_len - EAPOL_HDR_len) >> 8); pCurrPos += 1;
    *(u8 *)pCurrPos = (u8)(msj2_total_len - EAPOL_HDR_len); pCurrPos += 1;
    memset(&WPA_frame.EAPOL_HDR[3], msj2_total_len - EAPOL_HDR_len, 1);
/*****************************************************************************/
                                /* Key Descriptor */
    memcpy(pCurrPos, &WPA_frame.descriptor_type, 1); pCurrPos += 1;
/*****************************************************************************/
                            /*Key information field*/
```

```
   WPA_frame.key_information[0]          =          (u8)((WPA_SHA1_IN_MIC          |
WPA_PAIRWISE_KEY |
   WPA_KEY_MIC_FLAG) >> 8);
   WPA_frame.key_information[1]          =          (u8)(WPA_SHA1_IN_MIC          |
WPA_PAIRWISE_KEY |
   WPA_KEY_MIC_FLAG);
   memcpy(pCurrPos, WPA_frame.key_information, 2); pCurrPos += 2;
/*****************************************************************************/
   memcpy(pCurrPos, WPA_frame.key_length, 2); pCurrPos += 2;
/*****************************************************************************/
   memcpy(pCurrPos, WPA_frame.replay_counter, 8); pCurrPos += 8;
/*****************************************************************************/
   memcpy(pCurrPos, snonce, 32); pCurrPos += 32;
/*****************************************************************************/
   memcpy(pCurrPos, WPA_frame.EAPOL_key_iv, 16); pCurrPos += 16;
   memcpy(pCurrPos, WPA_frame.rsc, 8); pCurrPos += 8;
   memcpy(pCurrPos, WPA_frame.key_identifier, 8); pCurrPos += 8;
   memcpy(pCurrPos, WPA_frame.key_mic, 16); pCurrPos += 16;
/*****************************************************************************/
   WPA_frame.key_data_length[0] = (u8)(htons(0x1800) >> 8);
   WPA_frame.key_data_length[1] = (u8)(htons(0x1800));
   memcpy(pCurrPos, WPA_frame.key_data_length, 2); pCurrPos += 2;
/*************************aca va el key data = IE WPA*************************/
   memcpy(pCurrPos, WPA_IE, 24); pCurrPos += 24;
/*******************************calculo del mic*******************************/

   int len;
   len = cpu_to_le16((int)pCurrPos - (int)pCurrPos_init);

   /*nos fijamos si el mic se calcula con hmac_sha1 o hmac_md5*/

   nonce_rcvd -= 11; /*nos posicionamos en key information recibido*/

   if ((*nonce_rcvd & 0x03) == 2) {
   if (hmac_sha1(pCurrPos_init, len, EAPOL_mic_key, 16, &mic[0]))
      acxlog(L_DATA, "fallo en hmac_sha1 del mic\n");
      acxlog(L_DATA, "mic calculado con hmac_sha1\n");
   }
   else { if ((*nonce_rcvd & 0x03) == 1) {
   if (hmac_md5(pCurrPos_init, len, EAPOL_mic_key, 16, &mic[0]))
         acxlog(L_DATA, "fallo en hmac_md5 del mic\n");
            acxlog(L_DATA, "mic calculado con hmac_md5\n");
     }
     else {
            acxlog(L_DATA, "algoritmo del mic no identificado\n");
      }
   }
   }
   else {
   memcpy(pCurrPos, frame, msj2_total_len);
   pCurrPos += msj2_total_len;
   }

   pCurrPos -= 42; /*retrocedemos para ubicar el mic*/
   memcpy(pCurrPos, &mic[0], 16); /*posiciona el mic*/
```

```
    pCurrPos += 42;

    acxlog(L_DATA, "Enviando 2do mensaje del 4-way handshake\n");
    /*guardamos el mensaje transmitido con replay counter++ en el caso de
    que el punto de acceso no entienda el 2do mensaje enviado y ademas para
    acelerar el proceso en los subsiguientes 2dos mensajes dado que no es
    necesario hacer el desarrollo de PTK nuevamente porque se conoce el anonce*/
    pCurrPos -= msj2_total_len;
    memcpy(frame, pCurrPos, 16);
    frame[16] = (u8)(WPA_frame.replay_counter[7] + 1);
    pCurrPos += EAPOL_HDR_len + 5 + 8;/*17*/
    memcpy(&frame[EAPOL_HDR_len + 5 + 8], pCurrPos, 64);
    pCurrPos += 64 + 16;
    memset(&frame[EAPOL_HDR_len + 5 + 8 + 64], 0x00, 16);
    memcpy(&frame[EAPOL_HDR_len + 5 + 8 + 64 + 16],  pCurrPos, WPA_IE_len + 2);
    pCurrPos += WPA_IE_len + 2;

    FN_EXIT(0, OK);
    return pCurrPos;
}
```

four_way_4:

```
static u8 *four_way_4(struct rxhostdescriptor *rxdesc, u8 *pCurrPos, u8 *data_in, int
length) {

    FN_ENTER;
/************************verificacion del mic recibido************************/
    u8 mic_verify[20];
    memcpy(frame, data_in, EAPOL_HDR_len + mic_pos);
    memset(&frame[EAPOL_HDR_len + mic_pos], 0x00, 16);
    data_in += EAPOL_HDR_len + mic_pos + 16;
    memcpy(&frame[EAPOL_HDR_len + mic_pos + 16], data_in, WPA_IE_len + 2);
    data_in -= EAPOL_HDR_len + mic_pos + 16;

    data_in += EAPOL_HDR_len + 1; /*nos posicionamos en key information*/
    if ((*data_in & 0x03) == 2) {
    data_in -= EAPOL_HDR_len + 1;/*retrocedemos para calcular el mic*/
    if (hmac_sha1(frame, length, EAPOL_mic_key, 16, mic_verify));
        acxlog(L_DATA, "fallo en hmac_sha1 del mic\n");
        data_in += mic_pos + EAPOL_HDR_len; /*nos posicionamos en key_mic rcvd*/
        if (memcmp(data_in, mic_verify, 16)) {
            acxlog(L_DATA, "no matchearon los mics\n");
                return 0;
        }
    }
    else { if ((*data_in & 0x03) == 1) {
    data_in -= EAPOL_HDR_len + 1;
    if (hmac_md5(frame, length, EAPOL_mic_key, 16, mic_verify));
        acxlog(L_DATA, "fallo en hmac_sha1 del mic\n");
        data_in += mic_pos + EAPOL_HDR_len; /*nos posicionamos en key_mic rcvd*/
        if (memcmp(data_in, mic_verify, 16))
            acxlog(L_DATA, "no matchearon los mics\n");
                return 0;
```

```
    }
  }
  data_in -=   mic_pos + EAPOL_HDR_len;/*queda nuevamente apuntando al
principio*/
/*****************************armado del mensaje 2*****************************/
/*******************************EAPOL header*********************************/
  if (WPA_frame.replay_counter[7] == 0x02) {
  u8 *pCurrPos_init;
  pCurrPos_init = pCurrPos;
  memcpy(pCurrPos, WPA_frame.EAPOL_HDR, 2); pCurrPos += 2;
  *(u8 *)pCurrPos = (u8)((msj4_total_len - EAPOL_HDR_len) >> 8); pCurrPos += 1;
  *(u8 *)pCurrPos = (u8)(msj4_total_len - EAPOL_HDR_len); pCurrPos += 1;
  memset(&WPA_frame.EAPOL_HDR[3], msj2_total_len - EAPOL_HDR_len, 1);
/******************************************************************************/
                    /* Key Descriptor */
  memcpy(pCurrPos, &WPA_frame.descriptor_type, 1); pCurrPos += 1;
/******************************************************************************/
                      /*Key information field*/
  WPA_frame.key_information[0]          =          (u8)((WPA_SHA1_IN_MIC          |
WPA_PAIRWISE_KEY | WPA_KEY_MIC_FLAG | WPA_INSTALL_FLAG) >> 8);
  WPA_frame.key_information[1]          =          (u8)(WPA_SHA1_IN_MIC          |
WPA_PAIRWISE_KEY | WPA_KEY_MIC_FLAG | WPA_INSTALL_FLAG);
  memcpy(pCurrPos, WPA_frame.key_information, 2); pCurrPos += 2;
/******************************************************************************/
  memcpy(pCurrPos, WPA_frame.key_length, 2); pCurrPos += 2;
/******************************************************************************/
  memcpy(pCurrPos, WPA_frame.replay_counter, 8); pCurrPos += 8;
/******************************************************************************/
  memcpy(pCurrPos, snonce, 32); pCurrPos += 32;
/******************************************************************************/
  memcpy(pCurrPos, WPA_frame.EAPOL_key_iv, 16); pCurrPos += 16;
/*************************extraemos el rsc recibido**************************/
  data_in += EAPOL_HDR_len + 5 + 8 + 32 + 16;
  memcpy(WPA_frame.rsc, data_in, 8);
  data_in -= EAPOL_HDR_len + 5 + 8 + 32 + 16;
  memcpy(pCurrPos, WPA_frame.rsc, 8); pCurrPos += 8;
  memcpy(pCurrPos, WPA_frame.key_identifier, 8); pCurrPos += 8;
/******************************************************************************/
  memcpy(pCurrPos, WPA_frame.key_mic, 16); pCurrPos += 16;
/******************************************************************************/
  WPA_frame.key_data_length[0] = (u8)(0x00);
  WPA_frame.key_data_length[1] = (u8)(0x00);
  memcpy(pCurrPos, WPA_frame.key_data_length, 2); pCurrPos += 2;
  pCurrPos -= 99;
/*******************************calculo del mic********************************/
  int len;

  len = cpu_to_le16((int)pCurrPos - (int)pCurrPos_init);

  /*nos fijamos si el mic se calcula con hmac_sha1 o hmac_md5*/

  data_in += EAPOL_HDR_len + 1 ; /*nos posicionamos en key information*/
  if ((*data_in & 0x03) == 2) {
  if (hmac_sha1(pCurrPos_init, len, EAPOL_mic_key, 16, mic));
    acxlog(L_DATA, "fallo en hmac_sha1 del mic\n");
```

```
}
else { if ((*data_in & 0x03) == 1) {
if (hmac_md5(pCurrPos_init, len, EAPOL_mic_key, 16, mic));
   acxlog(L_DATA, "fallo en hmac_sha1 del mic\n");
   }
}
}
else {
memcpy(pCurrPos, frame, 99);
pCurrPos += 99;
}

pCurrPos += EAPOL_HDR_len + mic_pos; /*adelantamos para ubicar el mic*/
memcpy(pCurrPos, mic, 16); /*posiciona el mic*/
pCurrPos -= EAPOL_HDR_len + mic_pos; /*queda nuevamente en el principio*/

acxlog(L_DATA, "Enviando 4to mensaje del 4-way handshake\n");

/*guardamos el mensaje transmitido con replay counter++ en el caso de
que el punto de acceso no entienda el cuarto mensaje enviado y ademas para
acelerar el proceso en los subsiguientes cuartos mensajes */

memcpy(frame, pCurrPos, EAPOL_HDR_len + 5 + 7);/*16*/
frame[16] = (u8)(WPA_frame.replay_counter[7] + 1);
pCurrPos += EAPOL_HDR_len + 5 +8;/*17*/
memcpy(&frame[17], pCurrPos, 64); /*copia hasta key mic*/
memset(&frame[17 + 64], 0x00, 2); /*key_mic + key data length*/
pCurrPos += 64 + 16 + 2; /*queda en el final 99*/

FN_EXIT(0, OK);
return pCurrPos;
}
```

ccmp_conv.h

```
/**********************Includes del archivo acx100_conv.c************************/
#include <aes.h>
#include <ccmp_des_enc.h>
#include <ccmp_enc.h>
```

aes.h

```
/**************************Variables globales*********************************/
/*****************************************************************************/
unsigned char state[4][4];
unsigned char inv_s_box[] = {0x52, 0x09, 0x6a, 0xd5, 0x30, 0x36, 0xa5, 0x38, 0xbf,
0x40, 0xa3, 0x9e, 0x81, 0xf3, 0xd7, 0xfb,
                    0x7c, 0xe3, 0x39, 0x82, 0x9b, 0x2f, 0xff, 0x87, 0x34, 0x8e, 0x43,
0x44, 0xc4, 0xde, 0xe9, 0xcb,
                    0x54, 0x7b, 0x94, 0x32, 0xa6, 0xc2, 0x23, 0x3d, 0xee, 0x4c, 0x95,
0x0b, 0x42, 0xfa, 0xc3, 0x4e,
                    0x08, 0x2e, 0xa1, 0x66, 0x28, 0xd9, 0x24, 0xb2, 0x76, 0x5b, 0xa2,
0x49, 0x6d, 0x8b, 0xd1, 0x25,
```

