FACULTAD DE CIENCIAS

Facultad de Ciencias

Proyecto Fin de Carrera

Seguridad en redes inalámbricas: el protocolo WEP

Security on wireless networks: WEP protocol

Para acceder al Título de

INGENIERO EN INFORMÁTICA

Autor: Diego García Prieto
Director: Domingo Gómez Pérez
Codirector: Álvar Ibeas Martín

Septiembre - 2014

Seguridad en redes inalámbricas: el protocolo WEP

Resumen

La sustitución del cable de las redes LAN por el campo electromagnético que emplean las inalámbricas trae consigo un compromiso evidente para la seguridad de las comunicaciones: la tarea de "pinchar" el canal se simplifica enormemente y aumenta el riesgo de un uso no autorizado de la red.

La primera estandarización de las redes inalámbricas propuso el protocolo WEP como mecanismo de cifrado. A pesar de la rotundidad contenida en el acrónimo (*Wired Equivalent Privacy*), en la elección del criptosistema subyacente (el cifrador de flujo RC4) parecen haber pesado más los criterios de simplicidad y economía que las consideraciones de seguridad.

En este trabajo presentamos el funcionamiento del protocolo WEP, situándolo en el contexto del estándar IEEE 802.11 y detallando la formación y cifrado de los paquetes correspondientes. A continuación, estudiamos las bases de dos de los ataques criptoanalíticos que han dejado obsoleto este protocolo. Por una parte, el ataque FMS permite a un usuario ajeno a la red hacerse con la clave de acceso (*root key*), requiriendo para ello la observación y procesado de algunos millones de paquetes. Por otra, la estrategia conocida con el descriptivo nombre de "Chop-chop" permite descifrar individualmente los paquetes WEP (aunque no consigue filtrar la clave secreta del criptosistema) mediante la interacción con el punto de acceso.

Estos dos métodos constituyen un buen punto de partida para el estudio del criptoanálisis de redes inalámbricas, que cuenta con un variado repertorio de técnicas. El primero abrió la vía de ataque principal del cifrador RC4. En cuanto a Chop-Chop, parece haber resistido a las contramedidas de seguridad introducidas en el protocolo WPA —sucesor de WEP—. En la actualidad, el estándar de seguridad (conocido como WPA2) se basa en AES, un criptosistema simétrico más complejo y seguro. En este trabajo no abordamos el funcionamiento de los sucesores de WEP.

Keywords: WEP, 802.11, RC4, criptoanálisis, seguridad, redes, ataques, FMS, Chop-Chop

---

The substitution of the electromagnetic field employed by wireless networks for the cable of LANs brings out an obvious compromise in communication security: the task of peeping at the channel gets enormously simplified and the risk of unauthorized network usage increases.

The first standardization of wireless networks proposed the WEP protocol as encryption method. Despite the orotundity behind the acronym (Wireless Equivalent Privacy), simplicity and economy criteria seems to have outweighed security matters in the choice of its underlying cryptosystem (the RC4 stream cipher).

In this report we present the operation of WEP protocol, in the context of the IEEE 802.11 standard, and detail the way of composing and encrypting the corresponding packages. We study then the bases of two of the cryptanalytical attacks that have made this protocol obsolete. On the one hand, FMS attack permits an user external to the network to obtain the root key, requiring for that to eavesdrop and process several millions of packages. On the other hand, the approach known under the descriptive term "Chop-Chop" permits individually decrypting WEP packages (although it does not leak the cryptosystem secret key) by interaction with the access point.

These two methods make up a good starting point for the study of wireless network cryptanalysis, that involves a varied assortment of techniques. The former opened up the main approach to attack the RC4 cipher. As for Chop-Chop, it seems to have resisted the security countermeasures introduced in WPA protocol—WEP successor—. Currently, the security standard of wireless networks (known as WPA2) is based on AES, a more complex and secure symmetric cipher. In this report we do not address the operation of WEP successors.

Keywords: WEP, 802.11, RC4, cryptanalysis, security, networks, attacks, FMS, Chop-Chop

Abstract

Security on wireless networks: WEP protocol

Índice

Agradecimientos **1**

Introducción **3**

1 El criptosistema RC4 **5**
1.1 *One-time pad* 5
1.2 Cifrado en flujo y cifrado en bloque 8
1.3 Descripción 9
1.4 Generación del *keystream* 10

2 Wired Equivalent Privacy **17**
2.1 Estándar IEEE 802.11 17
2.2 Comprobación de redundancia cíclica 20
2.3 Descripción 24
2.4 Proceso de cifrado 25
2.5 Autenticación 26
2.6 Debilidades 27

3 Dos estrategias de criptoanálisis **29**
3.1 Ataque FMS 29
3.2 Ataque Chop-Chop 35

A Generador del *keystream* para RC4 **41**

B Obtención del *checksum* CRC32 **47**

Agradecimientos

Mis agradecimientos a los directores Domingo Gómez y Álvar Ibeas por su grandísima colaboración en la realización de este trabajo. No he podido elegir mejores profesores. ¡Bien hecho! y muchas gracias.

Introducción

Cuando aparecieron las primeras redes de comunicación inalámbricas en los años 90, también surgió el problema de su seguridad. Ya que el medio de transmision es el aire, todos los paquetes emitidos pueden ser capturados por cualquier estación aunque no sea su destino. Ante la imposibilidad física de cercar el aire, se creó el protocolo WEP para proteger los paquetes con una clave y sólo permitir su lectura a los usuarios que posean dicha clave.

El protocolo WEP fue publicado en el estándar IEEE 802.11 [802.11] en 1997. Su idea era proporcionar la misma seguridad como si de una red cableada se tratase, de ahí su nombre, Wired Equivalent Privacy, sin embargo pronto se vieron ataques que lo dejaron en entredicho. En su ciclo de vida, WEP ha sufrido muchos ataques de diferentes tipos (fuerza bruta, reutilización de *keystreams*, debilidad de IV en RC4 y explotación del sistema de detección de errores), motivando en la industria arreglos para cada ocasión hasta que se decidió que WEP ya no era seguro. En el estándar IEEE 802.11i de 2004 se reemplazó por WPA, y este posteriormente por WPA2, ambos más seguros. A pesar de esto, aún es habitual su uso en redes domésticas [Bittau *et al.*, 2006].

El proceso de publicación de ataques a WEP ilustra cómo la criptología se construye de manera empírica: una herramienta es buena y se emplea hasta que se hace público un fallo en su seguridad, que a su vez motiva el perfeccionamiento de la herramienta.

La criptología se divide en dos ramas: la criptografía se encarga de diseñar criptosistemas, y el criptoanálisis de atacarlos. El avance de cada una de las dos ramas de la criptología se apoya en el de la otra. A pesar de la popularidad de algunos sistemas como el RC4, no se cuenta con criptosistemas prácticos de los que se sepa con rigor que son seguros. Existe una gran variedad de criptosistemas, de uso muy corriente, cuya seguridad se basa en hechos que se suponen ciertos.

En el último cuarto del siglo pasado, la criptología experimentó un salto cualitativo, al inventarse los criptosistemas asimétricos (de clave pública), que rompen con el esquema simétrico tradicional en que dos comunicantes comparten una clave secreta. Estos sistemas satisfacen las necesidades modernas de comunicación, al conectar a un número elevado de participantes, proporcionando confidencialidad dos a dos. También son útiles para otros fines no relacionados estrictamente con la transferencia de mensajes, como las votaciones electrónicas o la compartición de secretos. Un inconveniente de los criptosistemas de clave pública estriba en un mayor coste computacional del proceso de cifrado. Por este motivo, la forma de actuación habitual consiste en su uso para la comunicación de claves secretas correspondientes a criptosistemas tradicionales (simétricos), a los que se sigue confiando el cifrado de los mensajes por ser más eficientes.

En el caso de WEP, objeto de estudio de esta memoria, cabe destacar dos estrategias de criptoanálisis. Por una parte, la propuesta de Fluhrer *et al.* [2001] (conocida como FMS) abrió la puerta a toda una familia de ataques basados en la misma idea. WEP prescribe que parte de la clave secreta sea distinta en cada paquete y se transmita en claro. Este hecho unido a distintas correlaciones que se han encontrado entre los primeros bytes de la clave

y el primer byte del texto en claro permite a este tipo de ataques revelar la parte oculta de la clave secreta. Por otra parte, la idea que publicó KoreK [2004c] no recuperaba la clave pero sí el contenido de cualquier paquete apoyándose en el código detector de errores. Esta idea (conocida como "ataque Chop-Chop") no sólo se ha mantenido en WEP sino que se ha utilizado para atacar a su sucesor WPA.

Es posible aprovechar estas y otras técnicas para utilizar sin autorización redes protegidas por WEP aun sin contar con conocimientos informáticos ni de criptoanálisis. Cabe destacar la recopilación de [Aircrack-ng], que acerca al gran público la mayoría de las herramientas que se conocen.

Este trabajo se divide en tres capítulos. En el primero se estudia el criptosistema en que se basa WEP: el cifrador de flujo RC4. La comprensión de los detalles del proceso pseudoaleatorio que genera la máscara de cifrado es necesaria para abordar el ataque FMS. A continuación, el segundo capítulo presenta el protocolo WEP y, en especial, los distintos campos que componen los paquetes 802.11. Nos detenemos en el mecanismo de detección de errores CRC32, en este caso por su importancia en el planteamiento de Chop-Chop. Por último, en el tercer capítulo se emplea la información anterior para explicar los dos ataques-modelo. Concluye la memoria con dos herramientas programadas en Python: la primera, de utilidad didáctica, permite simular el proceso de generación de la máscara de cifrado o *keystream* de RC4; la segunda, una implementación simplificada (sin pretender que sea eficiente) para el cálculo del *checksum* CRC32, también con propósitos didácticos, comprueba que los paquetes que circulan en una red contienen un *checksum* conforme a sus especificaciones.

Capítulo 1

El criptosistema RC4

RC4 es el criptosistema en el que se basa WEP. Pertenece a una familia de cifradores muy eficientes (tanto el cifrado como el descifrado tienen un coste computacional muy bajo y una implementación sencilla en *hardware* y en *software*) que derivan del famoso criptosistema *One-time pad*.

1.1 *One-time pad*

One-time pad (OTP) es un criptosistema simétrico que cifra la comunicación entre dos puntos con una clave aleatoria, secreta y no reutilizable, acordada por ambas partes y tan larga como cada mensaje. Previamente se concierta una clave entre ambos puntos de forma segura y en otra fase se procede al cifrado del mensaje para su transmisión. Este proceso consiste en aplicar la operación XOR, bit a bit, entre el texto claro y la clave. Posee la propiedad de ser irrompible.

Es una evolución del cifrado de Vernam [1919] para AT&T y la comunicación segura en teletipos. Cada mensaje se cifraba eléctricamente con una clave leída de un lazo de cinta de papel. Esta técnica no era irrompible debido a que, cumplido el ciclo del lazo, se repetía la clave.

Shannon [1949], en su desarrollo de la teoría de la información, demostró que *One-time pad* presentaba la propiedad de *secreto perfecto*:

> *"It is natural to define 'perfect secrecy' by the condition that, for all E the 'a posteriori' probabilities are equal to the 'a priori' probabilities independently of the values of these). In this case, intercepting the message has given the cryptanalyst no information."*[1]

Para una comunicación entre dos puntos, si un atacante captura un mensaje cifrado, no le aporta más información sobre el mensaje en claro que si nunca lo hubiese capturado. La única información que puede conseguir es la longitud del texto si no se han usado mecanismos de compresión o de engordamiento.

Requisitos para que OTP cumpla el *secreto perfecto*:

- La clave debe ser aleatoria.
- La generación y la distribución de la clave debe ser segura.
- La totalidad de la clave debe permanecer secreta.
- La clave no se debe reutilizar.

[1] E es el criptograma interceptado.

Ni con recursos computacionales ilimitados se lograría recuperar el mensaje original. Interceptado un mensaje cifrado de tamaño 4 bytes (4 caracteres ASCII) se puede probar con todas las posibles claves aleatorias dando como mensaje en claro cualquier palabra de 4 caracteres. Es 100% irrompible, sin embargo poco eficiente para envíos masivos de datos. En cambio en RC4 no se necesita una clave para cada mensaje y la seguridad que proporciona es muy inferior. En implementaciones iniciales de OTP la clave se comunicaba en un trozo de papel, facilitando su destrución. Incluso la KGB usaba minúsculos papeles tratados químicamente para arder y no dejar cenizas.

Este sistema también se usó en el famoso teléfono rojo entre Washington DC y Moscú en la guerra fría. La clave se acordaba en reuniones personales obteniendo una cinta muy larga. Esta cinta se usaba para cifrar las conversaciones entre ambos presidentes, cifrando cada palabra con un cierto fragmento de la cinta y no volviéndose a usar.

En la actualidad, como se ha mencionado antes, no es eficiente su uso en las comunicaciones modernas, en las que el envío de datos es masivo. El acuerdo de clave secreta entre ambos puntos de una comunicación para cada mensaje enviado es muy costoso. Por ello no se ha adoptado como una herramienta generalizada de seguridad, pero muchos sistemas lo toman como base para evaluar su compromiso entre seguridad perfecta y complejidad computacional.

Un dato interesante sobre este sistema es la paradoja que presenta. Si existe un canal seguro por donde se envía la clave, entonces ¿por qué no comunicarse por ese canal? La razón es que este canal es lento y requiere una preparación previa a la comunicación, en cambio, el canal inseguro es inmediato y permite comunicarse a distancia en cualquier momento.

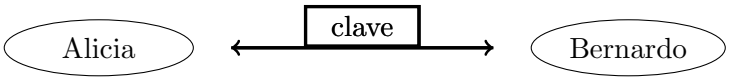
El problema del acuerdo de las contraseñas por un canal inseguro derivó en la invención de los criptosistemas de clave pública (criptografía asimétrica) [Diffie y Hellman, 1976]. Mediante estas técnicas se puede establecer una comunicación segura a través de un canal inseguro. Cada uno de los puntos que se comunican tiene dos claves: una pública y una privada. La clave pública de un punto sirve para que cualquiera pueda cifrar mensajes dirigidos a él, mientras que la privada, nunca expuesta, sirve para descifrar los mensajes recibidos. Un atacante puede escuchar el canal sin tener acceso a la información que se está compartiendo si no cuenta con recursos computacionales desmesurados. Los procesos de cifrado y descifrado en la criptografía de clave pública presentan un mayor coste computacional que los métodos tradicionales, por lo tanto, se emplean fundamentalmente para el envío seguro de una clave secreta por un medio inseguro, para a continuación realizar la comunicación mediante un criptosistema simétrico.

1.1.1 Ejemplo

Alicia y Bernardo quieren enviarse mensajes entre ellos, pero no quieren que nadie sepa su contenido. Para ello, siguen los siguientes pasos[2]:

1. Antes de la comunicación, Alicia y Bernardo acuerdan una clave aleatoria muy larga de forma segura.

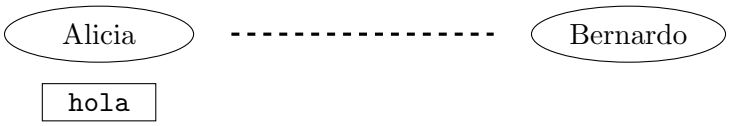
$$\text{clave} = 00101010001001001101011010101101111 10\ldots$$



[2] ASCII es la codificación usada.

2. Alicia quiere enviar el mensaje `hola`.

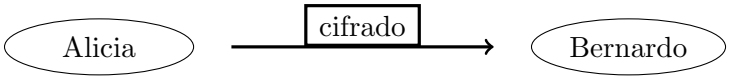
$$\text{hola} = \texttt{68} : \texttt{6F} : \texttt{6C} : \texttt{61} = 01101000011011110110110001100001$$



3. Alicia cifra el mensaje `hola` mediante la operación XOR con la clave. `hola` son 4 bytes así que le corresponden 4 bytes de clave (32 bits).

$$\texttt{hola} \oplus \text{clave32} = 01000010010010111011101011001100 = \texttt{42} : \texttt{4B} : \texttt{BA} : \texttt{CC}$$

4. Alicia envía a Bernardo el mensaje cifrado.



5. Bernardo recibe el mensaje y lo descifra mediante la operación XOR con la clave. El mensaje cifrado son 4 bytes así que le corresponden 4 bytes de clave (32 bits).

$$\text{cifrado} \oplus \text{clave32} = 01101000011011110110110001100001$$

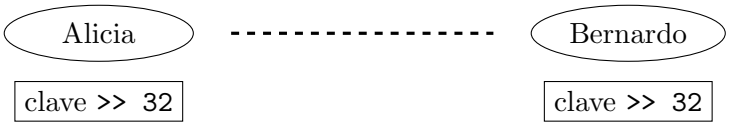
6. Bernardo lee el mensaje: `hola`.

$$01101000011011110110110001100001 = \texttt{68} : \texttt{6F} : \texttt{6C} : \texttt{61} = \texttt{hola}$$

7. Alicia y Bernardo eliminan los 32 bits usados para el cifrado/descifrado.

$$\text{clave} = 00101010001001001101011010101101 \| 11110\ldots$$

$$\text{clave} = 11110\ldots$$



Carlos, para averiguar qué le ha dicho Alicia a Bernardo, captura el mensaje cifrado. Si Alicia y Bernardo han elegido una buena clave, aleatoria, secreta y desechable, el mensaje capturado no aportará más información a Carlos de la que tenía sin el mensaje interceptado. Por ejemplo, se supone que antes de capturar el mensaje Carlos tenía un 50 % de probabilidad de saber si el bit de la posición tercera del mensaje en claro era un 1 ó un 0, pero una vez capturado el mensaje cifrado, mantiene la misma probabilidad.

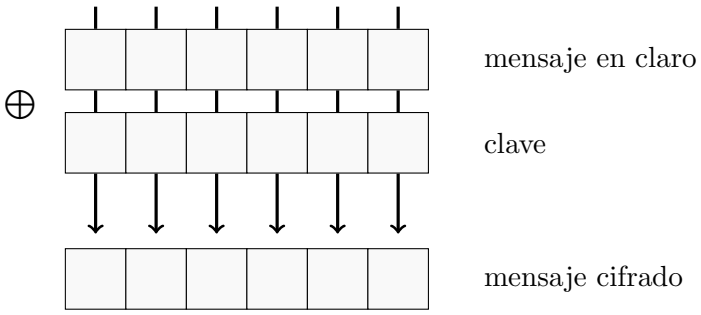
1.2 Cifrado en flujo y cifrado en bloque

Un criptosistema es un algoritmo de cifrado que oculta información para protegerla. Según el modelo tradicional (simétrico), para una comunicación secreta entre dos puntos, se acuerda una clave común, y en base a ella, el emisor cifra un mensaje, lo envía al destinatario, y este lo descifra con la misma clave. Su objetivo es evitar la lectura de los datos por parte de intrusos.

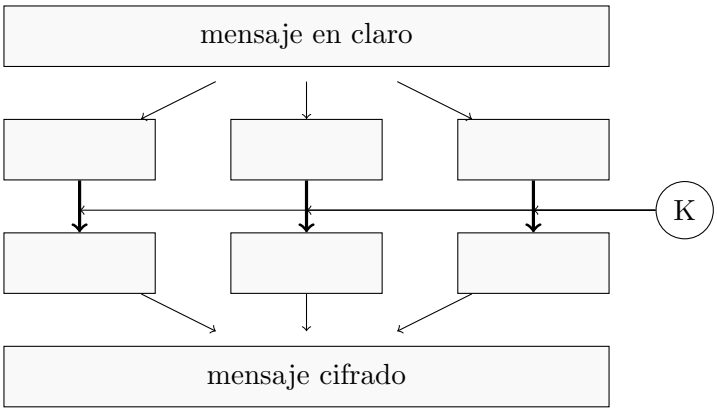
Un criptosistema de cifrado en flujo (*stream cipher*) es un criptosistema simétrico que imita el funcionamiento de *One-time pad*, sustituyendo la clave secreta aleatoria por una sucesión de bits que se obtiene a partir de una clave secreta finita mediante un proceso pseudoaleatorio. Como hemos explicado en la sección 1.1, la información se combina con la clave, bit a bit, mediante la operación XOR, generando su cifrado. La acción de descifrar se lleva a cabo con la misma operación XOR.

RC4 es el cifrador de flujo empleado para aportar seguridad en WEP.

Según eSTREAM los cifradores en flujo recomendados son *HC-128*, *Rabbit* y *Salsa20/12*. HC-128 y Rabbit están incluido en la nueva versión de CyaSSL, un protocolo ligero para la implementación de SSL/TLS orientada a sistemas embebidos y de código libre.



Un criptosistema de cifrado en bloque (*block cipher*) es un algoritmo que divide la información en bloques y los cifra individualmente, para finalmente agrupar todos los trozos. Para cada bloque se genera su cifrado, que tiene el mismo tamaño, mediante una transformación determinada por una clave.



En la actualidad, el *block cipher* más popular es AES[3], empleado en el protocolo WPA2 según el estandar IEEE 802.11i. El tamaño de los bloques es de 128 bits como mínimo. Fue desarrollado por dos criptólogos belgas, J. Daemen y V. Rijmen, y presentado con el nombre "Rijndael" a un concurso realizado por NIST[4] en 1997. Se buscaba un nuevo algoritmo de cifrado capaz de proteger información sensible para sustituir al criptosistema DES. En la competición, *Serpent* quedó en segunda posición y *Twofish* en tercera.

Para aplicaciones de tiempo real que requieren de un cifrado incremental, como por ejemplo, el cifrado de conversaciones telefónicas, un *stream cipher* es eficaz. En cambio, un *block cipher* no lo es en este tipo de aplicaciones ya que necesita tiempo para completar un bloque a cifrar, lo cual limita la inmediatez. Serían provechosos en aplicaciones donde se transmite grandes cantidades de información en un tiempo menos estricto.

1.3 Descripción

RC4 es un *stream cipher* muy extendido por su simplicidad y rapidez. Implementado en el protocolo de seguridad WEP para cifrar la información transmitida por redes inalámbricas, en su versión posterior WPA, y también usado en los protocolos SSL/TLS de la capa de transporte de la arquitectura de protocolos TCP/IP, para la comunicación segura punto a punto.

Al ser un cifrado de flujo, como se ha explicado en la sección 1.2, el proceso de cifrado es muy sencillo. para definirlo, se debe explicar cómo se genera la máscara o *keystream*. Se hará en la sección 1.4.

El procedimiento es el siguiente: para una clave corta de entrada K, conocida por los usuarios, se genera una cadena pseudoaleatoria de bytes del mismo tamaño que el mensaje a cifrar/descifrar. La cadena keystream X resultante en combinación con un mensaje claro M o cifrado $\tilde{M}$, mediante la operación XOR, da lugar al mensaje cifrado o claro, respectivamente.

$$X = RC4(K)$$

[3]Advanced Encryption Standard
[4]Instituto Nacional de Estándares y Tecnología de EE.UU.

$$M \oplus X \rightarrow \tilde{M}$$
$$\tilde{M} \oplus X \rightarrow M$$

Su nombre es el acrónimo de Rivest Cipher 4 debido a que fue diseñado por Ronald L. Rivest, miembro de la empresa RSA Security, en el año 1987. También es conocido por Ron's Code.

Fue un secreto registrado hasta que en 1994 se publicó anónimamente en Internet [Cypherpunks]. A día de hoy, no se ha producido la liberación oficial del criptosistema. Aunque la implementación no oficial de RC4 es legal, al ser una marca registrada, no puede ser utilizada con el nombre RC4, por lo que es posible encontrarlo con otros nombres como ARCFOUR, ARC4 o Alleged_RC4. La implementación en Python se llama ARC4 y se encuentra en la librería Crypto.

Para el uso de este criptosistema secreto, el estándar 802.11 recomendaba obtener los detalles de su implementación directamente de la compañía propietaria:

"WEP uses the RC4 PRNG algorithm from RSA Data Security Inc. Details of the RC4 algorithm are available from RSA. Please contact RSA for algorithm details and the uniform RC4 licensee terms that RSA offers to anyone wishing to use RC4 for the purpose of implementing the IEEE 802.11 WEP option."

Fluhrer *et al.* [2001] publicaron un informe donde se demostraba su debilidad. No se demostraba en el estudio, pero debido a que WEP se basa en RC4, se intuía que WEP también era débil. Posteriormente, un ataque apoyado en las ideas del estudio demostró que WEP ya no era seguro.

Aunque no está recomendado su uso en la actualidad, algunos sistemas basados en RC4 ya implementados son seguros para un uso común. Su seguridad depende de la implementación y su utilización.

1.4 Generación del *keystream*

Generar la sucesión pseudoaleatoria o *keystream* es un algoritmo sencillo que consta de dos más simples: Key Scheduling Algorithm (KSA, algoritmo 1) y Pseudo-Random Generation Algorithm (PRGA, algoritmo 2).

Este algoritmo maneja una permutación de 256 elementos, es decir, una lista donde aparece cada uno de los números del rango $[0, \ldots, 255]$ una sola vez. El algoritmo RC4-KSA desordena la permutación de forma pseudoaleatoria según una clave entrante. Una vez desordenada la permutación, el algoritmo RC4-PRGA toma valores de posiciones pseudoaleatorias para devolver una cadena de bytes de la misma longitud que el mensaje a cifrar/descifrar.

1. Key Scheduling Algorithm (RC4-KSA)

   El primer algoritmo inicializa la permutación a la identidad, es decir, cada valor es igual a su posición. A continuación, se entra en un bucle donde se calculan las posiciones a intercambiarse dentro de la permutación según la permutación del momento y la clave entrante. Algoritmo 1.

2. Pseudo-Random Generation Algorithm (RC4-PRGA)

   El segundo algoritmo es un bucle dependiente del tamaño del mensaje a cifrar/descifrar, que vuelve a mezclar la permutación, y retorna el valor de una posición como byte i-ésimo de X. Algoritmo 2.

Algoritmo 1: Key Scheduling Algorithm (RC4-KSA)

```
for i = 0, ..., 255 do
    S[i] ← i
end
j ← 0
for i = 0, ..., 255 do
    j ← (j + S[i] + K[i mod len(K)]) mod 256
    transponer(S, i, j)
end
```

Algoritmo 2: Pseudo-Random Generation Algorithm (RC4-PRGA)

```
i ← 0
j ← 0
for n = 0, ..., len(mensaje) − 1 do
    i ← (i + 1) mod 256
    j ← (j + S[i]) mod 256
    transponer(S, i, j)
    X[n] ← S[(S[i] + S[j]) mod 256]
end
```

1.5 Ejemplo

En esta sección se muestra un ejemplo del funcionamiento de RC4. Se desea hallar un keystream X generado por RC4 a partir de la clave $K = \texttt{71:01:00:12:34:56:78:90}$ para cifrar el mensaje `hola`.[5]

El ejemplo se realizará mediante un pequeño programa (Anexo A) que muestra el procedimiento paso a paso[6]:

1. Una permutación de tamaño 256 [00, . . . , FF] se inicializa a la identidad, es decir, en la posición 00 corresponde el valor 00, en la posición 01 corresponde el valor 01. . . Ver línea 2 del algoritmo 1. La permutación identidad resultante se ve en la ilustración 1.1.

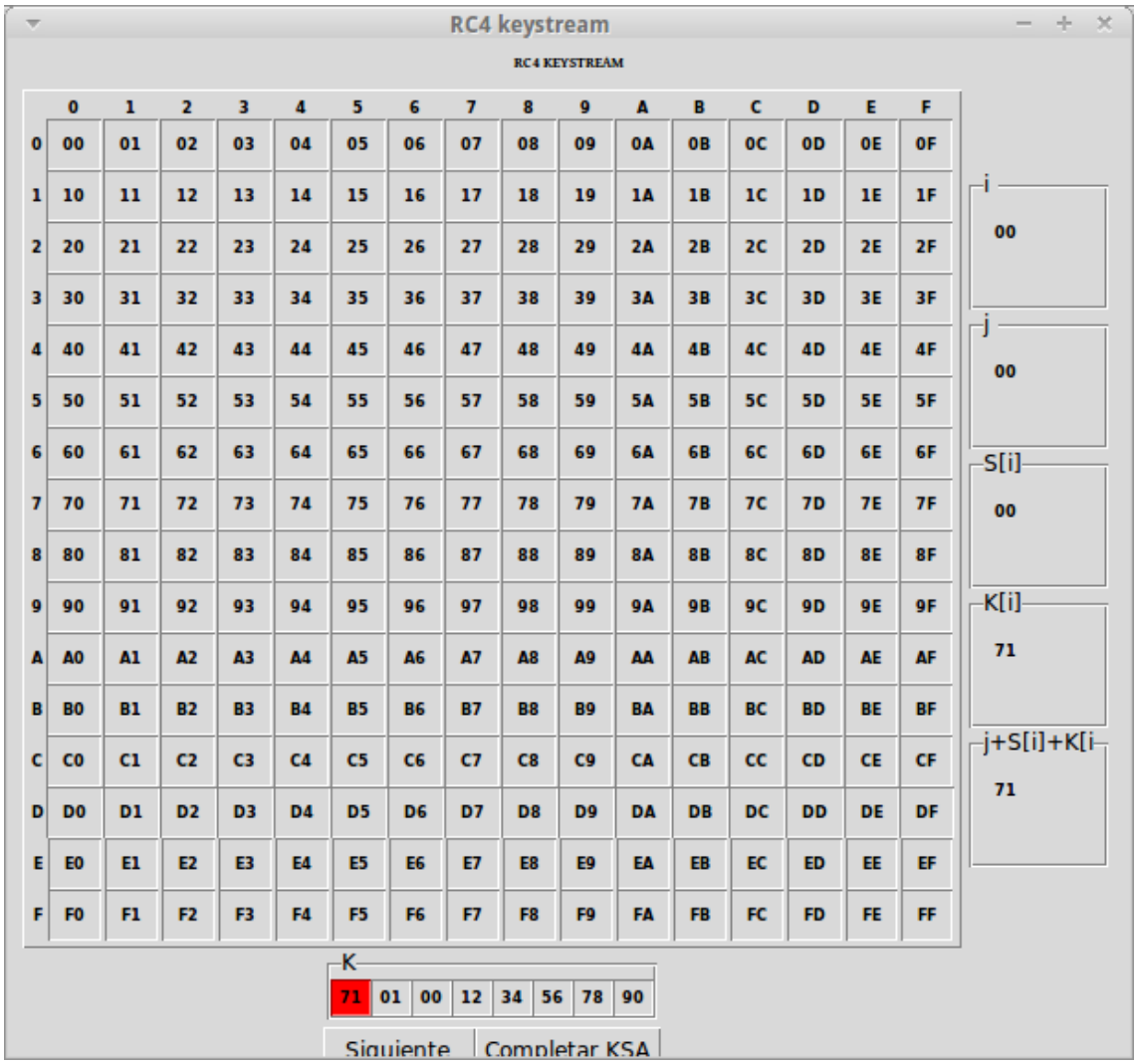


Ilustración 1.1: Permutación inicializada a la identidad.

[5] Utilizamos notación en hexadecimal: 71, se corresponde con el valor hexadecimal 0x71=113.

[6] En las imagenes, las casillas rojas son los valores que van a ser usados y las casillas amarillas los valores ya usados.

2. Se intercambia el valor de la primera posición de la permutación, la posición 00, con el valor de otra posición elegida de forma pseudoaleatoria. Ver líneas 6 y 7 del algoritmo 1.

$$i = 00$$

$$j = 00$$

$$K[i] = 71$$

$$\begin{aligned} j &\leftarrow (00 + S[00] + K[00 \bmod 8]) \bmod 256 \\ &= (00 + 00 + 71) \bmod 256 = 71 \end{aligned}$$

La posición 00 se intercambia con la posición 71. Ver ilustración 1.2.