```
                0x72, 0xf8, 0xf6, 0x64, 0x86, 0x68, 0x98, 0x16, 0xd4, 0xa4, 0x5c,
0xcc, 0x5d, 0x65, 0xb6, 0x92,
                0x6c, 0x70, 0x48, 0x50, 0xfd, 0xed, 0xb9, 0xda, 0x5e, 0x15, 0x46,
0x57, 0xa7, 0x8d, 0x9d, 0x84,
                0x90, 0xd8, 0xab, 0x00, 0x8c, 0xbc, 0xd3, 0x0a, 0xf7, 0xe4, 0x58,
0x05, 0xb8, 0xb3, 0x45, 0x06,
                0xd0, 0x2c, 0x1e, 0x8f, 0xca, 0x3f, 0x0f, 0x02, 0xc1, 0xaf, 0xbd,
0x03, 0x01, 0x13, 0x8a, 0x6b,
                0x3a, 0x91, 0x11, 0x41, 0x4f, 0x67, 0xdc, 0xea, 0x97, 0xf2, 0xcf,
0xce, 0xf0, 0xb4, 0xe6, 0x73,
                0x96, 0xac, 0x74, 0x22, 0xe7, 0xad, 0x35, 0x85, 0xe2, 0xf9, 0x37,
0xe8, 0x1c, 0x75, 0xdf, 0x6e,
                0x47, 0xf1, 0x1a, 0x71, 0x1d, 0x29, 0xc5, 0x89, 0x6f, 0xb7, 0x62,
0x0e, 0xaa, 0x18, 0xbe, 0x1b,
                0xfc, 0x56, 0x3e, 0x4b, 0xc6, 0xd2, 0x79, 0x20, 0x9a, 0xdb, 0xc0,
0xfe, 0x78, 0xcd, 0x5a, 0xf4,
                0x1f, 0xdd, 0xa8, 0x33, 0x88, 0x07, 0xc7, 0x31, 0xb1, 0x12, 0x10,
0x59, 0x27, 0x80, 0xec, 0x5f,
                0x60, 0x51, 0x7f, 0xa9, 0x19, 0xb5, 0x4a, 0x0d, 0x2d, 0xe5, 0x7a,
0x9f, 0x93, 0xc9, 0x9c, 0xef,
                0xa0, 0xe0, 0x3b, 0x4d, 0xae, 0x2a, 0xf5, 0xb0, 0xc8, 0xeb, 0xbb,
0x3c, 0x83, 0x53, 0x99, 0x61,
                0x17, 0x2b, 0x04, 0x7e, 0xba, 0x77, 0xd6, 0x26, 0xe1, 0x69, 0x14,
0x63, 0x55, 0x21, 0x0c, 0x7d};
unsigned char s_box[] = {0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01,
0x67, 0x2b, 0xfe, 0xd7, 0xab, 0x76,
                0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2,
0xaf, 0x9c, 0xa4, 0x72, 0xc0,
                0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5,
0xf1, 0x71, 0xd8, 0x31, 0x15,
                0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12, 0x80,
0xe2, 0xeb, 0x27, 0xb2, 0x75,
                0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b, 0xd6,
0xb3, 0x29, 0xe3, 0x2f, 0x84,
                0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb, 0xbe,
0x39, 0x4a, 0x4c, 0x58, 0xcf,
                0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9, 0x02,
0x7f, 0x50, 0x3c, 0x9f, 0xa8,
                0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6, 0xda,
0x21, 0x10, 0xff, 0xf3, 0xd2,
                0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7, 0x7e,
0x3d, 0x64, 0x5d, 0x19, 0x73,
                0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8,
0x14, 0xde, 0x5e, 0x0b, 0xdb,
                0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3, 0xac,
0x62, 0x91, 0x95, 0xe4, 0x79,
                0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4,
0xea, 0x65, 0x7a, 0xae, 0x08,
                0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd, 0x74,
0x1f, 0x4b, 0xbd, 0x8b, 0x8a,
                0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35, 0x57,
0xb9, 0x86, 0xc1, 0x1d, 0x9e,
                0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e, 0x87,
0xe9, 0xce, 0x55, 0x28, 0xdf,
```

```
                    0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d,
0x0f, 0xb0, 0x54, 0xbb, 0x16};
        unsigned long key_array[44];
        unsigned long rcon[] = {0x01000000, 0x02000000, 0x04000000, 0x08000000,
0x10000000,
                    0x20000000, 0x40000000, 0x80000000, 0x1b000000, 0x36000000};

        /*************************************************************************
****/
        /*****************************Prototipos**********************************
***/

        void aes_enc(u8 *data_to_tx, u8 *aes_out);
        void aes_desenc(u8 *data_to_rx, int len, u8 *aes_out);

/*funciones esenciales para el algoritmo de encriptación, llamadas cada vez q se desea
encriptar datos*/

        void sub_bytes(void);
        void shift_rows(void);
        void mix_columns(void);
        unsigned char mult(unsigned char n, unsigned char m);
        void add_round_key(unsigned char round);


/*funciones esenciales para el algoritmo de desencritación, llamadas cada vez q se
desea desencriptar datos*/


        void add_round_key(unsigned char round);
        void inv_mix_columns(void);
        inline unsigned char mult2(unsigned char m, unsigned char n);
        void inv_sub_bytes(void);
        void inv_shift_rows(void);


        /*funciones utilizadas por los dos procesos*/


        void copy_data_to_state(u8 *data);
        void copy_state_to_data(u8 *data);

        /*************************************************************************
****/
        /*********************************Definiciones*****************************
***/
        void copy_data_to_state(u8 *data){
                unsigned char i,j;

                for(j=0;j<4;j++){
                        for(i=0;i<4;i++) {
                                state[i][j] = data[4*j + i];
                        }
                }
        }
```

```
    /*****************************************************************************
****/
    /*****************************************************************************
****/
    void copy_state_to_data(u8 *data){
        unsigned char i,j;

        for(j=0;j<4;j++){
            for(i=0;i<4;i++) {
                data[4*j + i] = state[i][j];
            }
        }
    }
    /*****************************************************************************
****/
    /*****************************************************************************
****/
    void sub_bytes(void){
        unsigned char i,j;

        for(i=0;i<4;i++){

            for(j=0;j<4;j++) {
                state[i][j] = s_box[state[i][j]];
            }
        }
    }
    /*****************************************************************************
****/
    /*****************************************************************************
****/
    void shift_rows(void){
        unsigned char i,j,a[3];

        for(i = 1;i < 4;i++){
            j = 0;
            while (j < i) {
                a[j] = state[i][j];
                j++;
            }
            j = 0;
            while (j < (4 - i)){
                state[i][j] = state[i][j+i];
                j++;
            }
            j = 0;
            while (j < i){
                state[i][3 - j] = a[i - 1 - j];
                j++;
            }
        }
    }
    /*****************************************************************************
****/
```

```
	/*************************************************************************
****/
	void mix_columns(void){
		unsigned char j, vect[4];

		for(j = 0;j < 4;j++){
			vect[0] = (mult(0x2,state[0][j])) ^ (mult(0x3,state[1][j])) ^ state[2][j] ^
state[3][j];
			vect[1] = state[0][j] ^ (mult(0x2,state[1][j])) ^ (mult(0x3,state[2][j])) ^
state[3][j];
			vect[2] = state[0][j] ^ state[1][j] ^ (mult(0x2,state[2][j])) ^
(mult(0x3,state[3][j]));
			vect[3] = (mult(0x3,state[0][j])) ^ state[1][j] ^ state[2][j] ^
(mult(0x2,state[3][j]));
			state[0][j] = vect[0];
			state[1][j] = vect[1];
			state[2][j] = vect[2];
			state[3][j] = vect[3];
		}
	}
	/*************************************************************************
****/
	/*************************************************************************
****/
	unsigned char mult(unsigned char m, unsigned char n){
		unsigned int s;

		n = (unsigned int) n;
		if (m == 0x2){ /*multiplicación x 2*/
			s = n << 1;
			/*si luego de la mult x 2 supera 255 entonces hay aplicar módulo x8
+ x4 + x3 + x + 1*/
			if (s > 255) s = (s & 0xff) ^ (0x1b);
		}
		else{
			s = (n << 1) ^ n; /*multiplicación x 3*/
			/*si luego de la mult x 3 supera 255 entonces hay que aplicar módulo
x8 + x4 + x3 + x + 1*/
			if (s > 255) s = (s & 0xff) ^ (0x1b); /*descarte de b8 comenzando desde
b0*/
		}
		s = (unsigned char) s;
		return s;
	}
	/*************************************************************************
****/
	/*************************************************************************
****/
	void add_round_key(unsigned char round){
		unsigned char i,j, key[4][4];

		for(i = 0;i < 4;i++){
			key[3][i] = (unsigned char) key_array[(4 * round) + i];
			key[2][i] = (unsigned char) (key_array[(4 * round) + i] >> 8);
			key[1][i] = (unsigned char) (key_array[(4 * round) + i] >> 16);
```

```
                key[0][i] = (unsigned char) (key_array[(4 * round) + i] >> 24);
        }
        for(i = 0;i < 4;i++){
                for(j = 0;j < 4;j++) state[i][j] = state[i][j] ^ key[i][j];
        }
    }
    /*************************************************************************
****/
    /*************************************************************************
****/
    void inv_shift_rows(void){
        unsigned char i,j,a[3];
        for(i = 1;i < 4;i++){
                j = 3;
                while (j > (3 - i)) {
                        a[3 - j] = state[i][7 - j - i];
                        j--;
                }
                j = 3;
                while (j > (i - 1)){
                        state[i][j] = state[i][j - i];
                        j--;
                }
                j = 3;
                while (j > (3 - i)){
                        state[i][3 - j] = a[3 - j];
                        j--;
                }
        }
    }
    /*************************************************************************
****/
    /*************************************************************************
****/
    void inv_sub_bytes(void){
        unsigned char i,j;

        for(i=0;i<4;i++){
                for(j=0;j<4;j++) {
                        state[i][j] = inv_s_box[state[i][j]];
                }
        }
    }
    /*************************************************************************
****/
    /*************************************************************************
****/
    inline unsigned char mult2(unsigned char m, unsigned char n){
        unsigned int s = 0;

        n = (unsigned int) n;
        if (m & 0x1) s = n;
        if (m & 0x2) s = (n << 1) ^ s;
        if (m & 0x4) s = (n << 2) ^ s;
        if (m & 0x8) s = (n << 3) ^ s;
```

```
   /*si luego de la mult supera 255 entonces hay aplicar módulo (x8 + x4 + x3 + x + 1)
(polinomio irreducible) */
            if (s > 1023) s = (s & 0x3ff) ^ (0x6c); /*0x1b << 2, descarte de b10 comenzado
con b0 */
            if (s > 511) s = (s & 0x1ff) ^ (0x36); /*0x1b << 1, descarte de b9 */
            if (s > 255) s = (s & 0xff) ^ (0x1b); /*descarte de b8*/
            s = (unsigned char) s;
            return s;
      }
      /*************************************************************************
****/
      /*************************************************************************
****/
      void inv_mix_columns(void){
            unsigned char j, vect[4];

            for(j = 0;j < 4;j++){
            vect[0]     =     (mult2(0xe,state[0][j]))     ^     (mult2(0xb,state[1][j]))     ^
(mult2(0xd,state[2][j])) ^ (mult2(0x9,state[3][j]));
            vect[1]     =     (mult2(0x9,state[0][j]))     ^     (mult2(0xe,state[1][j]))     ^
(mult2(0xb,state[2][j])) ^ (mult2(0xd,state[3][j]));
            vect[2]     =     (mult2(0xd,state[0][j]))     ^     (mult2(0x9,state[1][j]))     ^
(mult2(0xe,state[2][j])) ^ (mult2(0xb,state[3][j]));
            vect[3]     =     (mult2(0xb,state[0][j]))     ^     (mult2(0xd,state[1][j]))     ^
(mult2(0x9,state[2][j])) ^ (mult2(0xe,state[3][j]));
                        state[0][j] = vect[0];
                        state[1][j] = vect[1];
                        state[2][j] = vect[2];
                        state[3][j] = vect[3];
            }
      }
      /*************************************************************************
****/
      /*******************************encriptacion*******************************
***/

      void aes_enc(u8 *data_to_tx, u8 *aes_out){

            unsigned char k;