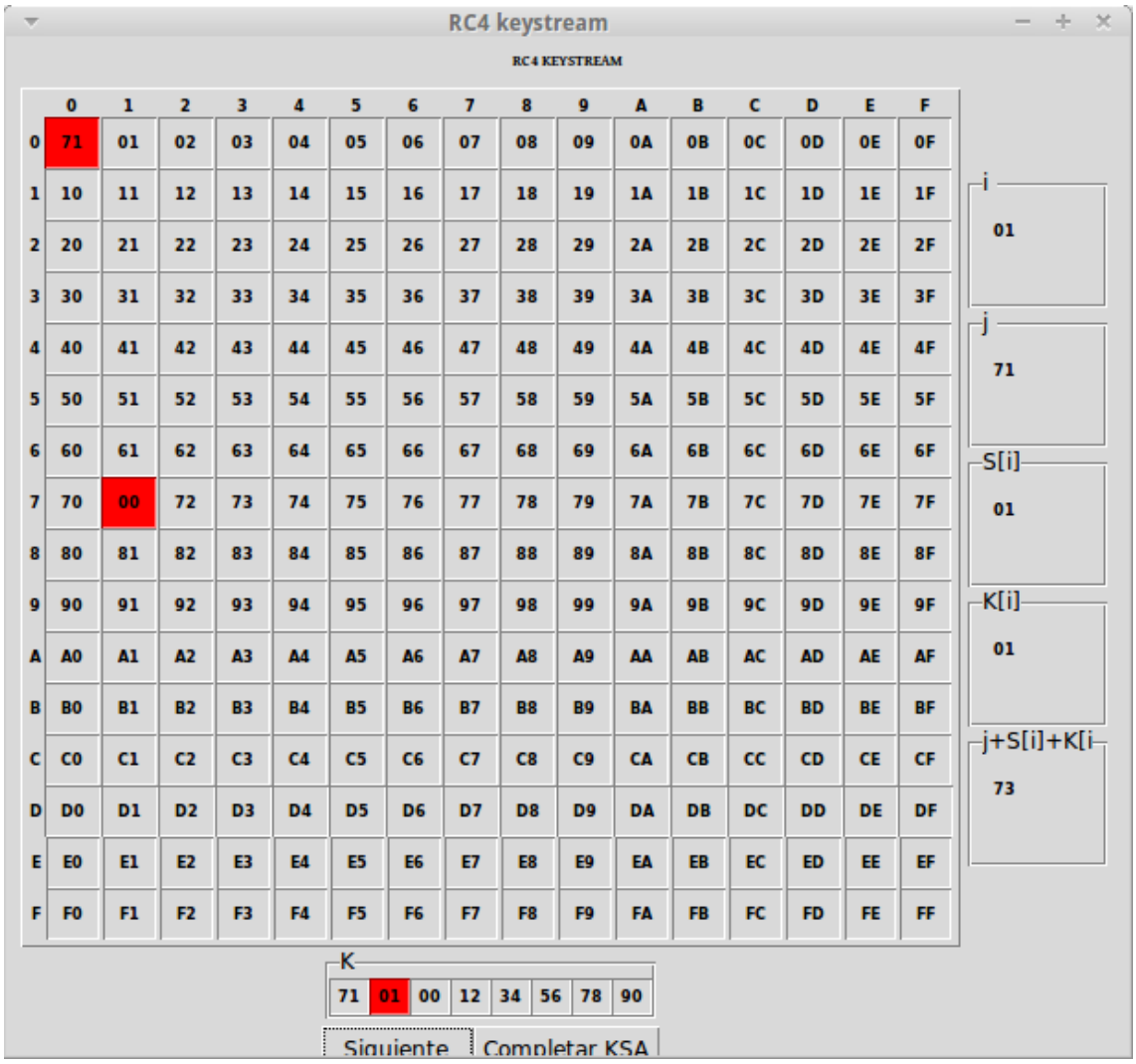


Ilustración 1.2: Permutación en el ciclo 1.

3. Se intercambia la posición 01 con otra.

$$i = 01$$

$$j = 71$$

$$K[i] = 01$$

$$\begin{aligned} j &\leftarrow (71 + S[01] + K[01 \bmod 8]) \bmod 256 \\ &= (71 + 01 + 01) \bmod 256 = 73 \end{aligned}$$

La posición 01 se intercambia con la posición 73. Ver ilustración 1.3.

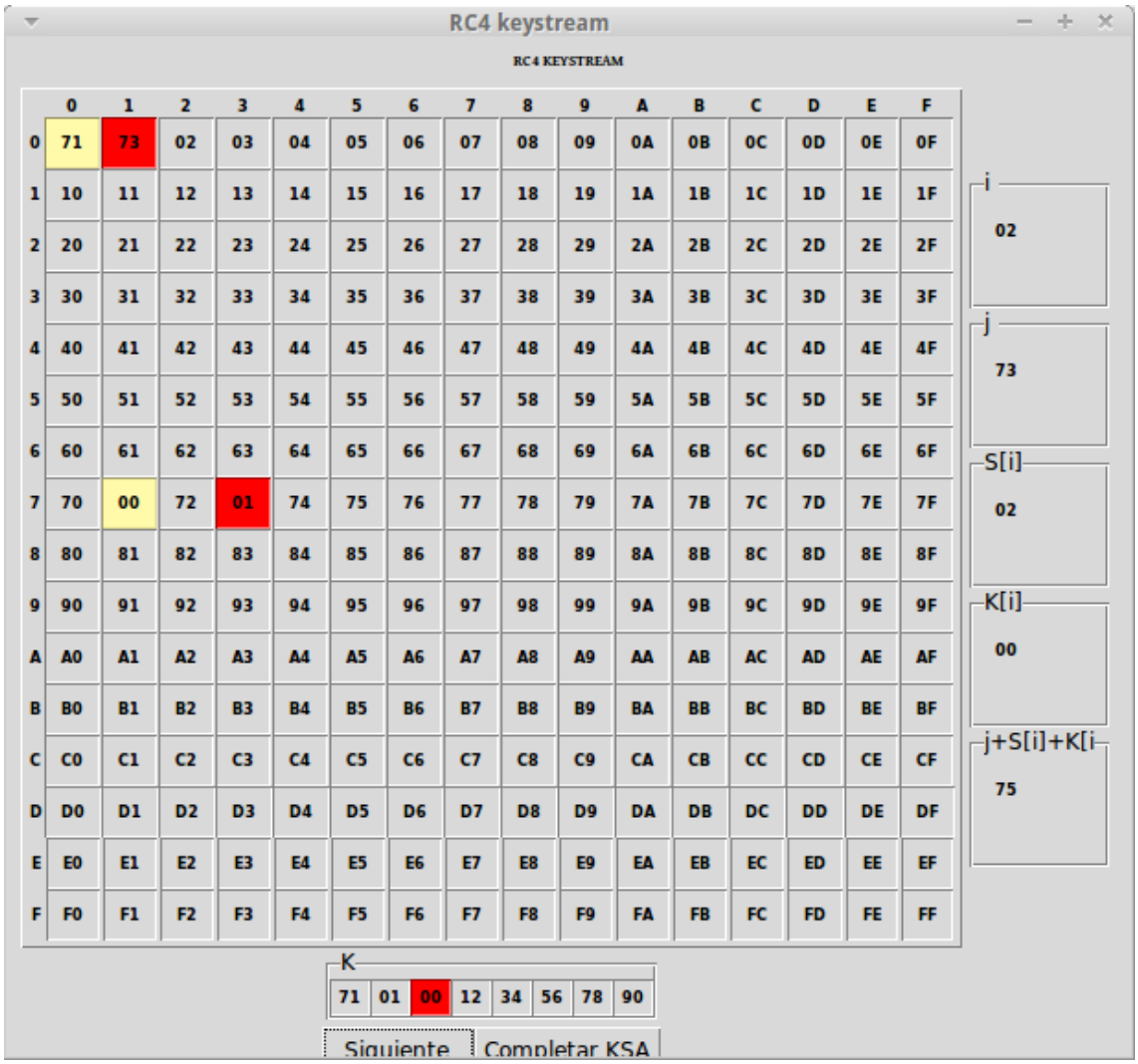
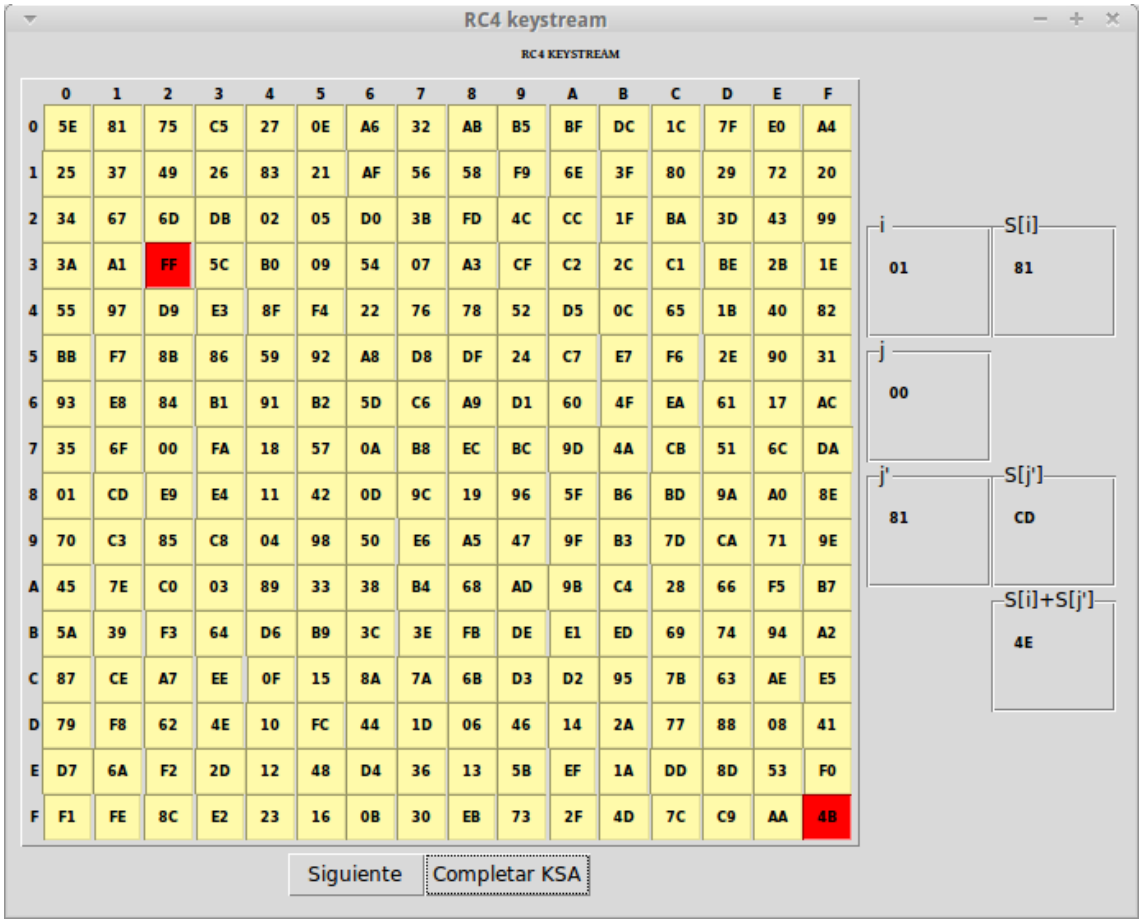


Ilustración 1.3: Permutación en ciclo 2

4. Así sucesivamente hasta la posición `FF` y la permutación queda totalmente desordenada. Ver resultado final en la figura 1.4.



	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	5E	81	75	C5	27	0E	A6	32	AB	B5	BF	DC	1C	7F	E0	A4
1	25	37	49	26	83	21	AF	56	58	F9	6E	3F	80	29	72	20
2	34	67	6D	DB	02	05	D0	3B	FD	4C	CC	1F	BA	3D	43	99
3	3A	A1	FF	5C	B0	09	54	07	A3	CF	C2	2C	C1	BE	2B	1E
4	55	97	D9	E3	8F	F4	22	76	78	52	D5	0C	65	1B	40	82
5	BB	F7	8B	86	59	92	A8	D8	DF	24	C7	E7	F6	2E	90	31
6	93	E8	84	B1	91	B2	5D	C6	A9	D1	60	4F	EA	61	17	AC
7	35	6F	00	FA	18	57	0A	B8	EC	BC	9D	4A	CB	51	6C	DA
8	01	CD	E9	E4	11	42	0D	9C	19	96	5F	B6	BD	9A	A0	8E
9	70	C3	85	C8	04	98	50	E6	A5	47	9F	B3	7D	CA	71	9E
A	45	7E	C0	03	89	33	38	B4	68	AD	9B	C4	28	66	F5	B7
B	5A	39	F3	64	D6	B9	3C	3E	FB	DE	E1	ED	69	74	94	A2
C	87	CE	A7	EE	0F	15	8A	7A	6B	D3	D2	95	7B	63	AE	E5
D	79	F8	62	4E	10	FC	44	1D	06	46	14	2A	77	88	08	41
E	D7	6A	F2	2D	12	48	D4	36	13	5B	EF	1A	DD	8D	53	F0
F	F1	FE	8C	E2	23	16	0B	30	EB	73	2F	4D	7C	C9	AA	4B

Ilustración 1.4: Permutación en el ciclo 255.

5. Se retorna el primer byte del keystream X.

$$i = \texttt{01}$$
$$j = S[i] = S[\texttt{01}] = \texttt{81}$$

$S[\texttt{01}]$ y $S[\texttt{81}]$ son intercambiados.

$$S[S[\texttt{01}] + S[\texttt{81}] \bmod 256] = S[(\texttt{CD} + \texttt{81}) \bmod 256] = S[\texttt{14E} \bmod 256] = S[\texttt{4E}] = \texttt{40}$$

El primer byte de X es `40`.

6. Así sucesivamente hasta obtener 4 bytes de X debido a que el mensaje `hola` tiene 4 bytes.

$$X = \texttt{40} : \texttt{01} : \texttt{B2} : \texttt{19}$$

7. Ahora se puede cifrar el mensaje con la operación XOR.

$$M \oplus X = \tilde{M}$$
$$\texttt{68} : \texttt{6F} : \texttt{6C} : \texttt{61} \oplus \texttt{40} : \texttt{01} : \texttt{B2} : \texttt{19} = \texttt{28} : \texttt{6E} : \texttt{DE} : \texttt{78}$$

Capítulo 2

Wired Equivalent Privacy

El protocolo WEP[1] es el protocolo de cifrado para redes inalámbricas inicialmente propuesto en el estándar IEEE 802.11 que permite ofrecer seguridad de acceso a la red y proteger la información transmitida. Debido a la naturaleza del medio de transmisión los mensajes intercambiados dentro de una red inalámbrica son más susceptibles de ser interceptados, por lo que este protocolo fue pensado para proporcionar la misma confidencialidad que existe en una red de computadores conectados mediante cable. WEP proporciona confidencialidad a través del uso del criptosistema RC4, explicado en el capítulo 1 e integridad a través del sistema CRC32, que se explicará en el sección 2.2.

La clave secreta para el RC4 consiste en una *root key* (Rk), que comparten todos los usuarios autorizados, a la que se añade un vector de inicialización (IV) que cambia en cada paquete que se transmite.

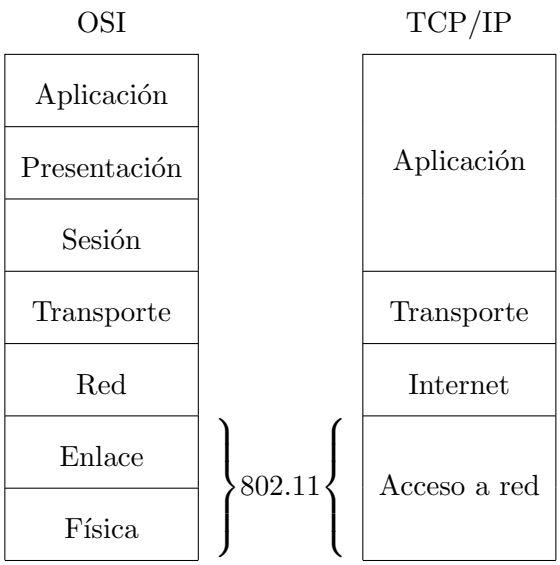
Para comprobar la integridad de los datos, cada paquete tiene un campo llamado *Integer Check Value* (ICV) que es el *checksum* de los datos que se envian en el paquete. Tanto los datos del paquete como el ICV están cifrados utilizando el algoritmo RC4.

2.1 Estándar IEEE 802.11

El estándar IEEE[2] 802.11 define las especificaciones de la capa física y la capa de enlace de los dos primeros niveles de la arquitectura de protocolos OSI para redes inalámbricas locales (WLAN) y metropolitanas (WMAN). Asimismo estas capas corresponden a la primera capa de acceso a red de la arquitectura de protocolos TCP/IP. Se encuentra dentro de la familia de estándares 802.x, que proponen protocolos orientados al acceso a redes LAN y MAN.

[1] Wired Equivalent Privacy (privacidad equivalente a cableado).

[2] El Instituto de Ingeniería Eléctrica y Electrónica (Institute of Electrical and Electronics Engineers, IEEE) es una asociación mundial de técnicos e ingenieros dedicada a la estandarización y al desarrollo en áreas técnicas. [IEEE]



2.1.1 Conceptos

Estos conceptos son necesarios para comprender las redes inalámbricas.

- Estación: puede ser un computador o un dispositivo de interfaz de red (*bridge*, *hub*, *switch*, *router*).
- Medio de transmisión: es el canal por el que circula la información. El medio de transmisión en el estándar IEEE 802.11 pueder tanto radiofrecuencia como infrarrojos.
- Punto de acceso (AP): dispositivo de interfaz que permite a estaciones conectarse a la red inalámbrica.
- Dominio de difusión: Equivalente a un área de comunicación. Dos estaciones deben pertenecer al mismo dominio de difusión para comunicarse.
- Conjunto de servicio básico (BSS): grupo de estaciones con capacidad para intercomunicarse entre si y que comparten un identificador de red (BSSID).
- Conjunto de servicio extendido (ESS): conjunto de varios BSS interconectados. Las estaciones de ambas redes se comunican entre ellas a través de los AP de sus subredes, y estas se comunica a través de una red de distribución (inalámbrica o no). Todos los AP comparten un identificador de red común (ESSID).
- ESSID: identificador alfanumérico de máximo 32 caracterés asignado a una red.
- BSSID: dirección física MAC[3] para identificar los paquetes pertenecientes a un BSS determinado y asociado exclusivamente a un AP.

2.1.2 Versiones

A continuación se muestra la tabla 2.1.2 con información relevante de los estándares IEEE 802.11 más populares.

[3]Una dirección MAC (Media Access Control) es un identificador de 48 bits que corresponde de forma única a un dispositivo de red.

Protocolo	Año	Frecuencia	Ancho de banda	Velocidad Máxima	Observaciones
802.11	1997	2.4 GHz	20 MHz	2 Mbps	WEP
802.11b	1999	2.4 GHz	20 MHz	11 Mbps	-
802.11a	1999	5 GHz	20 MHz	54 Mbps	-
802.11g	2003	2.4 GHz	20 MHz	54 Mbps	-
802.11i	2004	2.4 GHz	20 MHz	54 Mbps	WPA2
802.11n	2007	2.4-5 GHz	20-40 MHz	300 Mbps	

Table 2.1: Versiones de IEEE 802.11

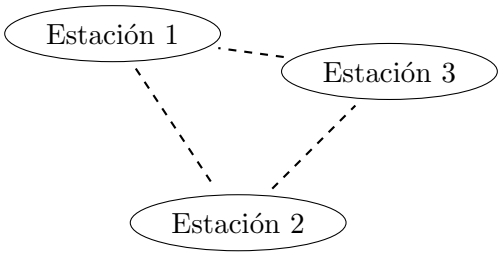
En el estándar 802.11i no se mejoraron las características técnicas de la capa física, se propusieron una serie de enmiendas para mitigar el efecto de los ataques en la seguridad de WEP. Se introdujo WPA2 con el criptosistema AES, pero al poseer requisitos computacionales muy exigentes surgió una implementación parcial antes de su publicación por parte de Wifi Alliance que se llamó WPA. Esto fue así por retrocompatibilidad de los dispositivos ya desplegados en el mercado, de lo contrario, habría obligado a la compra de nuevos aparatos.

2.1.3 Topologías de redes inalámbricas

Hay dos tipos de redes previstas en IEEE 802.11 según su topología: red de infraestructura y red ad-hoc.

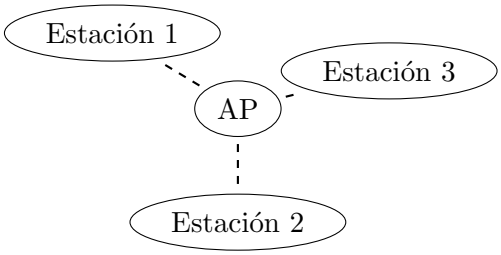
- Red ad-hoc.

  En una red de este tipo las estaciones se intercomunican directamente sin un componente central o punto de acceso. Su propósito es conectar estaciones durante un corto periodo de tiempo cuando no existe una infraestructura de red. Opcionalmente una de las estaciones puede proporcionar interconexión a otras redes.



- Red de infraestructura.

  Es la topología de red más común, donde un AP interconecta todas las estaciones de un dominio y además permite el acceso a otras redes.



2.1.4 Formato de las tramas

El estándar IEEE 802.11 define el formato de las tramas de nivel 2 (nivel de enlace) intercambiadas en una red inalámbrica.

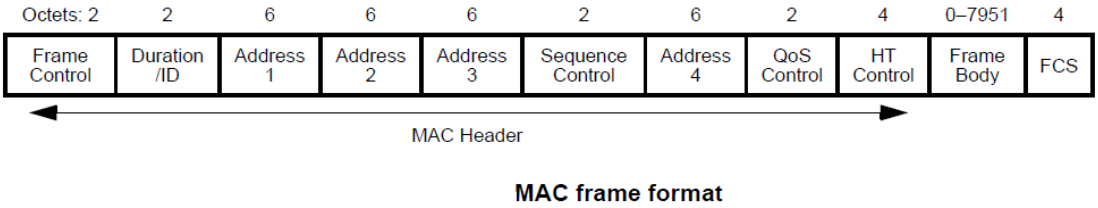


Ilustración 2.1: Formato de un paquete (frame)

La ilustración 2.2 muestra el formato de un paquete 802.11 real capturado, donde se puede observar el vector de inicialización IV, el campo *wepdata* y el código de detección de errores ICV. Mientras que el primero de ellos viaja sin cifrar, el campo que contiene la información y el ICV son cifrados.

Por otra parte, la ilustración 2.3 muestra el formato en claro de la parte correspondiente a WEP de la ilustración anterior 2.2. Cabe destacar cómo *wepdata*, además de contener la información, se ha fragmentado en otros campos más pequeños. Estos nuevos campos son cabeceras añadidas por protocolos superiores (LLC, SNAP, IP, TCP) en la arquitectura.

El ataque a texto claro conocido se aprovecha de la existencia de estas cabeceras que son invariantes y conocidas para todos los paquetes. Esto es, al tener conocimiento del valor de estas cabeceras y de su representación cifrada, se puede lograr su *keystream* simplemente combinándolos con la operación XOR.

2.2 Comprobación de redundancia cíclica

La comprobación de redundancia cíclica (*Cyclic Redundancy Check*, *CRC*) es un mecanismo de detección de errores empleado para detectar cambios accidentales en datos almacenados o transmitidos por algún medio. Adhiere información redundante a los datos para preservar su integridad.

El nombre de este mecanismo hace alusión a los códigos cíclicos, sin embargo no es uno de ellos. Se ha mantenido el nombre por tradición. Ambas herramientas comparten una definición basada en anillos de polinomios y sus cocientes.

Recibe como entrada un flujo variable de datos y retorna un resultado de longitud fija como salida. El resultado, llamado *checksum*, se añade a la cola de la información.

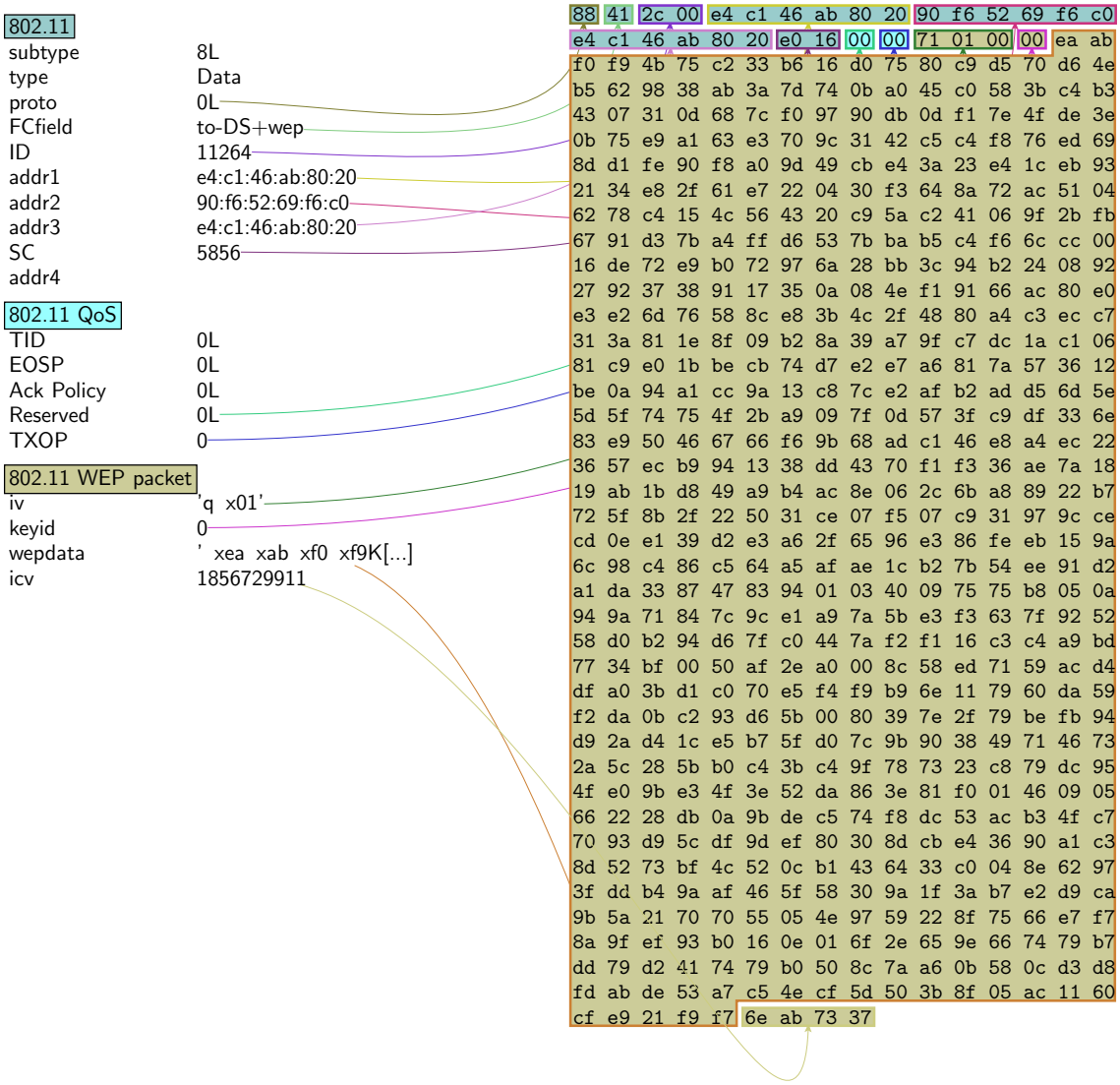


Ilustración 2.2: Paquete 802.11

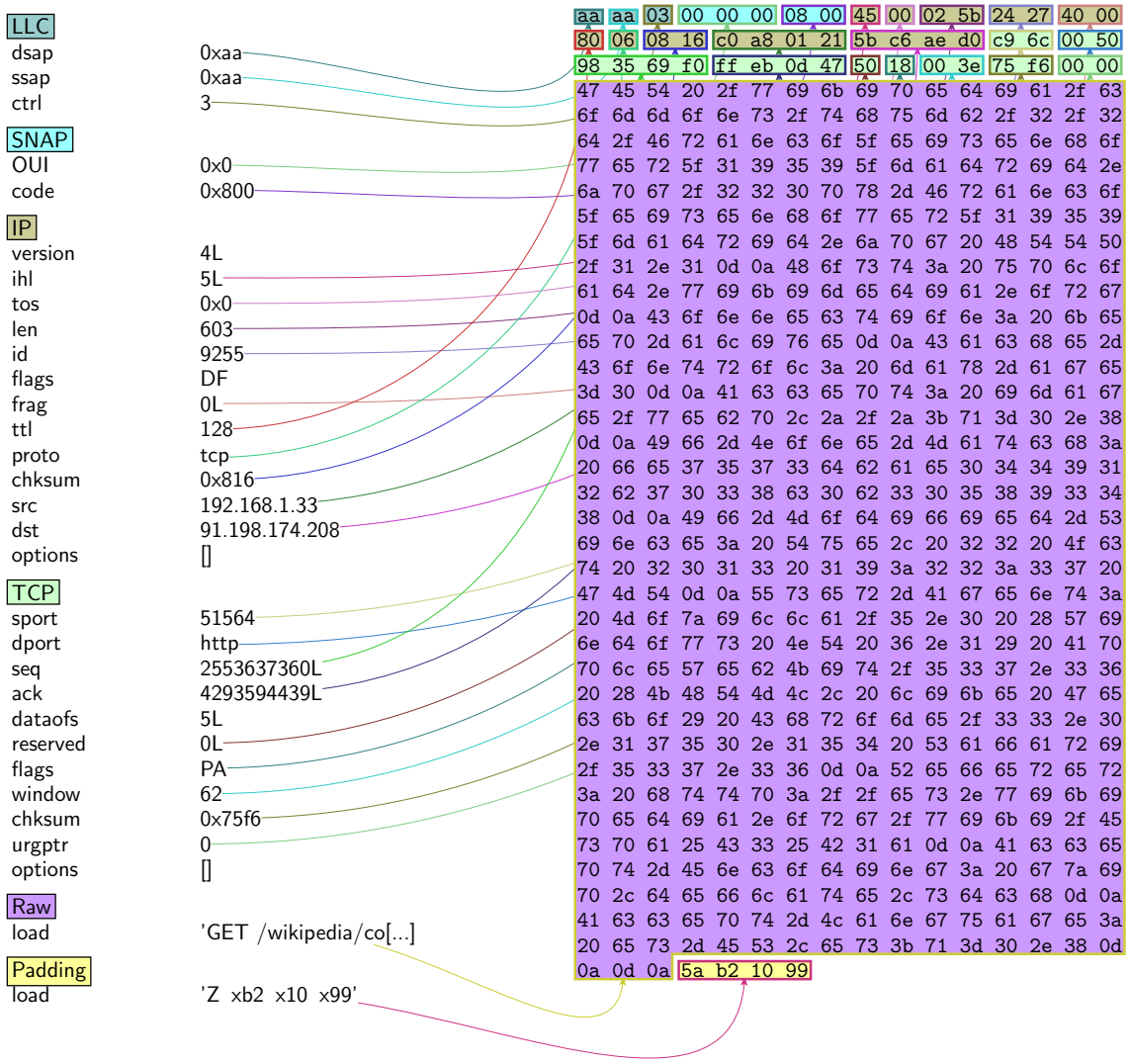


Ilustración 2.3: Payload de un paquete WEP sin cifrar

A cada paquete, antes de su transmisión, se le añade un *checksum* a la cola. Una vez recibido en su destino, se vuelve a calcular su *checksum* a partir de los datos y se compara con el *checksum* del origen. Si ambos *checksum*s no coinciden, el paquete ha sufrido alguna alteración en su viaje y se pide al emisor que lo envíe de nuevo. En caso afirmativo, se admite el paquete.