      /*-------------------------comienza el algoritmo----------------------------*/

            copy_data_to_state(&data_to_tx[0]);

            k = 0;

            add_round_key(k);

            for(k = 1;k < 11;k++){

                  sub_bytes();

                  shift_rows();

                  if (k != 10) mix_columns();
```

```
                add_round_key(k);

            }

            copy_state_to_data(aes_out);


        }
```

acx_ether_to_txdesc

```
int acx_ether_to_txdesc(wlandevice_t *priv,
                        struct txdescriptor *tx_desc,
                        const struct sk_buff *skb)
{
   unsigned short proto;            /* protocol type or data length, depending on whether
DIX or 802.3 ethernet format */
   u16 fc;
   struct txhostdescriptor *header;
   struct txhostdescriptor *payload;
   p80211_hdr_t * w_hdr;

   wlan_ethhdr_t *e_hdr;
   struct wlan_llc *e_llc;
   struct wlan_snap *e_snap;

   const u8 *a1 = NULL;
   const u8 *a2 = NULL;
   const u8 *a3 = NULL;

   FN_ENTER;

   if (unlikely(0 == skb->len)) {
            acxlog(L_DEBUG, "zero-length skb!\n");
            FN_EXIT(1, NOT_OK);
            return NOT_OK;
   }

   header = tx_desc->fixed_size.s.host_desc;
   if ((unsigned long)0xffffffff == (unsigned long)header) /* FIXME: happens on card
eject; better method? */
            return NOT_OK;
   payload = tx_desc->fixed_size.s.host_desc + 1;
   e_hdr = (wlan_ethhdr_t *)skb->data;

   /* step 1: classify ether frame, DIX or 802.3? */
   proto = ntohs(e_hdr->type);
   if (proto <= 1500) {
         acxlog(L_DEBUG, "tx: 802.3 len: %d\n", skb->len);
            /* codes <= 1500 reserved for 802.3 lengths */
            /* it's 802.3, pass ether payload unchanged,  */

            /* trim off ethernet header and copy payload to tx_desc */
```

```
            payload->length = cpu_to_le16(proto);

    } else {
            /* it's DIXII, time for some conversion */
            /* Create 802.11 packet. Header also contains llc and snap. */

            acxlog(L_DEBUG, "tx: DIXII len: %d\n", skb->len);

            /* size of header is 802.11 header + llc + snap */
            header->length = cpu_to_le16(WLAN_HDR_A3_LEN + sizeof(wlan_llc_t) +
sizeof(wlan_snap_t));
            /* llc is located behind the 802.11 header */
            e_llc = (wlan_llc_t*)(header->data + WLAN_HDR_A3_LEN);
            /* snap is located behind the llc */
            e_snap = (wlan_snap_t*)((u8*)e_llc + sizeof(wlan_llc_t));

            /* setup the LLC header */
            e_llc->dsap = (u8)0xaa;          /* SNAP, see IEEE 802 */
            e_llc->ssap = (u8)0xaa;
            e_llc->ctl = (u8)0x03;

            /* setup the SNAP header */
            e_snap->type = htons(proto);
            if (0 != acx_stt_findproto(proto)) {
                    /* memcpy(e_snap->oui, oui_8021h, WLAN_IEEE_OUI_LEN); */
                    COPY_OUI(e_snap->oui, oui_8021h);
            } else {
                    /* memcpy(e_snap->oui, oui_rfc1042, WLAN_IEEE_OUI_LEN); */
                    COPY_OUI(e_snap->oui, oui_rfc1042);
            }
            /* trim off ethernet header and copy payload to tx_desc */
            payload->length = cpu_to_le16(skb->len - sizeof(wlan_ethhdr_t));
    }
/*******************************************************************************
****/
    payload->length = enc((struct sk_buff *)skb, payload->data, priv);
/*******************************************************************************
****/
// memcpy(payload->data, skb->data + sizeof(wlan_ethhdr_t), le16_to_cpu(payload-
>length));
    payload->data_offset = 0;
    header->data_offset = 0;

    /* calculate total tx_desc length */
    tx_desc->total_length       =       cpu_to_le16(le16_to_cpu(payload->length)       +
le16_to_cpu(header->length));

    /* Set up the 802.11 header */
    w_hdr = (p80211_hdr_t*)header->data;

    /* It's a data frame */
    fc = host2ieee16(WLAN_SET_FC_FTYPE(WLAN_FTYPE_DATA) |
                     WLAN_SET_FC_FSTYPE(WLAN_FSTYPE_DATAONLY));
```

```
    switch (priv->macmode_joined) {
    case ACX_MODE_0_IBSS_ADHOC:
    case ACX_MODE_1_UNUSED:
            a1 = e_hdr->daddr;
            a2 = priv->dev_addr;
            a3 = priv->bssid;
            break;
    case ACX_MODE_2_MANAGED_STA:
            SET_BIT(fc, host2ieee16(WLAN_SET_FC_TODS(1)));
            a1 = priv->bssid;
            a2 = priv->dev_addr;
            a3 = e_hdr->daddr;
            break;
    case ACX_MODE_3_MANAGED_AP:
            SET_BIT(fc, host2ieee16(WLAN_SET_FC_FROMDS(1)));
            a1 = e_hdr->daddr;
            a2 = priv->bssid;
            a3 = e_hdr->saddr;
            break;
    default:                        /* fall through */
            acxlog(L_STD, "Error: Converting eth to wlan in unknown mode\n");
            goto fail;
    }
    MAC_COPY(w_hdr->a3.a1, a1);
    MAC_COPY(w_hdr->a3.a2, a2);
    MAC_COPY(w_hdr->a3.a3, a3);

    if ((u8)0 != priv->wep_enabled)
            SET_BIT(fc, host2ieee16(WLAN_SET_FC_ISWEP(1)));

    w_hdr->a3.fc = fc;
    w_hdr->a3.dur = 0;
    w_hdr->a3.seq = 0;

    /* the "<6>" output is from the KERN_INFO channel value */
    /* Can be used to debug conversion process */
#if DEBUG_CONVERT
    acxlog(L_DATA, "Original eth frame [%d]: ", skb->len);
    for (i = 0; i < skb->len; i++)
            acxlog(L_DATA, "%02x ", ((u8 *) skb->data)[i]);
    acxlog(L_DATA, "\n");

    acxlog(L_DATA, "802.11 header [%d]: ", le16_to_cpu(header->length));
    for (i = 0; i < le16_to_cpu(header->length); i++)
            acxlog(L_DATA, "%02x ", ((u8 *) header->data)[i]);
    acxlog(L_DATA, "\n");

    acxlog(L_DATA, "802.11 payload [%d]: ", le16_to_cpu(payload->length));
    for (i = 0; i < le16_to_cpu(payload->length); i++)
            acxlog(L_DATA, "%02x ", ((u8 *) payload->data)[i]);
    acxlog(L_DATA, "\n");
#endif

fail:
    FN_EXIT(0, OK);
```

```
    return OK;
}
```

acx_rxdesc_to_ether

```
 struct sk_buff *acx_rxdesc_to_ether(wlandevice_t *priv, const struct
            rxhostdescriptor *rx_desc)
{
    const u8 *daddr = NULL;
    const u8 *saddr = NULL;
    wlan_ethhdr_t *e_hdr;
    wlan_llc_t *e_llc;
    wlan_snap_t *e_snap;
    u8 *e_payload;
    p80211_hdr_t *w_hdr;
    int buflen;
    u16 fc;
    int payload_length;
    unsigned int payload_offset;
/* int i; */

    struct sk_buff *skb = NULL;

    FN_ENTER;

    payload_length = MAC_CNT_RCVD(rx_desc->data) - WLAN_HDR_A3_LEN;
    payload_offset = WLAN_HDR_A3_LEN;

    w_hdr = (p80211_hdr_t*)&rx_desc->data->buf;

    /* check if additional header is included */
    if (0 != (priv->rx_config_1 & RX_CFG1_INCLUDE_ADDIT_HDR)) {
            /* Mmm, strange, when receiving a packet, 4 bytes precede the packet. Is it
the CRC ? */
            w_hdr = (p80211_hdr_t*)(((u8*)w_hdr) + WLAN_CRC_LEN);
            payload_length -= WLAN_CRC_LEN;
    }
/*************************************************************************/
    payload_length = des_enc(w_hdr, payload_length, priv);
/*************************************************************************/
    /* setup some vars for convenience */
    fc = ieee2host16(w_hdr->a3.fc);
    if ((0 == WLAN_GET_FC_TODS(fc)) && (0 == WLAN_GET_FC_FROMDS(fc))) {
            daddr = w_hdr->a3.a1;
            saddr = w_hdr->a3.a2;
    }       else       if       (0       ==       (WLAN_GET_FC_TODS(fc))       &&       (1       ==
WLAN_GET_FC_FROMDS(fc))) {
            daddr = w_hdr->a3.a1;
            saddr = w_hdr->a3.a3;
    }       else       if       ((1       ==       WLAN_GET_FC_TODS(fc))       &&       (0       ==
WLAN_GET_FC_FROMDS(fc))) {
            daddr = w_hdr->a3.a3;
            saddr = w_hdr->a3.a2;
    } else {
```

```
            payload_offset = WLAN_HDR_A4_LEN;
            payload_length -= ( WLAN_HDR_A4_LEN - WLAN_HDR_A3_LEN );
            if (0 > payload_length) {
                    acxlog(L_STD, "A4 frame too short!\n");
                    return NULL;
            }
            daddr = w_hdr->a4.a3;
            saddr = w_hdr->a4.a4;
    }

    if (0 != WLAN_GET_FC_ISWEP(fc)) {
            /* chop off the IV+ICV WEP header and footer */
            acxlog(L_DATA | L_DEBUG, "rx: it's a WEP packet, chopping off IV and
ICV.\n");
            payload_length -= 8;
            payload_offset += 4;
    }

    e_hdr = (wlan_ethhdr_t *) ((u8*) w_hdr + payload_offset);

    e_llc = (wlan_llc_t *) ((u8*) w_hdr + payload_offset);
    e_snap = (wlan_snap_t *) (((u8 *) e_llc) + sizeof(wlan_llc_t));
    e_payload = ((u8 *) e_snap) + sizeof(wlan_snap_t);

    acxlog(L_DATA, "rx: payload_offset %i, payload_length %i\n", payload_offset,
payload_length);
    acxlog(L_XFER | L_DATA,
        "rx: frame info: llc.dsap %X, llc.ssap %X, llc.ctl %X, snap.oui %X%X%X,
snap.type %X\n",
        e_llc->dsap, e_llc->ssap,
        e_llc->ctl, e_snap->oui[0],
        e_snap->oui[1], e_snap->oui[2],
        e_snap->type);

    /* Test for the various encodings */
    if ( (payload_length >= sizeof(wlan_ethhdr_t)) &&
        ((e_llc->dsap != (u8)0xaa) || (e_llc->ssap != (u8)0xaa)) &&
        ((0 == memcmp(daddr, e_hdr->daddr, ETH_ALEN)) ||
         (0 == memcmp(saddr, e_hdr->saddr, ETH_ALEN)))) {
            acxlog(L_DEBUG | L_DATA, "rx: 802.3 ENCAP len: %d\n",
payload_length);
            /* 802.3 Encapsulated */
            /* Test for an overlength frame */

            if (unlikely(payload_length > WLAN_MAX_ETHFRM_LEN)) {
                    /* A bogus length ethfrm has been encap'd. */
                    /* Is someone trying an oflow attack? */
                    acxlog(L_STD, "rx: ENCAP frame too large (%d > %d)\n",
                            payload_length, WLAN_MAX_ETHFRM_LEN);
                    FN_EXIT(1, (int)NULL);
                    return NULL;
            }

            /* allocate space and setup host buffer */
            buflen = payload_length;
```