Su implementación en hardware es simple ya que básicamente sólo se requiere desplazamientos de registros y operaciones XOR. Por otra parte, este mecanismo tiene una interpretación matemática que facilita su anásilis.

A partir de aquí, se entenderá paquete como polinomio del anillo $\mathbb{F}_2[X]$, siendo cada bit un coeficiente binario. Cada byte se corresponde con un elemento de $\mathbb{F}_2[X]$ de grado menor que 8, según se muestra en los ejemplos siguientes en los que se usa la notación hexadecimal para represantar un byte:

`01`	`0000 0001`	X^7
`80`	`1000 0000`	$X^0 = 1$
`53`	`0101 0011`	$X^7 + X^6 + X^3 + X$

Si un paquete tiene varios bytes, el polinomio que se le asocia se forma multiplicando por X^{8i} el polinomio correspondiente al byte de la posición i-ésima.

$$
\begin{array}{lllllllllll}
\texttt{53:48:9D} & = & & (& X^7 & +X^6 & & & +X^3 & +X & &) \\
 & & X^8\times & (& & & & X^4 & & +X & &)+ \\
 & & X^{16}\times & (& X^7 & & +X^5 & +X^4 & +X^3 & & +1 &)+
\end{array}
$$

Con esta identificación, las sumas definidas en $\mathbb{F}_2[X]$ se corresponden con la operación XOR bit a bit.

Para definir este mecanismo de detección de errores es necesario fijar un polinomio P_{CRC}. Dependiendo de este polinomio surgen las diferentes variantes de CRC, entre ellas las más comunes CRC16, CRC32 y CRC64. WEP utiliza CRC32 con un polinomio de grado 32:

$$P_{\text{CRC}} = X^{32}+X^{26}+X^{23}+X^{22}+X^{16}+X^{12}+X^{11}+X^{10}+X^8+X^7+X^5+X^4+X^2+X+1$$

Hay otros polinomios que consiguen detectar más errores en condiciones concretas de ruido, pero este polinomio se comporta mejor en media para diferentes condiciones de ruido. En el informe de Hammond Jr *et al.* [1975] aparece la cita siguiente:

> *“ The code polynomial CRC32, is recommended as specific choice. The characteristic of this polynomial are investigated in detail and it is shown that the polynomial has probability of undetected error no larger than on the order of three times that the best polynomial tailored to each specific channel model. ”*

Un paquete ampliado con los bytes del CRC debe verificar la propiedad siguiente:

$$P \equiv P_{\text{ONE}} \bmod P_{\text{CRC}}, \tag{2.1}$$

donde P es el polinomio asociado al paquete ampliado y P_{ONE} un polinomio de grado 31 con todos los coeficientes a 1. Es decir, los paquetes válidos son los polinomios cuyo resto al dividir por P_{CRC} es P_{ONE}. En otras palabras, se considera la clase de equivalencia en el cuerpo $\mathbb{F}_2[X]/(P_{\text{CRC}})$.

Desde este punto, el símbolo $=$ indicará que ambos lados son iguales en $\mathbb{F}_2[X]$, mientras que el símbolo $\equiv$ indicará que las clases de equivalencia (elementos de $\mathbb{F}_2[X]/(P_{\text{CRC}})$) de ambos lados coinciden, es decir, que los polinomios son congruentes módulo P_{CRC}.

Debido a que el polinomio empleado P_{CRC} es de grado 32, el resto de la división (*checksum*) será de grado 31 o inferior con un tamaño de 32 bits (4 bytes).

El procedimiento para hallar el *checksum* de un mensaje consiste en:

- desplazar 4 bytes el mensaje a la izquierda, $Q \cdot X^{32}$
- rellenar esos 32 bits con unos, $Q \cdot X^{32} + P_{\text{ONE}}$
- calcular el resto de la división entre P_{CRC}. $(Q \cdot X^{32} + P_{\text{ONE}}) \bmod P_{\text{CRC}}$

En la siguiente demostración, planteamos la condición que tiene que cumplir el *checksum* y despejamos la incógnita ICV. La segunda ecuación se corresponde con los tres pasos arriba indicados.

$$Q \cdot X^{32} + \text{ICV} \equiv P_{\text{ONE}}$$
$$\text{ICV} \equiv Q \cdot X^{32} + P_{\text{ONE}}$$

2.2.1 Ejemplo

Se quiere encontrar el *checksum* del mensaje "SOS" cuya codificación ASCII es `53:4F:53`

$$Q = X^{23} + X^{22} + X^{19} + X^{17} + X^{15} + X^{14} + X^{13} + X^{12} + X^{9} + X^{7} + X^{6} + X^{3} + X$$

$$(Q \cdot X^{32} + P_{\text{ONE}}) \bmod P_{\text{CRC}} = ICV$$

$$ICV = X^{29} + X^{27} + X^{22} + X^{21} + X^{16} + X^{13} + X^{12} + X^{11} + X^{10} + X^{6} + X^{3} + X^{3} + 1$$

Este polinomio se corresponde con los bytes `92:3C:86:14`. El paquete ampliado `53:4F:53:92:3C:86:14` será aceptado por un algoritmo que compruebe el CRC al cumplir (2.1).

En este ejemplo se ha omitido un detalle que en realidad se lleva a cabo en la práctica. Dicho detalle consiste en conmutar los 32 primeros bits de un paquete, con el fin de evitar que en el caso de pérdida de una cadena de bits a 0 al comienzo del paquete el *checksum* sea el mismo como si nunca los hubiera tenido. Para el cálculo del campo ICV en el ejemplo de la ilustración 2.3 sí se ha tenido en cuenta este preproceso. Es decir, antes del cálculo del *checksum*, los cuatro primeros bytes `AA:AA:03:00` se transforman en `55:55:FC:FF`. El código del anexo B tiene en cuenta esta especificación para calcular el campo ICV.

2.3 Descripción

WEP permite el acceso a la red con hasta 4 claves secretas Rk distintas. Si una red está protegida por WEP, el punto de acceso AP se encargará de informarlo mediante la transmisión periódica de paquetes de administración llamados *beacon frames*. Sólo los paquetes que transmitan datos en una red WEP son cifrados, mientras que los paquetes

de gestión como los *beacon frames*, *acknowledgement frames* y similares se envían en claro.

La versión inicial sólo permitía claves de 40 bits (40 bits de Rk + 24 bits de vector de inicialización IV = 64 bits). En versiones posteriores se permitieron claves de 108 bits (108 bits de Rk + 24 bits de IV = 128 bits). Incluso algunos fabricantes alcanzaron claves de 232 bits.

Entre otras pequeñas modificaciones en el estándar, los fabricantes también han implementado por su cuenta el método de generar el vector de inicialización.

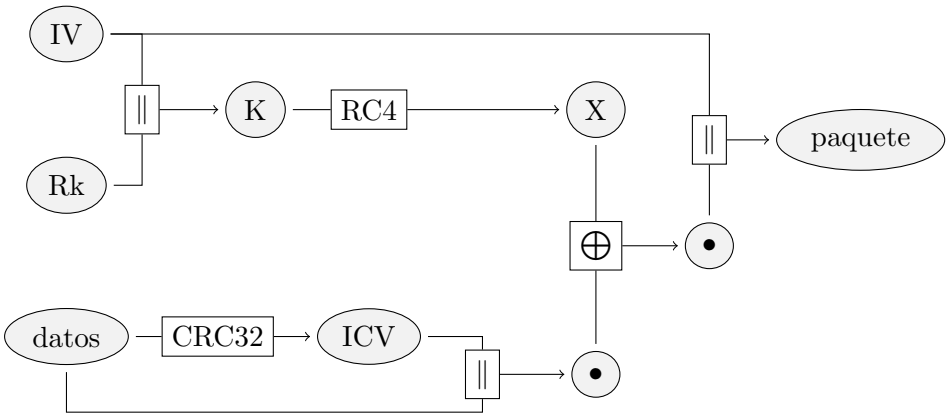
- El IV se puede obtener por un generador de números pseudoaleatorios PRNG independientemente de los pasados.
- Partiendo de un IV inicial, incrementar este en 1.

2.4 Proceso de cifrado

Pasos que un computador lleva a cabo para enviar un paquete por la red:

1. $CRC32$ calcula un *checksum* ICV a partir de los datos y es concatenado al final de los propios datos.
2. Se elige un IV siguiendo una de las opciones mencionadas anteriormente.
3. El IV y la Rk se concatenan y forman la clave $K = IV||Rk$.
4. La clave K se introduce en el algoritmo $RC4$ para generar un *keystream* X de la misma longitud que los datos y el ICV.
5. Se realiza la operación XOR entre los datos en claro y el ICV con X y se obtienen los datos cifrados.
6. Los datos cifrados, el IV en claro y las cabeceras construyen un paquete para ser enviado.

Los datos y el *checksum* son los únicos campos cifrados de todo el paquete generado.

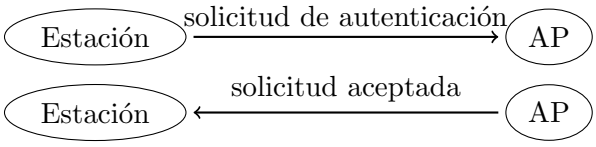


2.5 Autenticación

WEP define dos modos de cómo un usuario puede autenticarse para unirse a una red.

2.5.1 Autenticación de sistema abierto

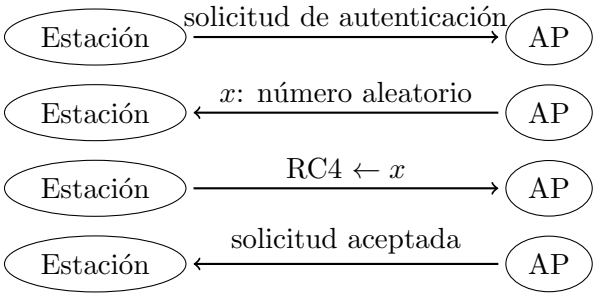
No existe autenticación propiamente dicha. El usuario envía una solicitud al punto de acceso AP para autenticarse, y este le responde afirmativamente.



2.5.2 Autenticación mediante clave compartida

Se realiza un diálogo entre el AP y el usuario, en el cual se intercambian 4 mensajes.

1. El usuario envía una solicitud al AP para unirse a la red.
2. El AP le responde con un número aleatorio sin cifrar.
3. El usuario devuelve el número cifrado con la clave compartida.
4. Si el número es correctamente descifrado por el AP, el usuario tendrá acceso a la red.



Al superar esta prueba, el usuario podrá enviar paquetes por la red.

2.6 Debilidades

Desde el principio a WEP se le ha ido descubriendo debilidades en el diseño que poco a poco han ido minando su seguridad hasta que en 2004 se abandonó definitivamente por no considerarse nada seguro. La primera de ellas incluso ya había salido antes de publicarse WEP [Wagner, 1995]. Según Bittau *et al.* [2006] existen cuatro debilidades en los que se apoyan cuatro tipos de ataques distintos: el uso de claves secretas cortas hace posible obtenerlas por fuerza bruta, la reutilización de *keystreams* permite la recuperación parcial de los paquetes, ciertas IV débiles en RC4 son utilizadas mediante ataques estadísticos para la revelación de la clave y el tratamiento del código CRC32 consigue la recuperación total de los mensajes. Al verse WEP totalmente desbordado, se procedió a la creación de su evolución WPA en el estándar IEEE 802.11i del año 2004 especialmente para subsanar todas las debilidades publicadas.

Se explican, a continuación, cada una de las debilidades expuestas anteriormente:

- Claves secretas cortas.

  Si una red tiene una clave secreta corta se puede averiguar mediante fuerza bruta, probando todas las posibilidades en un máximo de un mes con un ordenador común. En el caso de WEP, el estándar IEEE 802.11 original especificó una clave de 40 bits, la cual es bastante corta, así que algunas implementaciones establecieron la clave de WEP a 104 bits para evitar ataques de este estilo.

- Reutilización de *keystreams*.

  Si un *keystream* es descubierto, todos los paquetes cifrados con ese *keystream* podrán ser descifrados, y a la vez, será posible cifrar paquetes con ello y después transmitirlos por la red. El *keystream* se halla a partir de texto cifrado y texto claro con la operación XOR, pero como se supone que el texto claro no se conoce, esta debilidad nunca se tuvo en cuenta hasta que se encontró la forma de conocerlo.

  La técnica más práctica para hallar el texto claro se basa en el mecanismo de autenticación de un red de clave compartida (Shared Key Autentication) explicado en la sección 2.5. Consiste en que un estación para autenticarse en una red envía una solicitud al AP. Este le envía un número que la estación deberá cifrar y devolver, y si es correcto, la estación tendrá acceso a la red. Un atacante, que ha observado todo el diálogo, podrá averiguar el *keystream* con la operación XOR del mensaje claro y del cifrado. Basta interceptar otro paquete con el mismo IV que el del mecanismo para que el atacante pueda leer el trozo del mensaje correspondiente al tamaño del *keystream* conseguido.

  Aún pudiéndose conseguir *keystreams* y leer algún mensaje no es una debilidad muy preocupante porque hay 2^{24} *keystream* posibles, uno por cada IV, lo que hace al ataque demasiado complejo.

- RC4.

  La clave de una red WEP puede ser revelada usando IV débiles en el criptosistema RC4 según el informe de Fluhrer *et al.* [2001], aunque la vulnerabilidad de RC4 ya fue citada por Wagner [1995].

  En 2001, FMS publicaron el primer ataque en WEP donde se expone que la clave secreta de una red puede ser revelada usando IV débiles. El informe FMS recogía que con un paquete con una IV débil se podía hallar un byte de la clave, y por tanto, para un gran número de paquetes capturados se podía revelar la clave al completo. Para más detalle consultar sección 3.1.

La industria creó filtros para evitar usar la IV consideradas como débiles, lo que provocó el surgimiento de nuevas IV débiles, y que no se terminara la amenaza.

En la actualidad existe software al alcance de usuarios no expertos que emplea el ataque FMS para averiguar la clave de redes WEP sin necesitar conocimientos en la materia.

- *checksum* CRC32.

  Esta debilidad radica en ayudarse del código *checksum* del mecanismo de detección de errores para averiguar bytes de un paquete. Un ataque apoyado en ella es el ataque Chop-Chop que averigua byte a byte el contenido de un paquete por la cola. Otro ataque es el Ataque Inductivo de Arbaugh [2001] que consiste en lo mismo que Chop-Chop pero comenzando por el principio del paquete. Para más detalle del ataque Chop-Chop ver la sección 3.2.

Capítulo 3

Dos estrategias de criptoanálisis

WEP tiene una larga historia de ataques a sus espaldas. Al principio, los ataques no eran prácticos así que la industria no invirtió en nuevas soluciones de seguridad, sino que creó parches para mitigarlos. Con el paso del tiempo, los ataques se hicieron cada vez más fuertes utilizando técnicas más potentes, sin embargo, la industria siguió creando parches sin encontrar una solución definitiva.

Los ataques surgidos son de cuatro tipos según Bittau *et al.* [2006]. Unos emplean fuerza bruta hasta encontrar la clave correcta, otros averiguan los *keystreams* que cifran los mensajes, otros por medio de la estadística, logran la clave secreta aprovechando la debilidad de ciertos campos IV en el criptosistema RC4, y otros explotan el sistema de detección de errores del protocolo.

A continuación, se explica el ataque FMS que pertenece al tercer grupo, y el ataque Chop-Chop que pertenece al último. Estos y otros ataques están ampliamente explicados en la tesís de Tews [2007].

3.1 Ataque FMS

Fluhrer, Mantin y Shamir publicaron el primer ataque de recuperación de clave sobre WEP en 2001. En realidad, el artículo demostraba, de forma teórica, la debilidad del criptosistema RC4, pero al ser la base del protocolo de cifrado WEP, también se demostraba la de este. En 1995, la vulnerabilidad ya había sido apuntada por Wagner [1995] en un *post*, dos años antes del estándar 802.11.

El ataque se basa en la existencia de una correlación entre los primeros bytes de la clave secreta K, usada en RC4 para generar el *keystream* X, y el primer byte de X. Conociendo los primeros bytes de K y el primero de X, se puede averiguar el siguiente byte de K. La probabilidad de que ese valor sea correcto es superior a 1/256. Esto proporciona al atacante una pequeña ventaja frente la adivinación.

El esquema del ataque es el siguiente: suponiendo que un atacante conoce los l primeros bytes de K, entonces puede simular las l primeras iteraciones $[0, \ldots, l-1]$ del bucle de RC4-KSA (Algoritmo 1), obteniendo S_l y j_l[1]. La siguiente iteración l del bucle no puede simularse porque no se conoce $K[l]$. Se despeja $K[l]$:

$$j_{l+1} = j_l + K[l] + S_l[l]$$

$$K[l] = j_{l+1} - j_l - S_l[l]$$

[1]Los subíndices representan el estado del elemento. S_l representa a la permutación S en el estado l, es decir, pasadas l iteraciones.

Para obtener el valor $K[l]$ que buscamos, sólo es necesario conocer j_{l+1}. De continuar con la simulación, se sabe que el valor de la posición j_{l+1} será intercambiado con el valor de la posición l, por lo tanto, en la iteración siguiente $(l+1)$, el valor de la posición l, $S_{l+1}[l]$, estaría en la posición j_{l+1} en la iteración anterior l.

$$j_{l+1} = S_l^{-1}[S_{l+1}[l]]$$

Ahora se desconoce $S_{l+1}[l]$.

$$K[l] = S_l^{-1}[S_{l+1}[l]] - j_l - S_l[l]$$

Fluhrer, Mantin y Shamir propusieron las siguientes condiciones para averiguar esta incógnita:

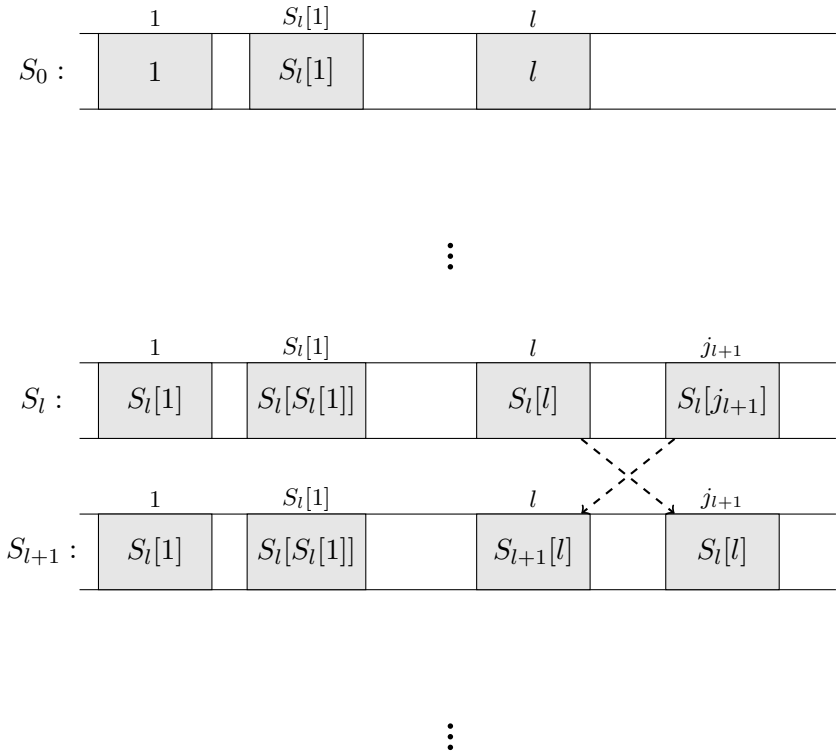
$$\begin{cases} S_l[1] < l \\ S_l[1] + S_l[S_l[1]] = l \end{cases} \tag{3.1}$$

Se supone que las condiciones se cumplen pasadas las l primeras iteraciones de RC4-KSA.

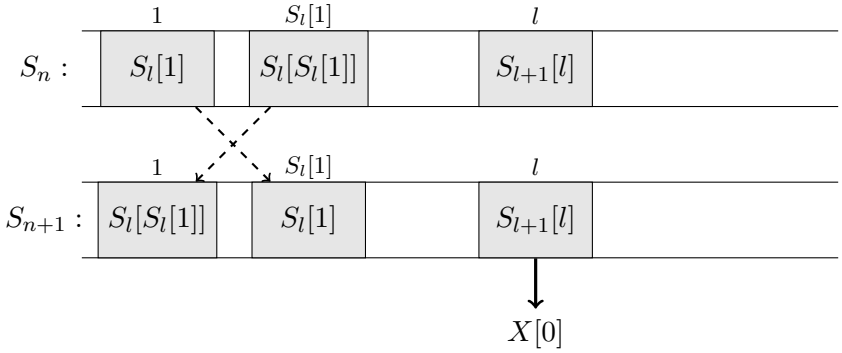
A partir de la siguiente iteración, si j cambia aleatoriamente para el resto de RC4-KSA, los valores $S[1]$, $S[S[1]]$ y $S[l]$ no se alterarán con una probabilidad de $(\frac{1}{e})^3 \approx 5\,\%$.

Cuando el primer byte de X, $X[0]$, es retornado por RC4-PRGA, j tomará el valor $S_n[1]$, y $S_n[1]$ y $S_n[j]$ serán intercambiados[2]. Después del intercambio, la condición $S[1] + S[S[1]] = l$ todavía se cumple, y $X[0]$ será $S[l]$.

Para una clave dada Rk, que el proceso anterior tenga éxito depende del vector de inicialización IV. Al capturar un paquete es inmediato comprobar si se cumplen las condiciones de (3.1) pero esto no garantiza que su IV sea débil, es decir, que las tres posiciones de la permutación de las que se ha hablado permanezcan invariantes entre los pasos $l+1$ y n.



[2]Han transcurrido n iteraciones. En RC4, $n = 256$, lo que significa que se ha completado KSA y se encuentra en la primera iteración de PRGA.



Si las condiciones de (3.1) se cumplen, la siguiente función:

$$S_l^{-1}[X[0]] - j_l - S_l[l]$$

tomará el valor de $K[l]$ con una probabilidad de $(\frac{1}{e})^3 \approx 5\,\%$.

Sólo existe un 5 % de probabilidad de que $K[l]$ sea correcto, por lo tanto, es necesario capturar muchos paquetes para aumentar la probabilidad, a lo sumo 9.000.000 de paquetes para un 50 % de eficacia, según los estudios consultados. Supiendo una red con velocidad a 100 Mbps y un tamaño MTU [3] por paquete de 7.981 bytes, se tardaría menos de dos horas en capturar 9.000.000 de paquetes. Aunque normalmente se puede conseguir suficientes paquetes en tiempo razonable, si el tráfico de la red no es elevada existen mecanismos para inducir tráfico.

Para una gran cantidad de paquetes, $K[l]$ será el valor más votado, es decir, el valor que haya salido en más ocasiones.

Para hallar $K[l+1]$ se volverá a analizar los paquetes capturados, teniendo en cuenta el valor de $K[l]$. Así hasta completar K. En caso de que K no sea la clave secreta correcta, se tomaría el siguiente valor más votado en el elemento de K donde estuviesen los votos más reñidos y se procedería a completar K de nuevo.

En las redes protegidas por WEP, los tres primeros bytes de K se pueden saber porque los vectores de inicialización IV son transmitidos sin cifrar en los paquetes. Así se dice en el estándar 802.11:

"The IV is transmitted in the clear since it does not provide an attacker with any information about the secret key."

Con la correlación, esta cita carece de razón si un atacante conoce el primer byte del mensaje claro.

El primer byte de X también es posible conocerlo. Debido a que los primeros bytes de texto claro de la mayoría de paquetes son fácilmente predecibles, el atacante es capaz de recuperar los primeros bytes de X usados para cifrarlos como se indica en la sección 2.1.4.

En resumen, para cada paquete capturado, el atacante conoce los tres primeros bytes de la clave secreta K, pudiendo ejecutar tres iteraciones de RC4, y el primer byter de X en algunos casos.

Otros ataques surgieron basándose en FMS. En 2004, KoreK añadió 16 correlaciones al ataque original logrando reducir la cantidad de paquetes capturados a 700.000 para conseguir un 50 % de probabilidad de éxito. Con PTW, publicado en 2007 por Pyshkin, Tews y Weinmann, es posible recolectar 40.000 paquetes en 60 segundos para alcanzar un 50 % de probabilidad de éxito.

[3] Maximum Transmission Unit

3.1.1 Ejemplo

Se muestra el ataque sobre un paquete cuyo IV (que sabe que se transmite en claro) es `16 : EB : FF`.

Conocidos los tres primeros bytes de la clave K, es posible ejecutar tres iteraciones de RC4. Ver ilustración 3.1.

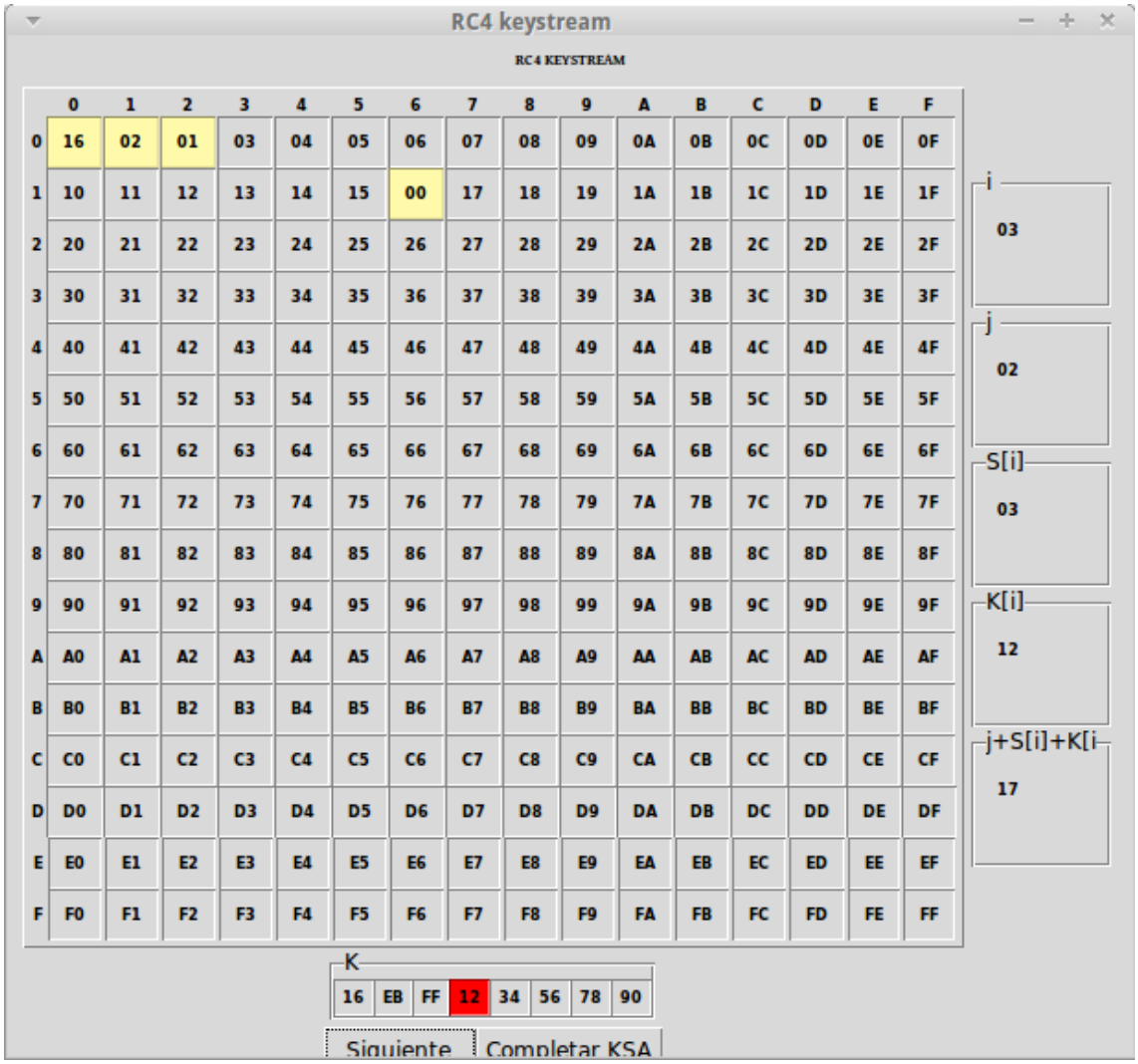


Ilustración 3.1: Tercera iteración

En la iteración $l = 3$, la posición `01` es ocupada por el valor `02` y la posición `02` por el valor `01` cumpliéndose las condiciones (3.1).

$$S_3[1] = 2 < 3$$

$$S_3[1] + S_3[S_3[1]] = 2 + 1 = 3$$

Además, como el primer byte $X[0]$ del *keystream* es conocido (en este caso `17`) por el ataque a texto claro conocido, se puede averiguar la posición j_{3+1} en la que se encontraría en la iteración $l = 3$, pudiendo calcular el valor de $K[l = 3]$ con la operación siguiente:

$$K[l] = j_{l+1} - j_l - S_l[l]$$

$$K[3] = j_4 - j_3 - S_3[3] = \texttt{17} - \texttt{02} - \texttt{03} = \texttt{12}$$

Este valor es correcto si $S[1]$, $S[S[1]]$ y $S[3]$ permanecen quietas a partir de la iteración 4 para el resto de RC4, y esto sólo ocurre con una probabilidad del 5 %, por lo que, en un ataque real, se debería hacer este procedimiento para muchos paquetes.

La siguientes ilustraciones (3.2) (3.3) no podría ejecutarlas un atacante pero son mostradas para demostrar que el IV es débil. En lo restante de RC4-KSA, $S[1]$, $S[S[1]]$ y $S[3]$ se han mantenido invariantes.