```
                /* FIXME: implement skb ring buffer similar to
                 * xircom_tulip_cb.c? */
                skb = dev_alloc_skb(buflen + 2);          /* +2 is attempt to align IP header */
                if (unlikely(NULL == skb)) {
                        acxlog(L_STD, "rx: FAILED to allocate skb\n");
                        FN_EXIT(1, (int)NULL);
                        return NULL;
                }
                skb_reserve(skb, 2);
                skb_put(skb, buflen);               /* make room */

                /* now copy the data from the 80211 frame */
                memcpy(skb->data, e_hdr, payload_length);    /* copy the data */

        } else if ((payload_length >= sizeof(wlan_llc_t) + sizeof(wlan_snap_t)) &&
                   (e_llc->dsap == (u8)0xaa) &&
                   (e_llc->ssap == (u8)0xaa) &&
                   (e_llc->ctl == (u8)0x03) &&
                   (((COMPARE_OUI( e_snap->oui, oui_rfc1042)==0) &&
/*                  (ethconv == WLAN_ETHCONV_8021h) &&  */
                   (0 != acx_stt_findproto(ieee2host16(e_snap->type)))) ||
                   (0 != COMPARE_OUI( e_snap->oui, oui_rfc1042))))
        {
                acxlog(L_DEBUG | L_DATA, "rx: SNAP+RFC1042 len: %d\n",
payload_length);
                /* it's a SNAP + RFC1042 frame && protocol is in STT */
                /* build 802.3 + RFC1042 */

                /* Test for an overlength frame */
                if (unlikely(payload_length + WLAN_ETHHDR_LEN >
WLAN_MAX_ETHFRM_LEN)) {
                        /* A bogus length ethfrm has been sent. */
                        /* Is someone trying an oflow attack? */
                        acxlog(L_STD, "rx: SNAP frame too large (%d > %d)\n",
                                payload_length, WLAN_MAX_ETHFRM_LEN);
                        FN_EXIT(1, (int)NULL);
                        return NULL;
                }

                /* allocate space and setup host buffer */
                buflen = payload_length + WLAN_ETHHDR_LEN;
                skb = dev_alloc_skb(buflen + 2);          /* +2 is attempt to align IP header */
                if (unlikely(NULL == skb)) {
                        acxlog(L_STD, "rx: FAILED to allocate skb\n");
                        FN_EXIT(1, (int)NULL);
                        return NULL;
                }
                skb_reserve(skb, 2);
                skb_put(skb, buflen);               /* make room */

                /* create 802.3 header */
                e_hdr = (wlan_ethhdr_t *) skb->data;
                MAC_COPY(e_hdr->daddr, daddr);
                MAC_COPY(e_hdr->saddr, saddr);
                e_hdr->type = htons(payload_length);
```

```
            /* Now copy the data from the 80211 frame.
               Make room in front for the eth header, and keep the
               llc and snap from the 802.11 payload */
            memcpy(skb->data + WLAN_ETHHDR_LEN,
                 e_llc,
                 payload_length);

  }  else if ((payload_length >= sizeof(wlan_llc_t) + sizeof(wlan_snap_t)) &&
               (e_llc->dsap == (u8)0xaa) &&
               (e_llc->ssap == (u8)0xaa) &&
               (e_llc->ctl == (u8)0x03) ) {
            acxlog(L_DEBUG | L_DATA, "rx: 802.1h/RFC1042 len: %d\n",
payload_length);
            /* it's an 802.1h frame || (an RFC1042 && protocol is not in STT) */
            /* build a DIXII + RFC894 */

            /* Test for an overlength frame */
            if (unlikely(payload_length - sizeof(wlan_llc_t) - sizeof(wlan_snap_t) +
WLAN_ETHHDR_LEN > WLAN_MAX_ETHFRM_LEN)) {
                  /* A bogus length ethfrm has been sent. */
                  /* Is someone trying an oflow attack? */
                  acxlog(L_STD, "rx: DIXII frame too large (%d > %d)\n",
                              payload_length - sizeof(wlan_llc_t) -
sizeof(wlan_snap_t),
                              WLAN_MAX_ETHFRM_LEN -
WLAN_ETHHDR_LEN);
                  FN_EXIT(1, (int)NULL);
                  return NULL;
            }

            /* allocate space and setup host buffer */
            buflen = payload_length + WLAN_ETHHDR_LEN - sizeof(wlan_llc_t) -
sizeof(wlan_snap_t);
            skb = dev_alloc_skb(buflen + 2);          /* +2 is attempt to align IP header */
            if (unlikely(NULL == skb)) {
                  acxlog(L_STD, "rx: FAILED to allocate skb\n");
                  FN_EXIT(1, (int)NULL);
                  return NULL;
            }
            skb_reserve(skb, 2);
            skb_put(skb, buflen);               /* make room */

            /* create 802.3 header */
            e_hdr = (wlan_ethhdr_t *) skb->data;
            MAC_COPY(e_hdr->daddr, daddr);
            MAC_COPY(e_hdr->saddr, saddr);
            e_hdr->type = e_snap->type;

            /* Now copy the data from the 80211 frame.
               Make room in front for the eth header, and cut off the
               llc and snap from the 802.11 payload */
            memcpy(skb->data + WLAN_ETHHDR_LEN,
                 e_payload,
                 payload_length - sizeof(wlan_llc_t) - sizeof(wlan_snap_t));
```

```
   } else {
            acxlog(L_DEBUG | L_DATA, "rx: NON-ENCAP len: %d\n",
payload_length);
            /* any NON-ENCAP */
            /* it's a generic 80211+LLC or IPX 'Raw 802.3' */
            /*  build an 802.3 frame */
            /* allocate space and setup hostbuf */

            /* Test for an overlength frame */
            if (unlikely(payload_length + WLAN_ETHHDR_LEN >
WLAN_MAX_ETHFRM_LEN)) {
                  /* A bogus length ethfrm has been sent. */
                  /* Is someone trying an oflow attack? */
                  acxlog(L_STD, "rx: OTHER frame too large (%d > %d)\n",
                        payload_length,
                        WLAN_MAX_ETHFRM_LEN - WLAN_ETHHDR_LEN);
                  FN_EXIT(1, (int)NULL);
                  return NULL;
            }

            /* allocate space and setup host buffer */
            buflen = payload_length + WLAN_ETHHDR_LEN;
            skb = dev_alloc_skb(buflen + 2);          /* +2 is attempt to align IP header */
            if (unlikely(NULL == skb)) {
                  acxlog(L_STD, "rx: FAILED to allocate skb\n");
                  FN_EXIT(1, (int)NULL);
                  return NULL;
            }
            skb_reserve(skb, 2);
            skb_put(skb, buflen);               /* make room */

            /* set up the 802.3 header */
            e_hdr = (wlan_ethhdr_t *) skb->data;
            MAC_COPY(e_hdr->daddr, daddr);
            MAC_COPY(e_hdr->saddr, saddr);
            e_hdr->type = htons(payload_length);

            /* now copy the data from the 80211 frame */
            memcpy(skb->data + WLAN_ETHHDR_LEN, e_llc, payload_length);
   }

   skb->dev = priv->netdev;
   skb->protocol = eth_type_trans(skb, priv->netdev);

   /* the "<6>" output is from the KERN_INFO channel value */
/* Can be used to debug conversion process */
#if DEBUG_CONVERT
   acxlog(L_DATA, "p802.11 frame [%d]: ", (MAC_CNT_RCVD(rx_desc->data)));
   for (i = 0; i < MAC_CNT_RCVD(rx_desc->data); i++)
            acxlog(L_DATA, "%02x ", ((u8 *) w_hdr)[i]);
   acxlog(L_DATA, "\n");

   acxlog(L_DATA, "eth frame [%d]: ", skb->len);
   for (i = 0; i < skb->len; i++)
            acxlog(L_DATA, "%02x ", ((u8 *) skb->data)[i]);
```

```
    acxlog(L_DATA, "\n");
#endif

    FN_EXIT(0, OK);
    return skb;

}
```

ccmp_enc.h

```
**********armado de la mpdu encriptada**********/
/**********global data**********/
u8 PN[6];
extern u8 pmk[32];
u8 counter[16];
/**********/
extern void PRF(u8 *key, int key_len, u8 *prefix, int prefix_len, u8 *data,
    int data_len, u8 *output, int len);
int enc(struct sk_buff *skb, u8 *dest_data, wlandevice_t *priv);
/**********/
/**********/
int enc(struct sk_buff *skb, u8 *dest_data, wlandevice_t *priv) {
/**********construccion del ccmp header**********/
/**********construccion del Packet Number (PN)**********/
    u8 prefix[] = {'P','N',' ','c','o','n','s','t','r','u','c','t','i','o','n',0x00};
    /*len: longitud de la MPDU*/
    u16 len = (u16)cpu_to_le16(skb->len - sizeof(wlan_ethhdr_t));
    u8 data[len + 16]; /* se le suma el pad maximo */
    u32 time;

    rdtscl(time);
    data[0] = (u8)time;
    data[1] = (u8)(time >> 8);
    data[2] = (u8)(time >> 16);
    data[3] = (u8)(time >> 24);

    PRF(pmk, 32, prefix, strlen(prefix), data, 4, PN, 6);

    u8 ccmp_header[8];
    memcpy(ccmp_header, PN, 2);
    memcpy(&ccmp_header[4], &PN[2], 4);
    ccmp_header[2] = 0x00;
    ccmp_header[3] = 0x04; /*segun el libro*/
/**********/
/**********armado del primer bloque**********/

    u8 first_block[16];
    first_block[0] = 0x59;     /*********01011001********/
    first_block[1] = 0x00;     /********priority field*****/
    memcpy(&first_block[2], priv->dev_addr, 6);
    memcpy(&first_block[6], PN, 6);
    /*longitud del plaintext data (MPDU)*/
    u16 length = (u16)cpu_to_le16(skb->len - sizeof(wlan_ethhdr_t));
    first_block[14] = (u8)(length >> 8);
```

```
   first_block[15] = (u8)length;
/*****************************************************************************/
/***************************armado del MAC header*****************************/
   u8 mac_header[24];
   mac_header[0] = 0x08;/*bits 4,5,6,11,12,13 = 0*/
   mac_header[1] = 0x41;/*no incluye duration field*/
   memcpy(&mac_header[2], priv->bssid, 6);
   memcpy(&mac_header[8], priv->dev_addr, 6);
   memcpy(&mac_header[14], priv->bssid, 6);
   mac_header[20] = 0x00;
   mac_header[21] = 0x00;/*22 bytes hay que agregar 2 bytes para el primer pad*/
/*****************************************************************************/
/************************armado de la trama para el mic***********************/
   u8 pad_len; /*se fija la cantidad de pad a agregar*/
   if (len % 16 != 0) {
   u16 mult16 = (u16)(len / 16);
   pad_len = 16 - (u8)(len - mult16);
   }
   else pad_len = 0;

   u8 mic_aes[16];
   memcpy(data, first_block, 16);
   memcpy(&data[16], mac_header, 22);
   memcpy(&data[38], ccmp_header, 8);
   /*1st block + mac header + ccmp hdr = 46*/
   memset(&data[46], 0x00, 2); /*se agregan 2 bytes en 0*/
   /*copia de la MPDU*/
   memcpy(&data[48], skb->data + sizeof(wlan_ethhdr_t), skb->len -
sizeof(wlan_ethhdr_t));
   /*agregado del ultimo pad*/
   memset(&data[48 + cpu_to_le16(skb->len - sizeof(wlan_ethhdr_t))], 0x00,
(u8)pad_len);
   if (cbc_mac(data, 16 + 22 + 8 + 2 + len + pad_len, mic_aes))
   acxlog(L_BINSTD | L_ASSOC, "fallo el cbc-mac");
/*****************************************************************************/
/**************armado de la trama para ser encriptada con ccmp****************/
   memcpy(data, ccmp_header, 8);
   memcpy(&data[8], skb->data + sizeof(wlan_ethhdr_t), skb->len -
sizeof(wlan_ethhdr_t));
   memcpy(&data[8 + cpu_to_le16(skb->len - sizeof(wlan_ethhdr_t))], mic_aes, 8);
   memset(&data[8 + cpu_to_le16(skb->len - sizeof(wlan_ethhdr_t)) + 8], 0x00, 16);
/*****************************************************************************/
/*****************************armado del contador*****************************/
   memcpy(counter, first_block, 14);
   counter[14] = 0x00;
   counter[15] = 0x01;
/*****************************************************************************/
/*******************************encriptado con ccmp***************************/
   u16 i;
   u8 aux_aes[16], j;
   for (i = 0; i < (u16)((cpu_to_le16(skb->len - sizeof(wlan_ethhdr_t)) + pad_len +
8/*mic_aes*/) / 16); i++) {
      aes_enc(counter, aux_aes);
   for (j = 0;j < 16; j++) data[i*16 + j + 8/*ccmp hdr*/] ^= aux_aes[j];
   if (counter[15] == 255) {
```