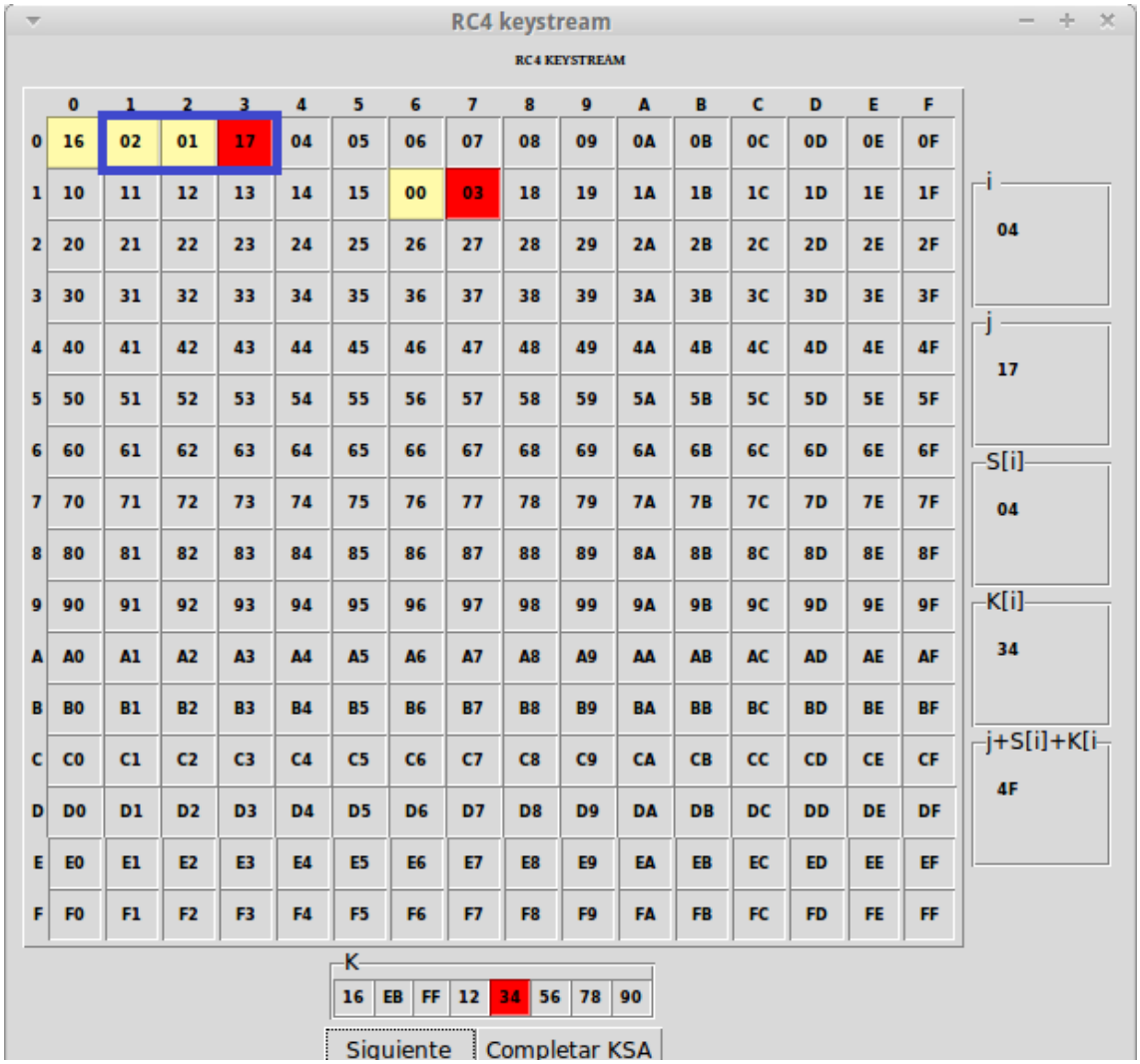


Ilustración 3.2: Cuarta iteración

Este paquete sería desechado para saber el valor de $K[4]$ porque con tal IV en la iteración $l = 4$ no se cumplirían las condiciones (3.1).

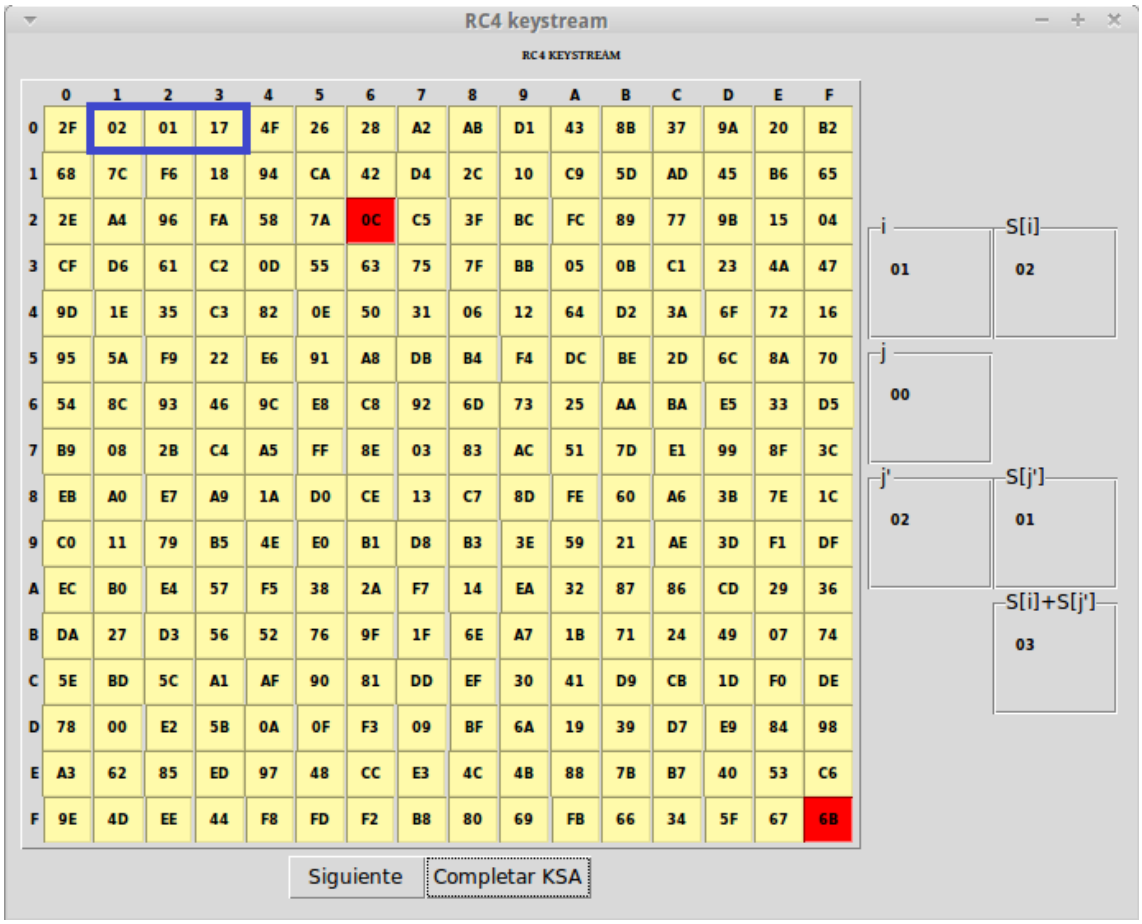


Ilustración 3.3: Iteración 255

3.2 Ataque Chop-Chop

El ataque Chop-Chop es un ataque al protocolo WEP publicado en el foro "netstumbler.org" por el usuario KoreK en el año 2004. Puede ser visto como la versión inversa del Ataque Inductivo de Arbaugh [KoreK, 2004d] [KoreK, 2004a] [KoreK, 2004c] [KoreK, 2004b].

Permite averiguar el contenido de un paquete cifrado capturado de una red WEP descifrándolo byte a byte por la cola. Cada byte es probado por fuerza bruta, generándose cada vez, tras una serie de pasos, un paquete nuevo que es enviado a un oráculo (en la práctica, el punto de acceso AP) y este determina si el paquete posee un *checksum* válido. En caso afirmativo, el byte probado corresponde al byte en claro.

Por cada byte se necesita una media de 128 llamadas al oráculo para averiguar si es correcto, ya que para 8 bits hay 256 posibles combinaciones. Por lo tanto, el promedio de llamadas al oráculo para conseguir descifrar un paquete de m bytes es de $128 \cdot m$ llamadas.

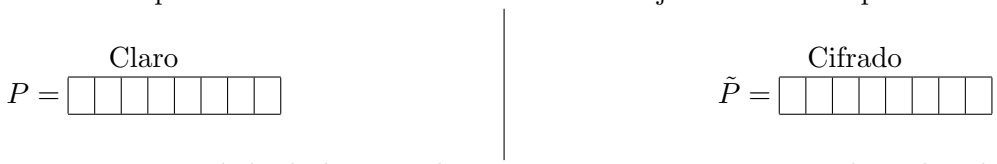
No obtiene la clave raíz *rootkey*, ni se basa en posibles debilidades del algoritmo de cifrado RC4, simplemente emplea fuerza bruta. Este procedimiento se ha de hacer para cada paquete capturado al no lograr un *keystream* maestro que pueda descifrar todos los paquetes transmitidos por la red.

3.2.1 Proceso

El proceso de descifrado comienza con la captura de un paquete. Este posee muchos campos de información, pero los únicos necesarios para el proceso de descifrado, aplicando el ataque Chop-Chop, son los siguientes:

- Initialization Vector (IV): 3 bytes, no cifrado.
- Data: n bytes, cifrado.
- Integer Check Value (ICV) CRC32: 4 bytes, cifrado.

Además, para comprender el proceso de descifrado, en esta exposición se manejarán dos caras por paquete, la cara clara y la cara cifrada. La cara clara es la información que se quiere ocultar, y la cifrada es la información ya oculta y que el atacante puede ver. Todos los cambios que se realicen en una cara se verán reflejados en la otra por linealidad.



El proceso se seguirá desde la cara clara para una mayor comprensión, saltando en algunas ocasiones a la cara cifrada.

El paquete capturado es un polinomio P compuesto por los bits de los campos *data* y *checksum*. Justo los campos que se desean descifrar.

$$P = data||checksum$$

Al ser capturado se supone que es válido, es decir, que su *checksum* es acorde al campo *data*, y que será aprobado por el oráculo. Siendo esto cierto, debe cumplirse la siguiente ecuación:

$$P \equiv P_{\text{ONE}}$$

donde P es el polinomio del paquete, P_{ONE} un polinomio de grado 31 con todos los coeficientes a 1, y P y P_{ONE} son congruentes módulo P_{CRC}.

Se trunca el último byte, obteniéndose como resultado un polinomio Q de tamaño 8 bits menor que P y el byte truncado R.

$$P = Q \cdot X^8 + R$$

Lo que se intentará hacer será averiguar el byte R por fuerza bruta. Se va a definir un polinomio cuyo *checksum* es válido si y sólo si el byte elegido R es correcto. Este polinomio se construye a partir de Q.

Combinando las dos ecuaciones anteriores de la siguiente forma se puede encontrar un equivalente al polinomio Q en $\mathbb{F}_2[X]/(P_{\text{CRC}})$:

$$P \equiv P_{\text{ONE}}$$
$$P = Q \cdot X^8 + R$$
$$Q \cdot X^8 + R \equiv P_{\text{ONE}}$$
$$Q \cdot X^8 + R + R \equiv P_{\text{ONE}} + R$$

$R + R$ se anula porque el anillo tiene característica 2.

$$Q \cdot X^8 \equiv P_{\text{ONE}} + R$$

Debido a que $\mathbb{F}_2[X]/(P_{\text{CRC}})$ es un cuerpo, X^8 es invertible.

$$X^8 \cdot P_{INV} \equiv 1$$

$$\begin{aligned} P_{INV} = \quad & X^{31} + X^{29} + X^{27} + X^{24} + X^{23} + X^{22} + X^{20} + X^{17} + X^{16} + X^{15} + \\ & X^{14} + X^{13} + X^{10} + X^9 + X^7 + X^5 + X^2 + X \end{aligned}$$

$$Q \cdot X^8 \cdot P_{INV} \equiv (P_{\text{ONE}} + R) \cdot P_{INV}$$

$X^8 \cdot P_{INV}$ se anula porque $X^8 \cdot P_{INV} \equiv 1$

$$P_Q = P_{INV} \cdot (P_{\text{ONE}} + R) \equiv Q$$

En esta ecuación se ha de sustituir R por un R de prueba. Este nuevo R será una prueba que se hará por fuerza bruta hasta hallar el R correcto, y eso ocurrirá cuando el oráculo admita el paquete resultante.

$$P_Q = P_{INV} \cdot (P_{\text{ONE}} + R_{PRUEBA}) \equiv Q$$

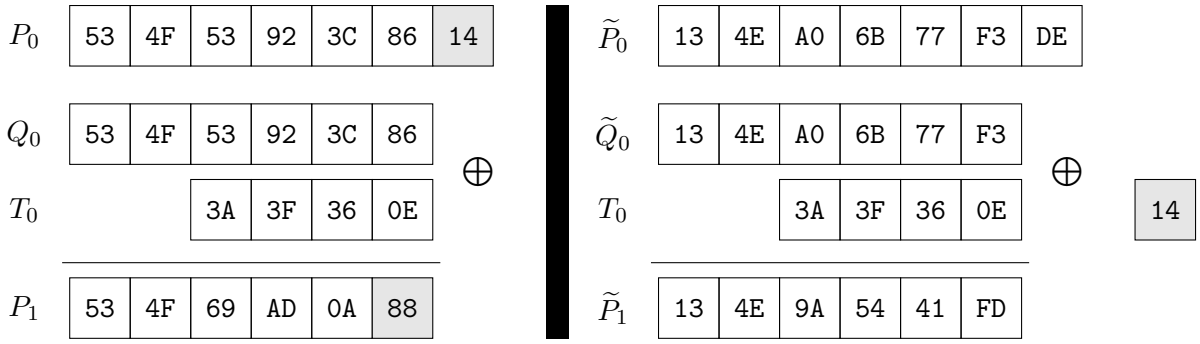
Una vez hallado el polinomio equivalente a Q en $\mathbb{F}_2[X]/(P_{\text{CRC}})$, le sumamos Q y P_{ONE}. Esta suma y el algoritmo RC4 son lineales, por lo tanto, la suma en la cara clara se hará también en la cara cifrada.

$$T = P_{INV} \cdot (P_{\text{ONE}} + R_{PRUEBA}) + P_{\text{ONE}}$$

$$P_{NUEVO} = T + Q = P_{INV} \cdot (P_{\text{ONE}} + R_{PRUEBA}) + Q + P_{\text{ONE}}$$

$$\tilde{P}_{NUEVO} = T + \tilde{Q} = P_{INV} \cdot (P_{\text{ONE}} + R_{PRUEBA}) + \tilde{Q} + P_{\text{ONE}}$$

Si R_{PRUEBA} es correcto, $P_{INV} \cdot (P_{\text{ONE}} + R_{PRUEBA}) + Q \equiv 0$, debido a que son equivalentes en el cuerpo $\mathbb{F}_2[X]/(P_{\text{CRC}})$, y con ello se consigue el polinomio P_{ONE}, que obviamente, es congruente con P_{ONE} módulo P_{CRC}. Para averiguarlo, se genera un paquete a partir de $\tilde{P}_{NUEVO}$ y se envía al oráculo. Si es correcto, el oráculo pide al atacante que se autentifique, en caso contrario, el paquete es descartado silenciosamente.



Para averiguar el siguiente byte de P, se ha de averiguar el último byte de P_{NUEVO} realizando el mismo procedimiento. Pero este byte no corresponde con el byte claro del polinomio P, así que se ha de hacer un paso más. En cada iteración, el polinomio T que modifica Q para darle validez, se acumula a un polinomio D.

$$D_i = T_{i-1} \gg X^8 + T_i$$

donde $T_{i-1} \gg X^8$ es el T de la iteración anterior desplazado 8 bits a la derecha, y T_i, el T de la iteración actual.

Teniendo el polinomio D y habiendo averiguado el byte de cualquier iteración, haciendo XOR de los últimos bytes de ambos polinomios, hallamos el byte claro del paquete original de la posición correspondiente a la iteración.

Este proceso se realiza iterativamente hasta que sólo quedan 4 bytes por resolver. Entonces, sabiendo que el polinomio P_{NUEVO} de la iteración anterior ha sido admitido por el oráculo, que P_{CRC} son 33 bits y que P_{NUEVO} son 32 bits, se puede decir que P_{NUEVO} son 32 bits a 1 porque P_{NUEVO} tiene grado menor que P_{CRC}. Finalmente, se realiza una operación XOR entre P_{NUEVO} y los 4 primeros bytes de D y ya se habrá descifrado los 4 últimos bytes del paquete P.

3.2.2 Ejemplo

Se muestra un ejemplo del ataque Chop-Chop sobre un paquete. En el lado izquierdo de la imagen se ve el ataque sobre el paquete claro y en el lado derecho se ve el ataque sobre el paquete cifrado.

Se comienza truncando el último byte de $\tilde{P}_0$ quedando un paquete $\tilde{Q}_0$ un byte menor. Este será sumado por T_0 para cada byte de prueba hasta que el oráculo lo acepte. En este caso el byte de prueba correcto es el byte 14.