```
        counter[14]++;
        counter[15] = 0x00;
    }
    else counter[15]++;
    }
    /*datos encriptados a la variable de destino*/
    memcpy(dest_data, data, skb->len - sizeof(wlan_ethhdr_t) + 16 /*8 ccmp hdr + 8
mic_aes*/);
    /*retorna la longitud total de datos encriptados*/
    return (int)(cpu_to_le16(skb->len - sizeof(wlan_ethhdr_t)) + 16);
}
```

ccmp_des_enc.h

```
/*****************************desencriptacion*********************************/
/*****************************************************************************/
int cbc_mac(u8 *source, int len, u8 *mic_aes);
int des_enc(p80211_hdr_t *w_hdr, int payload_length, wlandevice_t *priv);
/*****************************************************************************/
/******los datos recibidos se desencriptan y se pasan al buffer del socket******/

int cbc_mac(u8 *source, int len, u8 *mic_aes) {
    int i, j;
    if (len == 0) return 1;
    if ((len % 16) != 0) return 1;

    aes_enc(&source[0], &source[0]);
    for(i = 1;i < (int)(len / 16);i++) {
    for(j = 0;j < 16;j++) source[(i - 1)*16 + j] ^= source[i*16 + j];
    aes_enc(&source[i*16], &source[i*16]);
    }
    memcpy(mic_aes, &source[len - 1 - 16], 16);
    return 0;
}
/*****************************************************************************/
/*****************************************************************************/
int des_enc(p80211_hdr_t *w_hdr, int payload_length, wlandevice_t *priv) {

    u8 *payload = (u8 *)((u8 *)w_hdr + WLAN_HDR_A3_LEN);

/**************************primero extraemos el PN****************************/
    u8 PN_rcvd[16];
    memcpy(PN_rcvd, payload, 2);
    memcpy(&PN_rcvd[2], &payload[4], 4);

/*************armamos el counter y el first_block para desencriptar************/
    u8 first_block_rcvd[16];
    u8 counter[16];
    memset(first_block_rcvd, 0x59, 1);/*flag*/
    memset(&first_block_rcvd[1], 0x00, 1);/*priority*/
    memcpy(&first_block_rcvd[2], priv->bssid, 6);/*direccion de el punto de acceso*/
    memcpy(&first_block_rcvd[8], PN_rcvd, 6);/*PN recibido*/
    int dlen = payload_length - 16;/*longitud de la MPDU*/
    first_block_rcvd[15] = (u8)dlen;
```

```
   first_block_rcvd[14] = (u8)(dlen >> 8);
   memcpy(counter, first_block_rcvd, 14);
   counter[14] = 0x00;
   counter[15] = 0x01;

/**********************llamamos a la desencriptacion*************************/

   payload += 8; /*posicionamos en el ciphertext*/
   u8 ciphertext[payload_length - 8 + 16];/*16 por si no es multiplo de 16*/
   memcpy(ciphertext, payload, payload_length - 8); /*8 del ccmp header*/
   memset(&ciphertext[payload_length - 8], 0x00, 16);
   u16 i;
   u8 aux_aes[16], j;     /*deberia ser multiplo de 16*/
   for (i = 0;i < (u16)((u16)((dlen + 8)/16) + (dlen + 8)%16 + 16);i++) {
   aes_enc(counter, aux_aes);
   for(j = 0;j < 16;j++) ciphertext[i*16 + j] ^= aux_aes[j];
   if (counter[15] == 255) {
      counter[14]++;
      counter[15] = 0x00;
   }
   else counter[15]++;
   }

/**********************verificacion del mic_aes*******************************/
   u8 data[payload_length + 24/*mac_hdr*/ + 16 /*1st block*/+ 8 /*ccmp hdr*/
   + 16/*max pad*/];
   memcpy(data, first_block_rcvd, 16);

   /******continua el mac hdr******/

   data[16] = 0x08;/* frame control */
   data[17] = 0x42;/* frame control, no incluye el duration field*/
   memcpy(&data[18], priv->dev_addr, 6);
   memcpy(&data[24], priv->bssid, 6);
   memcpy(&data[30], priv->dev_addr, 6);
   memset(&data[36], 0x00, 2);/*secuence number*/

   /*********ahora vemos el pad********/

   u8 pad_len;
   if ((payload_length - 16) % 16 != 0) pad_len = (payload_length - 16) % 16;
   else pad_len = 0;
   u8 mic_aes[16];

   memset(&data[36], 0x00, 2);
   memcpy(&data[38], ciphertext, payload_length - 16);
   memcpy(&data[38 + payload_length - 16/*del mic_aes y el ccmp hdr*/], 0x00,
(u8)pad_len);
/**********************llamamos al calculo del mic_aes*************************/

   if (cbc_mac(data, payload_length - 16 + pad_len, mic_aes)) {
   acxlog(L_BINSTD | L_ASSOC, "fallo el cbc-mac");
      return 0;
   }
/************************nos fijamos si matchean los mics**********************/
```

```
   if (memcmp(&ciphertext[payload_length - 8], mic_aes, 8)) {
   acxlog(L_BINSTD | L_ASSOC, "no matchean los mics");
   return 0;
   }

/**********************copiamos la MPDU recibida al buffer*******************/

   memcpy(payload /*w_hdr + WLAN_HDR_A3_LEN*/, ciphertext, payload_length -
8);
   return (payload_length - 16);
}
```

MANUAL DE OPERACIÓN

El driver es muy simple de usar, todo se basa en la ejecución del script */scripts/start_net*. La Password debe ser configurada antes de ser compilado el driver en la función *auxiliar*. Al momento de la construcción de la función *auxiliar* se configuró la password: *AccessPoint*.

MEDICIONES

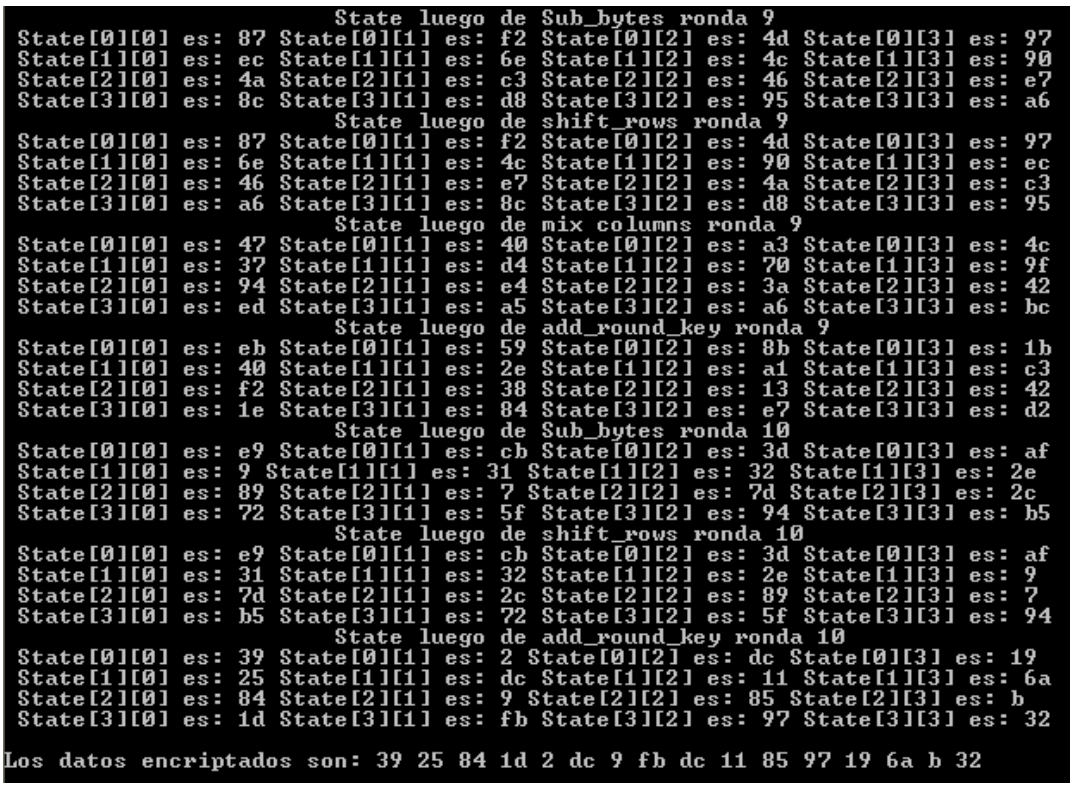
Para la primer parte del Proyecto la forma de testearlo fue ingresando datos que en el documento fips 197 se encuentran como vectores de testeo y observando que la salida sea la correcta según el documento.

El documento propone una clave = **2b 7e 15 16 28 ae d2 a6 ab f7 15 88 09 cf 4f 3c**, y datos de ingreso = **32 43 f6 a8 88 5a 30 8d 31 31 98 a2 e0 37 07 34** para que el algoritmo arroje los datos encriptados = **39 25 84 1d 02 dc 09 fb dc 11 85 97 19 6a 0b 32.**

El programa espera la confirmación si el usuario desea ver las operaciones sobre el *state* o no. Al iniciar el programa aestx.exe luego de ingresar los datos se ve lo siguiente:

```
                              Proyecto Final

       Presentacion del Sistema de Encriptacion AES en lenguaje C

                              Integrante

                         Leandro Martin Del Gesso

                               Tutores

                  Juan Carlos Bonadero, Monica Liberatori

                   Aqui se observara la encriptacion

       A continuacion se le pedira ingresar los datos a encriptar
       (16 bytes) que conformaran el State

       Ingrese el valor data_rx[0]: 32
       Ingrese el valor data_rx[1]: 43
       Ingrese el valor data_rx[2]: f6
       Ingrese el valor data_rx[3]: a8
       Ingrese el valor data_rx[4]: 88
       Ingrese el valor data_rx[5]: 5a
       Ingrese el valor data_rx[6]: 30
       Ingrese el valor data_rx[7]: 8d
       Ingrese el valor data_rx[8]: 31
       Ingrese el valor data_rx[9]: 31
       Ingrese el valor data_rx[10]: 98
       Ingrese el valor data_rx[11]: a2
       Ingrese el valor data_rx[12]: e0
       Ingrese el valor data_rx[13]: 37
       Ingrese el valor data_rx[14]: 07
       Ingrese el valor data_rx[15]: 34

                              State inicial
State[0][0] es: 32 State[0][1] es: 88 State[0][2] es: 31 State[0][3] es: e0
State[1][0] es: 43 State[1][1] es: 5a State[1][2] es: 31 State[1][3] es: 37
State[2][0] es: f6 State[2][1] es: 30 State[2][2] es: 98 State[2][3] es: 7
State[3][0] es: a8 State[3][1] es: 8d State[3][2] es: a2 State[3][3] es: 34

       A continuacion se le pedira ingresar la clave master
       (128 bits) en 4 partes de 32 bits (4 bytes)

Ingrese la Master_key[0] en hexadecimal(32 bits, 4 bytes): 2b7e1516
Ingrese la Master_key[1] en hexadecimal(32 bits, 4 bytes): 28aed2a6
Ingrese la Master_key[2] en hexadecimal(32 bits, 4 bytes): abf71588
Ingrese la Master_key[3] en hexadecimal(32 bits, 4 bytes): 09cf4f3c


Desea ver todas las operaciones sobre el state? (s/n, default: n)
_
```

A continuación se muestra la salida con parte de las operaciones sobre el *state* a partir de la ronda Nº 9:



Como se puede apreciar, la salida es la correspondiente.

De la misma manera se probó la desencriptación, se ingresaron los datos que fueron salidas del programa anterior, la misma clave y obviamente debió arrojar los datos de entrada el aestx.exe.

El principio de aesrx.exe se muestra aquí:

La salida es:

Como se puede apreciar los programas se ejecutan perfectamente.

En cuanto al archivo aes.exe se observan las dos operaciones juntas. Ingresándole los mismos datos que a aestx.exe se observa lo siguiente:

```
                              Proyecto Final

       Presentacion del Sistema de Encriptacion AES en lenguaje C

                              Integrante

                         Leandro Martin Del Gesso

                                Tutores

                   Juan Carlos Bonadero, Monica Liberatori

       A continuacion se le pedira ingresar los datos a encriptar
       (16 bytes) que conformaran el State

       Ingrese data_rx[0] en hexadecimal (1 byte): 32
       Ingrese data_rx[1] en hexadecimal (1 byte): 43
       Ingrese data_rx[2] en hexadecimal (1 byte): f6
       Ingrese data_rx[3] en hexadecimal (1 byte): a8
       Ingrese data_rx[4] en hexadecimal (1 byte): 88
       Ingrese data_rx[5] en hexadecimal (1 byte): 5a
       Ingrese data_rx[6] en hexadecimal (1 byte): 30
       Ingrese data_rx[7] en hexadecimal (1 byte): 8d
       Ingrese data_rx[8] en hexadecimal (1 byte): 31
       Ingrese data_rx[9] en hexadecimal (1 byte): 31
       Ingrese data_rx[10] en hexadecimal (1 byte): 98
       Ingrese data_rx[11] en hexadecimal (1 byte): a2
       Ingrese data_rx[12] en hexadecimal (1 byte): e0
       Ingrese data_rx[13] en hexadecimal (1 byte): 37
       Ingrese data_rx[14] en hexadecimal (1 byte): 07
       Ingrese data_rx[15] en hexadecimal (1 byte): 34

                              State inicial
State[0][0] es: 32 State[0][1] es: 88 State[0][2] es: 31 State[0][3] es: e0
State[1][0] es: 43 State[1][1] es: 5a State[1][2] es: 31 State[1][3] es: 37
State[2][0] es: f6 State[2][1] es: 30 State[2][2] es: 98 State[2][3] es: 7
State[3][0] es: a8 State[3][1] es: 8d State[3][2] es: a2 State[3][3] es: 34

       A continuacion se le pedira ingresar la clave master
       (128 bits) en 4 partes de 32 bits (4 bytes)

Ingrese la Master_key[0] en hexadecimal(32 bits, 4 bytes): 2b7e1516
Ingrese la Master_key[1] en hexadecimal(32 bits, 4 bytes): 28aed2a6
Ingrese la Master_key[2] en hexadecimal(32 bits, 4 bytes): abf71588
Ingrese la Master_key[3] en hexadecimal(32 bits, 4 bytes): 09cf4f3c


Desea ver todas las operaciones sobre el state? (s/n, default: n)
n


Los datos encriptados son: 39 25 84 1d 2 dc 9 fb dc 11 85 97 19 6a b 32



Los datos desencriptados son: 32 43 f6 a8 88 5a 30 8d 31 31 98 a2 e0 37 7 34
```

Lo que resume finalmente la primera etapa del proyecto.

Para la segunda etapa del Proyecto, todas las mediciones se pudieron realizar mediante el debug que provee el driver, el cual va enviando mensajes al archivo */var/log/syslog* como se mencionó anteriormente.

El debug puede ser configurado como: nulo, mínimo y máximo. Si bien el debug máximo tiene la ventaja de que muestra todas las tramas transmitidas y recibidas, posee la desventaja de que toma bastante tiempo en mandar cada mensaje al archivo, por lo que, si se busca observar las tramas hay que tener en cuenta que los tiempos reales son afectados por el debug y en gran medida como se mostrará más adelante.

A continuación se muestra la parte del archivo *syslog* en la cual está la autenticación a sistema abierto, la asociación y la recepción del primer mensaje del 4-way-handshake con la correspondiente respuesta del adaptador configurado con máximo debug.

```
May 31 18:46:03 slackware kernel: tx: pkt (MGMT/Authen): len 35 (30/6) rate 0001 status
1
May 31 18:46:03 slackware kernel: tx: 802.11 header[30]: <4>B0 00 00 80 00 13 46 FD B1
5B 00 0F 3D 58 7E 14
May 31 18:46:03 slackware kernel: 00 13 46 FD B1 5B 00 00 00 00 01 00 00 00
May 31 18:46:03 slackware kernel: tx: 802.11 payload[6]: <4>00 00 01 00 00 00
May 31 18:46:03 slackware kernel: 35865752 us       <== acx_dma_tx_data
May 31 18:46:03 slackware kernel: 35865777 us     <== acx_transmit_authen1
May 31 18:46:03 slackware kernel: 35865812 us     ==> acx_set_status
May 31 18:46:03 slackware kernel: 35865956 us       ==> acx_set_timer
May 31 18:46:03 slackware kernel: <acx_set_timer> Elapse = 1500000
May 31 18:46:03 slackware kernel: 35866013 us       <== acx_set_timer
May 31 18:46:03 slackware kernel: 35866040 us     <== acx_set_status
May 31 18:46:03 slackware kernel: 35866066 us   <== acx_complete_dot11_scan
May 31 18:46:03 slackware kernel: Send a stop scan cmd...
May 31 18:46:03 slackware kernel: 35866118 us   ==> acx_issue_cmd
May 31 18:46:03 slackware kernel: acx_issue_cmd cmd 0x9 timeout 5000.
May 31 18:46:03 slackware kernel: input pdr (len=0):
May 31 18:46:03 slackware kernel: cmd_type 0x0000, cmd_status 0x0000 [Idle]
May 31 18:46:03 slackware kernel: 35867083 us     ==> acx_schedule
May 31 18:46:03 slackware kernel: 35867736 us       ==> acx_interrupt
May 31 18:46:03 slackware kernel: IRQTYPE: 0x200, irq_mask: 0x98E5
May 31 18:46:03 slackware kernel: Got Command Complete IRQ
May 31 18:46:03 slackware kernel: 35867822 us       <== acx_interrupt
May 31 18:46:03 slackware kernel: 35877525 us       ==> acx_interrupt
May 31 18:46:03 slackware kernel: IRQTYPE: 0x2, irq_mask: 0x98E5
May 31 18:46:03 slackware kernel: Got Tx Complete IRQ
May 31 18:46:03 slackware kernel: 35877637 us       <== acx_interrupt
May 31 18:46:03 slackware kernel: 35884000 us       ==> acx_interrupt
May 31 18:46:03 slackware kernel: IRQTYPE: 0x8, irq_mask: 0x98E5
May 31 18:46:03 slackware kernel: 35884086 us         ==> acx_process_rx_desc
May 31 18:46:03 slackware kernel: rx: buf 9 full
May 31 18:46:03 slackware kernel: acx_process_rx_desc: using curr_idx 9, rx_tail is now
10
May 31 18:46:03 slackware kernel: 35884194 us           ==> acx_get_packet_type_string
May 31 18:46:03 slackware kernel: d5e2ff6b          <== acx_get_packet_type_string:
cf870920
May 31 18:46:03 slackware kernel: rx: pkt 09 (MGMT/Authen): time 5501495 len 30 signal
32 SNR 0 macstat 83 phystat 00 phyrate 10 status 2
May 31 18:46:03 slackware kernel: rx: 802.11 buf[30]: <4>B0 00 3A 01 00 0F 3D 58 7E 14
00 13 46 FD B1 5B
May 31 18:46:03 slackware kernel: 00 13 46 FD B1 5B 80 BA 00 00 02 00 00 00
May 31 18:46:03 slackware kernel: 35884536 us           ==> acx_rx_ieee802_11_frame
May 31 18:46:03 slackware kernel: 35884579 us             ==> acx_process_mgmt_frame
May 31 18:46:03 slackware kernel: diferencia entre auth/req/resp: 18838  us
```

```
May 31 18:46:03 slackware kernel: 35884660 us                ==> auxiliar
May 31 18:46:03 slackware kernel: 35884698 us                  ==> PasswordHash
May 31 18:46:03 slackware kernel: 36256075 us                  <== PasswordHash
May 31 18:46:03 slackware kernel: 36256114 us                  ==> PRF
May 31 18:46:03 slackware kernel: 36256338 us                  <== PRF
May 31 18:46:03 slackware kernel: 36256364 us                  ==> PRF
May 31 18:46:03 slackware kernel: 36256487 us                  <== PRF
May 31 18:46:03 slackware kernel: 36256517 us                  ==> init_WPA_frame
May 31 18:46:03 slackware kernel: 36256570 us                  <== init_WPA_frame
May 31 18:46:03 slackware kernel: diferencia auxiliar: 371903 us
May 31 18:46:03 slackware kernel: 36256628 us               <== auxiliar
May 31 18:46:03 slackware kernel: 36256670 us               ==> acx_process_authen
May 31 18:46:03 slackware kernel: 00:0F:3D:58:7E:14 <4>00:0F:3D:58:7E:14
<4>00:13:46:FD:B1:5B <4>00:13:46:FD:B1:5B <4>00:13:46:FD:B1:5B
May 31 18:46:03 slackware kernel: Algorithm is ok
May 31 18:46:03 slackware kernel: 36256872 us                  ==>
acx_sta_list_get_from_hash
May 31 18:46:03 slackware kernel: 36256910 us                  <==
acx_sta_list_get_from_hash
May 31 18:46:03 slackware kernel: Got current client for sta hash tab
May 31 18:46:03 slackware kernel: Found acceptable client
May 31 18:46:03 slackware kernel: 36256989 us                  ==> acx_sta_list_add
May 31 18:46:03 slackware kernel: 36257023 us                    ==> acx_sta_list_alloc
May 31 18:46:03 slackware kernel: d81bdffd                   <== acx_sta_list_alloc:
c87213f8
May 31 18:46:03 slackware kernel: d81bf410                <== acx_sta_list_add:
c87213f8
May 31 18:46:03 slackware kernel: acx_process_authen auth seq step 2.
May 31 18:46:03 slackware kernel: 36257192 us                  ==> acx_set_status
May 31 18:46:03 slackware kernel: 36257292 us                    ==> acx_set_timer
May 31 18:46:03 slackware kernel: <acx_set_timer> Elapse = 1500000
May 31 18:46:03 slackware kernel: 36257355 us                    <== acx_set_timer
May 31 18:46:03 slackware kernel: 36257385 us                  <== acx_set_status
May 31 18:46:03 slackware kernel: 36257422 us                  ==> acx_transmit_assoc_req
May 31 18:46:03 slackware kernel: Sending association request, awaiting response! NOT
ASSOCIATED YET.
May 31 18:46:03 slackware kernel: 36257486 us                    ==> acx_get_tx_desc
May 31 18:46:03 slackware kernel: tx: got desc 1, 14 remain
May 31 18:46:03 slackware kernel: 36257550 us                    <== acx_get_tx_desc
May 31 18:46:03 slackware kernel: 36257615 us                    ==> acx_dma_tx_data
May 31 18:46:03 slackware kernel: UHOH, packet has a different length than struct
framehdr (0x18 vs. 0x26)
May 31 18:46:03 slackware kernel: 36257780 us                      ==>
acx_get_packet_type_string
May 31 18:46:03 slackware kernel: d81d1100                     <==
acx_get_packet_type_string: cf870920
May 31 18:46:03 slackware kernel: tx: pkt (MGMT/AssocReq): len 67 (67/43) rate 0001
status 3
May 31 18:46:03 slackware kernel: tx: 802.11 header[67]: <4>00 00 00 80 00 13 46 FD B1
5B 00 0F 3D 58 7E 14
May 31 18:46:03 slackware kernel: 00 13 46 FD B1 5B 00 00 21 00 64 00 00 03 4C 41
May 31 18:46:03 slackware kernel: 43 01 08 82 84 8B 96 0C 18 30 48 DD 16 00 50 F2
May 31 18:46:03 slackware kernel: 01 01 00 00 50 F2 04 01 00 00 50 F2 04 01 00 00
May 31 18:46:03 slackware kernel: 50 F2 02
May 31 18:46:03 slackware kernel: tx: 802.11 payload[43]: <4>96 0C 18 30 48 DD 16 00 50
F2 01 01 00 00 50 F2
May 31 18:46:03 slackware kernel: 04 01 00 00 50 F2 04 01 00 00 50 F2 02 F2 04 01
May 31 18:46:03 slackware kernel: 00 00 50 F2 04 01 00 00 50 F2 02
May 31 18:46:03 slackware kernel: 36258549 us                    <== acx_dma_tx_data
May 31 18:46:03 slackware kernel: d81e33f2                 <== acx_transmit_assoc_req:
00000000
May 31 18:46:03 slackware kernel: d81e430a               <== acx_process_authen:
00000001
May 31 18:46:03 slackware kernel: d81e50db             <== acx_process_mgmt_frame:
00000000
May 31 18:46:03 slackware kernel: d81e5e0c           <== acx_rx_ieee802_11_frame:
00000000
May 31 18:46:03 slackware kernel: acx_process_rx_desc: using curr_idx 10, rx_tail is now
11
May 31 18:46:03 slackware kernel: 36258765 us           ==> acx_get_packet_type_string
May 31 18:46:03 slackware kernel: d81e8b6b           <== acx_get_packet_type_string:
cf870920
```

```
May 31 18:46:03 slackware kernel: rx: pkt 10 (MGMT/Beacon): time 5509647 len 101 signal
25 SNR 0 macstat e5 phystat 10 phyrate 10 status 3
May 31 18:46:03 slackware kernel: rx: 802.11 buf[101]: <4>80 00 00 00 FF FF FF FF FF FF
00 13 46 FD B1 5B
May 31 18:46:03 slackware kernel: 00 13 46 FD B1 5B 90 BA 81 F1 9D 62 00 00 00 00
May 31 18:46:03 slackware kernel: 14 00 31 04 00 03 4C 41 43 01 08 82 84 8B 96 0C
May 31 18:46:03 slackware kernel: 18 30 48 03 01 06 05 04 00 01 00 00 07 06 4C 41
May 31 18:46:03 slackware kernel: 20 01 0B 14 2A 01 02 32 04 12 24 60 6C DD 16 00
May 31 18:46:03 slackware kernel: 50 F2 01 01 00 00 50 F2 04 01 00 00 50 F2 04 01
May 31 18:46:03 slackware kernel: 00 00 50 F2 02
May 31 18:46:03 slackware kernel: 36259477 us           ==> acx_rx_ieee802_11_frame
May 31 18:46:03 slackware kernel: 36259508 us             ==> acx_process_mgmt_frame
May 31 18:46:03 slackware kernel: d81fa9b5             <== acx_process_mgmt_frame:
00000000
May 31 18:46:03 slackware kernel: d81fb627           <== acx_rx_ieee802_11_frame:
00000000
May 31 18:46:03 slackware kernel: acx_process_rx_desc: using curr_idx 11, rx_tail is now
12
May 31 18:46:03 slackware kernel: 36259639 us           ==> acx_get_packet_type_string
May 31 18:46:03 slackware kernel: d81fdf77           <== acx_get_packet_type_string:
cf870920
May 31 18:46:03 slackware kernel: rx: pkt 11 (MGMT/Beacon): time 5530127 len 101 signal
25 SNR 0 macstat e5 phystat 10 phyrate 10 status 3
May 31 18:46:03 slackware kernel: rx: 802.11 buf[101]: <4>80 00 00 00 FF FF FF FF FF FF
00 13 46 FD B1 5B
May 31 18:46:03 slackware kernel: 00 13 46 FD B1 5B A0 BA 81 41 9E 62 00 00 00 00
May 31 18:46:03 slackware kernel: 14 00 31 04 00 03 4C 41 43 01 08 82 84 8B 96 0C
May 31 18:46:03 slackware kernel: 18 30 48 03 01 06 05 04 00 01 00 00 07 06 4C 41
May 31 18:46:03 slackware kernel: 20 01 0B 14 2A 01 02 32 04 12 24 60 6C DD 16 00
May 31 18:46:03 slackware kernel: 50 F2 01 01 00 00 50 F2 04 01 00 00 50 F2 04 01
May 31 18:46:03 slackware kernel: 00 00 50 F2 02
May 31 18:46:03 slackware kernel: 36260347 us           ==> acx_rx_ieee802_11_frame
May 31 18:46:03 slackware kernel: 36260378 us             ==> acx_process_mgmt_frame
May 31 18:46:03 slackware kernel: d820fe1e             <== acx_process_mgmt_frame:
00000000
May 31 18:46:03 slackware kernel: d8210b67           <== acx_rx_ieee802_11_frame:
00000000
May 31 18:46:03 slackware kernel: 36260476 us         <== acx_process_rx_desc
May 31 18:46:03 slackware kernel: Got Rx Complete IRQ
May 31 18:46:03 slackware kernel: 36260531 us       <== acx_interrupt
May 31 18:46:03 slackware kernel: 36260710 us     <== acx_schedule
May 31 18:46:03 slackware kernel: cmd_type 0x0009, cmd_status 0x0001 [Success]
May 31 18:46:03 slackware kernel: d821c5c1   <== acx_issue_cmd: 00000000
May 31 18:46:03 slackware kernel: 36260952 us <== acx_after_interrupt_task
May 31 18:46:03 slackware kernel: 36270798 us ==> acx_interrupt
May 31 18:46:03 slackware kernel: IRQTYPE: 0x2, irq_mask: 0x98E5
May 31 18:46:03 slackware kernel: Got Tx Complete IRQ
May 31 18:46:03 slackware kernel: 36270905 us <== acx_interrupt
May 31 18:46:03 slackware kernel: 36277381 us ==> acx_interrupt
May 31 18:46:03 slackware kernel: IRQTYPE: 0x8, irq_mask: 0x98E5
May 31 18:46:03 slackware kernel: 36277454 us   ==> acx_process_rx_desc
May 31 18:46:03 slackware kernel: rx: buf 12 full
May 31 18:46:03 slackware kernel: acx_process_rx_desc: using curr_idx 12, rx_tail is now
13
May 31 18:46:03 slackware kernel: 36277541 us     ==> acx_get_packet_type_string
May 31 18:46:03 slackware kernel: d83b31c7     <== acx_get_packet_type_string: cf870920
May 31 18:46:03 slackware kernel: rx: pkt 12 (MGMT/AssocResp): time 5540849 len 40
signal 32 SNR 0 macstat 83 phystat 00 phyrate 10 status 3
May 31 18:46:03 slackware kernel: rx: 802.11 buf[40]: <4>10 00 3A 01 00 0F 3D 58 7E 14
00 13 46 FD B1 5B
May 31 18:46:03 slackware kernel: 00 13 46 FD B1 5B B0 BA 31 00 00 00 01 C0 01 08
May 31 18:46:03 slackware kernel: 82 84 8B 96 0C 18 30 48
May 31 18:46:03 slackware kernel: 36277924 us     ==> acx_rx_ieee802_11_frame
May 31 18:46:03 slackware kernel: 36277962 us       ==> acx_process_mgmt_frame
May 31 18:46:03 slackware kernel: diferencia entre assoc/req/resp: 412221  us
May 31 18:46:03 slackware kernel: 36278036 us         ==> acx_process_assocresp
May 31 18:46:03 slackware kernel: d83bf9bb         <== acx_process_assocresp: 00000001
May 31 18:46:03 slackware kernel: d83c0727       <== acx_process_mgmt_frame: 00000000
May 31 18:46:03 slackware kernel: d83c13e5     <== acx_rx_ieee802_11_frame: 00000000
May 31 18:46:03 slackware kernel: 36278192 us   <== acx_process_rx_desc
May 31 18:46:03 slackware kernel: Got Rx Complete IRQ
May 31 18:46:03 slackware kernel: 36278239 us <== acx_interrupt
May 31 18:46:03 slackware kernel: 36278283 us ==> acx_after_interrupt_task
```

```
May 31 18:46:03 slackware kernel: configuring: type=0xA len=2
May 31 18:46:03 slackware kernel: 36278354 us   ==> acx_issue_cmd
May 31 18:46:03 slackware kernel: acx_issue_cmd cmd 0x2 timeout 5000.
May 31 18:46:03 slackware kernel: input pdr (len=6):
May 31 18:46:03 slackware kernel: 0A 00 02 00 01 C0
May 31 18:46:03 slackware kernel: cmd_type 0x0000, cmd_status 0x0000 [Idle]
May 31 18:46:03 slackware kernel: 36278955 us     ==> acx_interrupt
May 31 18:46:03 slackware kernel: IRQTYPE: 0x200, irq_mask: 0x98E5
May 31 18:46:03 slackware kernel: Got Command Complete IRQ
May 31 18:46:03 slackware kernel: 36279028 us     <== acx_interrupt
May 31 18:46:03 slackware kernel: cmd_type 0x0002, cmd_status 0x0001 [Success]
May 31 18:46:03 slackware kernel: d83db69e   <== acx_issue_cmd: 00000000
May 31 18:46:03 slackware kernel: 36279261 us   ==> acx_set_status
May 31 18:46:03 slackware kernel: tx: carrier on after association
May 31 18:46:03 slackware kernel: tx: wake queue after association
May 31 18:46:03 slackware kernel: 36279421 us   <== acx_set_status
May 31 18:46:03 slackware kernel: ASSOCIATED!
May 31 18:46:03 slackware kernel: 36279455 us <== acx_after_interrupt_task
May 31 18:46:03 slackware kernel: 36309546 us ==> acx_interrupt
May 31 18:46:03 slackware kernel: IRQTYPE: 0x8, irq_mask: 0x98E5
May 31 18:46:03 slackware kernel: 36309649 us   ==> acx_process_rx_desc
May 31 18:46:03 slackware kernel: rx: buf 13 full
May 31 18:46:03 slackware kernel: acx_process_rx_desc: using curr_idx 13, rx_tail is now
14
May 31 18:46:03 slackware kernel: 36309746 us     ==> acx_get_packet_type_string
May 31 18:46:03 slackware kernel: d86c57b7     <== acx_get_packet_type_string: cf870920
May 31 18:46:03 slackware kernel: rx: pkt 13 (DATA/DataOnly): time 5543343 len 131
signal 32 SNR 0 macstat 83 phystat 10 phyrate 10 status 4
May 31 18:46:03 slackware kernel: rx: 802.11 buf[131]: <4>08 02 3A 01 00 0F 3D 58 7E 14
00 13 46 FD B1 5B
May 31 18:46:03 slackware kernel: 00 13 46 FD B1 5B C0 BA AA AA 03 00 00 00 88 8E
May 31 18:46:03 slackware kernel: 01 03 00 5F FE 00 8A 00 10 00 00 00 00 00 00 00
May 31 18:46:03 slackware kernel: 01 77 C9 69 D1 BC AF 2E 7B 56 D1 5E 11 E0 C8 E6
May 31 18:46:03 slackware kernel: 19 89 79 D1 F4 76 CE 5B C2 A3 93 3F 65 23 F8 ED
May 31 18:46:03 slackware kernel: 12 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
May 31 18:46:03 slackware kernel: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
May 31 18:46:03 slackware kernel: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
May 31 18:46:03 slackware kernel: 00 00 00
May 31 18:46:03 slackware kernel: 36310614 us     ==> acx_rx_ieee802_11_frame
May 31 18:46:03 slackware kernel: 36310653 us       ==> acx_4_way
May 31 18:46:03 slackware kernel: 36310693 us         ==> acx_get_tx_desc
May 31 18:46:03 slackware kernel: tx: got desc 2, 13 remain
May 31 18:46:03 slackware kernel: 36310754 us         <== acx_get_tx_desc
May 31 18:46:03 slackware kernel: 36310805 us         ==> four_way_2
May 31 18:46:03 slackware kernel: dev_adrr < AP's addr, snonce < anonce
May 31 18:46:03 slackware kernel: 36310878 us           ==> PRF
May 31 18:46:03 slackware kernel: 36311339 us           <== PRF
May 31 18:46:03 slackware kernel: mic calculado con hmac_sha1
May 31 18:46:03 slackware kernel: Enviando 2do mensaje del 4-way handshake
May 31 18:46:03 slackware kernel: 36311639 us         <== four_way_2
May 31 18:46:03 slackware kernel: la longitud de la trama es 123
May 31 18:46:03 slackware kernel: 36311701 us         ==> acx_dma_tx_data
May 31 18:46:03 slackware kernel: UHOH, packet has a different length than struct
framehdr (0x20 vs. 0x26)
May 31 18:46:03 slackware kernel: 36311854 us           ==> acx_get_packet_type_string
May 31 18:46:03 slackware kernel: d86f8f4b          <== acx_get_packet_type_string:
cf870920
May 31 18:46:03 slackware kernel: tx: pkt (DATA/DataOnly): len 123 (155/123) rate 0001
status 4
May 31 18:46:03 slackware kernel: tx: 802.11 header[155]: <4>08 01 00 00 00 13 46 FD B1
5B 00 0F 3D 58 7E 14
May 31 18:46:03 slackware kernel: 00 13 46 FD B1 5B 00 00 AA AA 03 00 00 00 88 8E
May 31 18:46:03 slackware kernel: 01 03 00 77 FE 01 0A 00 10 00 00 00 00 00 00 00
May 31 18:46:03 slackware kernel: 01 2B D6 44 10 32 14 EE 68 FC B7 64 66 92 43 2B
May 31 18:46:03 slackware kernel: 9B E3 1F D3 78 35 04 BB 12 6D AB 0E 45 21 7D EC
May 31 18:46:03 slackware kernel: D5 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
May 31 18:46:03 slackware kernel: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
May 31 18:46:03 slackware kernel: 00 E8 B3 D1 EA 79 70 F4 56 5E 98 67 0D 43 56 4B
May 31 18:46:03 slackware kernel: 04 00 18 DD 16 00 50 F2 01 01 00 00 50 F2 04 01
May 31 18:46:03 slackware kernel: 00 00 50 F2 04 01 00 00 50 F2 02
May 31 18:46:03 slackware kernel: tx: 802.11 payload[123]: <4>0A 00 10 00 00 00 00 00 00
00 01 2B D6 44 10 32
May 31 18:46:03 slackware kernel: 14 EE 68 FC B7 64 66 92 43 2B 9B E3 1F D3 78 35
```