P_0	53	4F	53	92	3C	86	14
Q_0	53	4F	53	92	3C	86	
T_0			3A	3F	36	0E	
P_1	53	4F	69	AD	0A	88	

$\oplus$

$\widetilde{P}_0$	13	4E	A0	6B	77	F3	DE
$\widetilde{Q}_0$	13	4E	A0	6B	77	F3	
T_0			3A	3F	36	0E	
$\widetilde{P}_1$	13	4E	9A	54	41	FD	

$\oplus$ 14

Este es el procedimiento a seguir para los nuevos paquetes admitidos por el oráculo pero no se logrará el descifrado correcto, sino el descifrado de la iteración anterior. Para conseguir el byte en claro del paquete original se debe emplear el paquete D, generado por acumulación de los T de las distintas iteraciones.

Sólo hace falta iterar el proceso hasta el momento que el paquete admitido por el oráculo sea de tamaño 4 bytes.

Q_1	53	4F	69	AD	0A
C_1		FE	96	30	07
P_2	53	B1	FF	9D	0D
Q_2	53	B1	FF	9D	
C_2	AC	4E	00	62	
P_3	FF	FF	FF	FF	

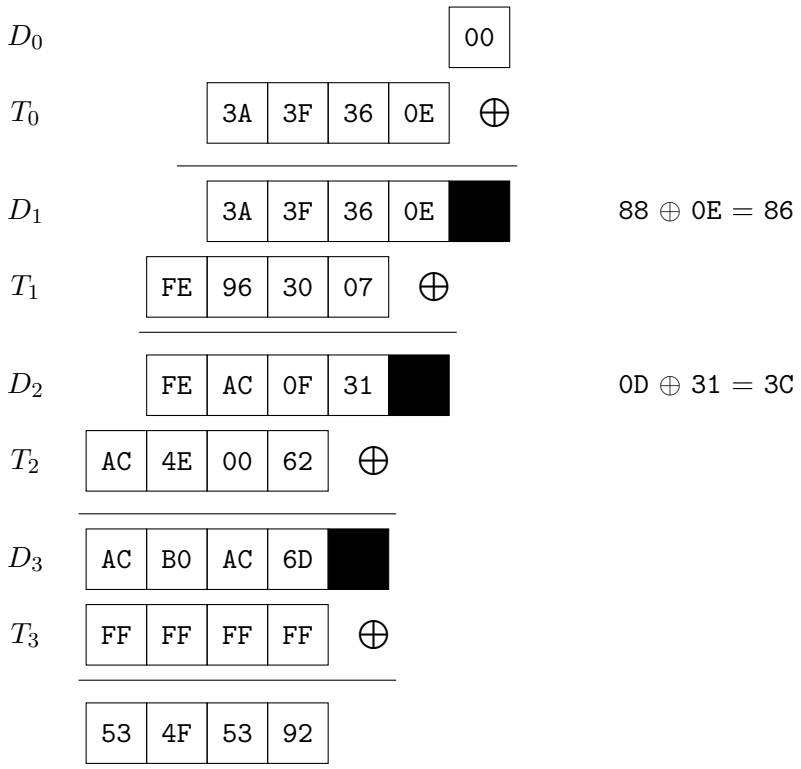
$\oplus$

$\oplus$

$\widetilde{Q}_1$	13	4E	9A	54	41
C_1		FE	96	30	07
$\widetilde{P}_2$	13	B0	0C	64	46
$\widetilde{Q}_2$	13	B0	0C	64	
C_2	AC	4E	00	62	
$\widetilde{P}_3$	BF	FE	0C	06	

$\oplus$ 88

$\oplus$ 0D



$88 \oplus 0E = 86$

$0D \oplus 31 = 3C$

Apéndice A

Generador del *keystream* para RC4

En este apéndice se encuentra el código de un simulador gráfico de RC4. Por su sencillez, en vez de un diseño UML y los casos de uso del software, se incluye una descripción informal de la estructura y el funcionamiento del software.

El programa principal se encarga de dibujar la ventana y presentar la interfaz gráfica al usuario. Esta ventana esta compuesta de varias casillas que representan cada una de las posiciones de la permutación S. La clave se puede cambiar modificando la variable K en el código.

El lenguaje de programación utilizado python en su versión 2.7 y todos los módulos utilizados se encuentran presentes en la librería estándar.

```
import Tkinter
import ttk
import tkFont
import permutacion
import casilla

N = 2 ** 8
K = [0x16,0xEB,0xFF,0x12,0x34,0x56,0x78,0x90]

root = Tkinter.Tk()
root.title('RC4 keystream')

frame = ttk.Frame(root, padding="1m 1m 1m 1m")
frame.grid()

titulo = ttk.Frame(frame, padding="1m 1m 1m 1m")
titulo.grid()

fuente1 = tkFont.Font(family='Gentium Book Basic', size=20, weight='bold')
fuente2 = tkFont.Font(family='Terminus', size=15, weight='bold')

texto1 = ttk.Label(titulo, text='RC4 KEYSTREAM', font=fuente1)
texto1.grid()

cuerpo = ttk.Frame(frame, padding="1m 1m 1m 1m")
cuerpo.grid(row=1)

lateral = ttk.Frame(cuerpo, padding="1m 1m 1m 1m")
lateral.grid(column=1)

caja = ttk.Frame(cuerpo, padding="1m 1m 1m 1m", relief=Tkinter.GROOVE)
```

```
caja.grid(column=0, row=0)

S = permutacion.Permutacion_dib(N, caja)

i = 0
cas_i = casilla.Casilla(lateral, 0, 0, 'i', 0)
j = 0
cas_j = casilla.Casilla(lateral, 1, 0, 'j', 0)
cas_Si = casilla.Casilla(lateral, 2, 0, 'S[i]', S.S[i])
cas_Ki = casilla.Casilla(lateral, 3, 0, 'K[i]', K[i % len(K)])
cas_suma = casilla.Casilla(lateral, 4, 0, 'j+S[i]+K[i]', (j + S.S[i] + K[i
    % len(K)]) % N)

clave = ttk.LabelFrame(cuerpo, text='K')
clave.grid()

bytes = []
c=0
for x in K:
    bytes.append(ttk.Label(clave, text='{0:02X}'.format(x),
                           relief=Tkinter.SUNKEN,
                           font=fuente2,
                           padding='1m 1m 1m 1m'))
    bytes[-1].grid(row=0, column=c)
    c += 1
bytes[0].config(background='#f00')


pie = ttk.Frame(cuerpo, padding="1m 1m 1m 1m")
pie.grid()

preparando = True

def dale():
    global i, j, preparando
    if preparando:
        j = (j + S.S[i] + K[i % len(K)]) % N
    else:
        j = (j + S.S[i]) % N
    S.swap(i, j)
    if not(preparando):
        print '{0:02X}'.format(S.S[(S.S[i]+S.S[j]) % N])
    i += 1
    if i == N:
        preparando = False
        i = 1
        j = 0
        cas_Si.recoloca(0, 1)
        global cas_Sj_, cas_j_, cas_Sij
        cas_j_=casilla.Casilla(lateral, 2, 0, 'j\'', (j+S.S[i]) % N)
        cas_Sj_=casilla.Casilla(lateral, 2, 1, 'S[j\']', S.S[(j+S.S[i]) % N
    ])
        cas_Sij=casilla.Casilla(lateral, 3, 1, 'S[i]+S[j\']', (S.S[i]+S.S[(
    j+S.S[i]) % N]) % N)
        cas_Ki.quita()
        cas_suma.quita()
        clave.grid_remove()
    cas_i.cambia(i)
    cas_j.cambia(j)
    cas_Si.cambia(S.S[i])
    if preparando:
        cas_Ki.cambia(K[i % len(K)])
```

```
        cas_suma.cambia((j + S.S[i] + K[i % len(K)]) % N)
        bytes[(i-1) % len(K)].config(background=root.cget('bg'))
    else:
        cas_j_.cambia((j+S.S[i]) % N)
        cas_Sj_.cambia(S.S[(j+S.S[i]) % N])
        cas_Sij.cambia((S.S[i]+S.S[(j+S.S[i]) % N]) % N)
    bytes[i % len(K)].config(background='#f00')


def completo():
    while preparando:
        dale()

b_sig = ttk.Button(pie, text='Siguiente', command=dale)
b_sig.grid()
b_KSA = ttk.Button(pie, text='Completar KSA', command=completo)
b_KSA.grid(row=0, column=1)

root.mainloop()
```

Listing A.1: Programa principal

```python
# -*- coding: utf-8 -*-

import Tkinter
import ttk
import tkFont

class Fallo(Exception):
    pass

class Permutacion():
    """ S: lista de N elementos """

    def __init__(self,N):
        self.N = N
        self.S = range(N)

    def swap(self, i, j):
        if not(0 <= i < self.N) or not(0 <= j < self.N):
            raise Fallo('Permutacion.swap(): índices fuera de rango')

        self.S[i], self.S[j] = self.S[j], self.S[i]

class Permutacion_dib(Permutacion):
    u""" Widget con una permutación de S_{256}. """

    def __init__(self, N, parent):
        Permutacion.__init__(self, N)
        if self.N != 256:
            raise Fallo('La permutación debe estar en S_{256}.')

        fuente2 = tkFont.Font(family='Terminus', size=15, weight='bold')
        self.casillas = []
        self.contenido = []
        for i in range(16):
            aux = ttk.Label(parent, text='{0:X}'.format(i), font=fuente2)
            aux.grid(row=0, column=i+1)
        for i in range(16):
            aux = ttk.Label(parent, text='{0:X}'.format(i), font=fuente2)
            aux.grid(column=0, row=i+1)
        f = 0
        c = 0
        for i in range(self.N):
            self.contenido.append(Tkinter.StringVar())
            self.contenido[-1].set('{0:02X}'.format(self.S[i]))
            self.casillas.append(ttk.Label(parent,
                                           textvariable=self.contenido[-1],
                                           font=fuente2,
                                           padding="3m 3m 3m 3m",
                                           relief=Tkinter.SUNKEN))
            self.casillas[-1].grid(row=f+1, column=c+1)
            c += 1
            if c == 16:
                f += 1
                c = 0
        self.anterior = False

    def swap(self, i, j):
        Permutacion.swap(self, i, j)
        self.contenido[i].set('{0:02X}'.format(self.S[i]))
        self.contenido[j].set('{0:02X}'.format(self.S[j]))
        self.casillas[i].config(background='#ff0000')
```

```
        self.casillas[j].config(background='#ff0000')
        if self.anterior:
            self.casillas[self.anterior[0]].config(background='#fffaaa')
            self.casillas[self.anterior[1]].config(background='#fffaaa')
        self.anterior = (i,j)
```

Listing A.2: Algorithmo RC4

```
import Tkinter
import ttk
import tkFont

class Casilla:

    def __init__(self, parent, v_pos, h_pos, nombre, valor):

        fuente2 = tkFont.Font(family='Terminus', size=15, weight='bold')
        self.frame = ttk.LabelFrame(parent,
                                    padding='5m 5m 5m 5m',
                                    text=nombre,
                                    height='2c',
                                    width='2c')

        self.frame.grid(row=v_pos, column=h_pos)
        self.frame.grid_propagate(0)

        self.variable = Tkinter.StringVar()
        self.variable.set('{0:02X}'.format(valor))
        self.labl = ttk.Label(self.frame,
                              textvariable=self.variable,
                              font=fuente2)
        self.labl.grid()

    def cambia(self, valor):
        self.variable.set('{0:02X}'.format(valor))

    def quita(self):
        self.frame.grid_remove()

    def recoloca(self, f, c):
        self.frame.grid(row=f, column=c)
```

Listing A.3: Clase auxiliar

Apéndice B

Obtención del *checksum* CRC32

En este apéndice, se encuentra el código de un módulo que implementa el código de redundancia cíclica CRC32. Este módulo puede ser importado por otros programas pero la razón de esta implementación es pedagógica, más que práctica, ya que existe un método que realiza el cálculo del *checksum* de una cadena de caracteres.

```
# -*- coding: utf-8 -*-

import binascii

P_CRC32 = 0x104C11DB7

def int2bin(num):
    return bin(num)[2:]

def ord_crc(s):
    """
    Esta función devuelve la representación de un carácter como un
    polinomio.
    """
    binary_rep = int2bin(ord(s))
    binary_rep = '0' * (8 - len(binary_rep)) + binary_rep
    #Invertimos los bits y devolvemos el correspondiente long.
    return long(binary_rep[::-1],2)

def resto_division(Q, P=P_CRC32):
    if Q.bit_length() < P.bit_length():
        return Q
    else:
        D = P << (Q.bit_length() - P.bit_length())
        return resto_division(Q ^ D, P)


def clave(numero):
    """
    Transforma un entero en el carácter correspondiente pero
    TENIENDO EN CUENTA la especificación de crc32.
    """
    resp = ''
    while numero != 0:
        num = 0
        pos = 7
        while pos >= 0:
            if numero % 2:
                num += 2**pos
            pos -= 1
```

```
            numero = numero >> 1
        resp = chr(num) + resp
    return resp

def crc32(texto):
    """
    Función que devuelve el checksum de un texto utilizando crc32.
    """

    # Se adjunta al texto 4 hexadecimales \xFF
    texto = texto + "\xff\xff\xff\xff"

    # Transformamos el texto en una lista de long
    lista = [ord_crc(s) for s in texto]
    # Ahora invertimos los cuatro primeros bytes
    for i in xrange(4):
        lista[i] ^= 255
    #iniciamos el resto a cero
    resto = 0
    # recorremos la lista
    for b in lista:
        #primero hay que desplazar el resto 8 posiciones y sumarle b
        resto = resto << 8
        resto += b
        #se halla el resto de dividir el nuevo resto entre P_CRC32
        resto = resto_division(resto, P_CRC32)
    for u in clave(resto):
        print '{0:02X}'.format(ord(u)),
    return resto
```

Listing B.1: Módulo con la función crc32

Referencias

802.11. IEEE Std. 802.11-1997. part 11. Technical report, IEEE, 1997.

Aircrack-ng. WEP and WPA-PSK keys cracking program. http://www.aircrack-ng.org/.

W. A. Arbaugh. An inductive chosen plaintext attack against WEP/WEP2. *IEEE document*, 802(01):230, 2001.

A. Bittau, M. Handley y J. Lackey. The final nail in WEP's coffin. In *Security and Privacy, 2006 IEEE Symposium on*, pages 15–pp. IEEE, 2006.

Cypherpunks. Thank you Bob Anderson, 1994. URL http://www.cypherpunks.venona.com/archive/1994/09/msg00304.html.

W. Diffie y M. E. Hellman. New directions in cryptography. *IEEE Trans. Information Theory*, IT-22(6):644–654, 1976.

S. Fluhrer, I. Mantin y A. Shamir. Weaknesses in the key scheduling algorithm of RC4. In *Selected areas in cryptography*, pages 1–24. Springer, 2001.

J. L. Hammond Jr, J. E. Brown y S.-S. S. Liu. Development of a transmission error model and an error control model. Technical report, DTIC Document, 1975.

IEEE. Institute of Electrical and Electronics Engineers. http://www.ieee.org/about/index.html.

KoreK. Aircrack, yet another WEP cracking tool for linux. http://www.netstumbler.org/unix-linux/aircrack-yet-another-wep-cracking-tool-for-linux-t11878-15.html, 2004a.

KoreK. Chopchop: experimental WEP attacks. http://www.netstumbler.org/unix-linux/chopchop-experimental-wep-attacks-t12489.html, 2004b.

KoreK. Next generation of WEP attacks. http://www.netstumbler.org/news/next-generation-of-wep-attacks-t12277.html, 2004c.

KoreK. WEP is broken. http://www.netstumbler.org/mac-os/need-security-pointers-t11869.html, 2004d.

C. E. Shannon. Communication theory of secrecy systems. *Bell system technical journal*, 28(4):656–715, 1949.

E. Tews. Attacks on the WEP protocol. *IACR Cryptology ePrint Archive*, 2007:471, 2007.

G. Vernam. Secret signaling system, July 22 1919. US Patent 1,310,719.

D. Wagner. Re: Weak keys in RC4, 1995. URL http://www.cs.berkeley.edu/~daw/my-posts/my-rc4-weak-keys. *post* en sci.crypt.