```
May 31 18:46:03 slackware kernel: 04 BB 12 6D AB 0E 45 21 7D EC D5 00 00 00 00 00
May 31 18:46:03 slackware kernel: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
May 31 18:46:03 slackware kernel: 00 00 00 00 00 00 00 00 00 00 00 E8 B3 D1 EA 79
May 31 18:46:03 slackware kernel: 70 F4 56 5E 98 67 0D 43 56 4B 04 00 18 DD 16 00
May 31 18:46:03 slackware kernel: 50 F2 01 01 00 00 50 F2 04 01 00 00 50 F2 04 01
May 31 18:46:03 slackware kernel: 00 00 50 F2 02 01 00 00 50 F2 02
May 31 18:46:03 slackware kernel: 36313483 us         <== acx_dma_tx_data
     May 31 18:46:03 slackware kernel: diferencia 4-way: 2821 us
May 31 18:46:03 slackware kernel: 36313689 us       <== acx_4_way
May 31 18:46:03 slackware kernel: d87257ff     <== acx_rx_ieee802_11_frame: 00000000
May 31 18:46:03 slackware kernel: 36313763 us   <== acx_process_rx_desc
May 31 18:46:03 slackware kernel: Got Rx Complete IRQ
May 31 18:46:03 slackware kernel: 36313826 us <== acx_interrupt
May 31 18:46:03 slackware kernel: 36329332 us ==> acx_interrupt
May 31 18:46:03 slackware kernel: IRQTYPE: 0x2, irq_mask: 0x98E5
May 31 18:46:03 slackware kernel: Got Tx Complete IRQ
May 31 18:46:03 slackware kernel: 36329437 us <== acx_interrupt
May 31 18:46:03 slackware kernel: 36373504 us ==> acx_interrupt
May 31 18:46:03 slackware kernel: IRQTYPE: 0x8, irq_mask: 0x98E5
May 31 18:46:03 slackware kernel: 36373580 us   ==> acx_process_rx_desc
May 31 18:46:03 slackware kernel: rx: buf 14 full
May 31 18:46:03 slackware kernel: acx_process_rx_desc: using curr_idx 14, rx_tail is now
15
May 31 18:46:03 slackware kernel: 36373672 us     ==> acx_get_packet_type_string
May 31 18:46:03 slackware kernel: d8cde146     <== acx_get_packet_type_string: cf870920
May 31 18:46:03 slackware kernel: rx: pkt 14 (MGMT/Beacon): time 5550607 len 101 signal
25 SNR 0 macstat e5 phystat 00 phyrate 10 status 4
May 31 18:46:03 slackware kernel: rx: 802.11 buf[101]: <4>80 00 00 00 FF FF FF FF FF FF
00 13 46 FD B1 5B
May 31 18:46:03 slackware kernel: 00 13 46 FD B1 5B D0 BA 81 91 9E 62 00 00 00 00
May 31 18:46:03 slackware kernel: 14 00 31 04 00 03 4C 41 43 01 08 82 84 8B 96 0C
May 31 18:46:03 slackware kernel: 18 30 48 03 01 06 05 04 00 01 00 00 07 06 4C 41
May 31 18:46:03 slackware kernel: 20 01 0B 14 2A 01 02 32 04 12 24 60 6C DD 16 00
May 31 18:46:03 slackware kernel: 50 F2 01 01 00 00 50 F2 04 01 00 00 50 F2 04 01
May 31 18:46:03 slackware kernel: 00 00 50 F2 02
```

Como se puede apreciar, el punto de acceso llega al primer mensaje del 4-way-handshake, pero luego de la respuesta correspondiente por parte del adaptador, no se recibe el tercer mensaje y al cabo de un segundo como se mencionó anteriormente el punto de acceso repite el mensaje. Los mensajes que son importantes y se desean destacar están subrayados.

De la misma manera que se uso en el driver una llamada al sistema para tener noción del tiempo transcurrido, para le debug se utilizó de forma tal de saber el tiempo transcurrido entre dos eventos. Es por es que aparecen mensajes “diferencia…..” de tiempo.

Como se observa los tiempos que se manejas con este debug son del orden de los milisegundos y es mucho tiempo.

Para disminuir esos tiempos pero aún así pudiendo hacer un debug se recurre a la opción de mínimo debug que muestra lo siguiente:

```
Jun  1 10:56:48 slackware kernel: <acx_join_bssid> JoinBSSID MAC:00 13 46 FD B1 5B
```

```
Jun  1 10:56:48 slackware kernel: Sending authentication1 request, awaiting response!
Jun  1 10:56:48 slackware kernel: diferencia entre auth/req/resp: 17481  us
Jun  1 10:56:48 slackware kernel: diferencia auxiliar: 376453 us
Jun  1 10:56:48 slackware kernel: 00:0F:3D:58:7E:14 <4>00:0F:3D:58:7E:14
<4>00:13:46:FD:B1:5B <4>00:13:46:FD:B1:5B <4>00:13:46:FD:B1:5B
Jun  1 10:56:48 slackware kernel: Algorithm is ok
Jun  1 10:56:48 slackware kernel: Got current client for sta hash tab
Jun  1 10:56:48 slackware kernel: Found acceptable client
Jun  1 10:56:48 slackware kernel: acx_process_authen auth seq step 2.
Jun  1 10:56:48 slackware kernel: Sending association request, awaiting response! NOT
ASSOCIATED YET.
Jun  1 10:56:48 slackware kernel: diferencia entre assoc/req/resp: 414375  us
Jun  1 10:56:48 slackware kernel: ASSOCIATED!
Jun  1 10:56:48 slackware kernel: diferencia 4-way: 613 us
Jun  1 10:56:49 slackware kernel: Got Info IRQ: status 0x0002, type 0xe400: (unknown)
Jun  1 10:56:49 slackware last message repeated 2 times
Jun  1 10:56:49 slackware kernel: diferencia 4-way: 129 us
Jun  1 10:56:50 slackware kernel: Got Info IRQ: status 0x0002, type 0xe400: (unknown)
Jun  1 10:56:50 slackware last message repeated 5 times
Jun  1 10:56:50 slackware kernel: diferencia 4-way: 98 us
Jun  1 10:56:51 slackware kernel: Got Info IRQ: status 0x0002, type 0xe400: (unknown)
Jun  1 10:56:51 slackware kernel: diferencia 4-way: 86 us
Jun  1 10:56:52 slackware kernel: Got Info IRQ: status 0x0002, type 0xe400: (unknown)
Jun  1 10:56:52 slackware kernel: Got Info IRQ: status 0x0002, type 0xe400: (unknown)
Jun  1 10:56:52 slackware kernel: DEAUTHEN <4>00:0F:3D:58:7E:14 <4>00:0F:3D:58:7E:14
<4>00:13:46:FD:B1:5B <4>00:13:46:FD:B1:5B <4>00:13:46:FD:B1:5B
Jun  1 10:56:52 slackware kernel: Processing deauthen packet. Hmm, should this have
happened?
Jun  1 10:56:52 slackware kernel: Got Info IRQ: status 0x0002, type 0xe400: (unknown)
```

Se pueden ver las diferencias de tiempos que disminuyeron notablemente. Pero además se puede observar como va disminuyendo el tiempo que tarda al adaptador en responder a los sucesivos primeros mensajes recibidos desde el punto de acceso llegando a un valor de 86us el cual varía según el momento. El último mensaje no superó el valor de 100us ni disminuyó de 80us en todas las pruebas con mínimo Debug.

No se pudo lograr que el punto de acceso responda el tercer mensaje del 4-way-handshake luego de probar todo lo posible, de aquí que no se pudo testear el código de la encriptación dado que no se ejecuta sin antes haber terminado el 4-way-handshake.

BIBLIOGRAFÍA

1. **802.11 Wireless Networks –** The Definitive guide. Matthew S. Gast.
2. **Real 802.11 Security** – Wi-Fi Protected Access and 802.11i. Jon Edney, William A. Arbaugh.
3. **LINUX Device Drivers** Third Edition- Jonathan Corbet, Alessandro Rubini, and Greg Kroah-Hartman.
4. **C++ - The complete reference. S**econd edition. Herbert Schildt.
5. **802.11i-2004. Documento bajado de www.IEEE.org.**
6. **Federal Information Processing Standars (FIPS) Publication 197.** November 26th 2001. Announcing Advanced Encryption Standard (AES).
7. **Driver Xsupplicant** bajado de www.sourceforge.net. Desarrollado en linux pero para adaptadores Atheros que presentan distinta arquitectura que el usado en el Proyecto.
8. **Hostapd-0.4.9.** Programa que posibilita llevar cabo un punto de acceso en un adaptador inalámbrico bajado de http://fenyo.net/newweb/hostapd.html.

CONCLUSIONES

Con respecto a la primera etapa, se logró que los programas funcionen de forma tal que reprodujeron exactamente el algoritmo AES.

Los tres programas ofrecen la opción de ver o no en pantalla las operaciones sobre la matriz *state,* lo que podría ser aplicado con fines pedagógicos a la enseñanza de redes inalámbricas, a modo de ejemplo o explicación adicional para ilustrar el funcionamiento del algoritmo AES.

En la segunda etapa se lograron implementar los procedimientos denominados autenticación del sistema abierto y asociación, etapas previas al intercambio de datos, quedando el driver habilitado para poder funcionar con WPAPSK – AES.

Como verificación del funcionamiento para el driver modificado para AES, los mensajes recibidos y enviados fueron testeados mediante los programas xsupplicant y hostapd, que simulan los dos extremos de una red inalámbrica, y dieron resultados positivos, confirmando que las tramas enviadas y recibidas fueron correctas.

No se logró la autenticación 4-way-handshake del mensaje por parte del punto de acceso. Ésto llevó a suponer el que el punto de acceso no obedece la teoría supuesta, y que llevara a cabo alguna otra forma de desarrollo de claves o introdujera tiempos entre tramas que no estén especificados en la bibliografía disponible.

Lamentablemente no se dispuso de una cantidad de dispositivos para intercambiarlos, observar distintos comportamientos y evaluar la situación en caso de que alguno de ellos falle.

Finalmente se desea dejar la constancia de la experiencia y conocimientos adquiridos a lo largo del Proyecto Final. El mismo, no está puramente relacionado con los conocimientos adquiridos durante la carrera de Ingeniería Electrónica. La programación en C y la introducción en el

sistema operativo Linux que se debió aprender son muy solicitadas en la industria en general, lo cual asegura su utilidad futura, dadas sus innumerables aplicaciones.