Seguridad en Sistemas y Redes de Telecomunicación

Solución (no única) del examen – 11.1.2019

1. Ejercicio 1 (valor: 1/3)

Modo ECB

- no es adecuado porque no oculta la estructura repetitiva de los datos a cifrar
- es criptográficamente imposible modificar un bloque de información, en particular es criptográficamente imposible insertar M0'

Modo CBC

- encadenando bloques se oculta la repetición de tramas
- es fácil modificar el primer bloque manipulando el vector inicial

Modo CTR

- se oculta la repetición al usar una clave derivada diferente en cada bloque
- es fácil modificar el primer bloque

En el caso de CTR, la modificación es trivial.

En bloque correcto resulta de hacer esta operación de descifrado

$$M0 = C0 \oplus Ek(IV \,||\, 0)$$

si queremos M0', necesitamos modificar C0 a C0', tal que

$$M0' = C0' \oplus Ek(IV \,||\, 0) \rightarrow C0' = M0' \oplus Ek(IV \,||\, 0)$$

En el caso de CBC, no podemos modificar C0 porque alteraríamos los demás bloques de datos. Pero sí podemos manipular IV.

El bloque correcto resulta de hacer esta operación de descifrado

$$M0 = C0 \oplus IV$$

si queremos M0', necesitamos modificar IV a IV', tal que

$$M0' = C0 \oplus IV' \rightarrow IV' = M0' \oplus C0$$

Resumen

	ECB	CBC	CTR
datos repetitivos	se nota	se ocultan	se ocultan
forzar M0'	imposible	fácil	fácil

2. Ejercicio 2 (valor: 1/3)

R debe tener un número de bits suficientes para evitar ataques de fuerza bruta. Es decir, 128 bits o más.

K se usa como secreto compartido y, por el mismo razonamiento, debe ser de 128 bits o mayor.

Si H() da más bits de los N necesarios, se escogen N cualesquiera, por ejemplo, los N primeros.

Si H() da menos bits de los N necesarios, se puede usar H() dos o más veces, con alguna sal para provocar dispersión

$$H^n(R) \,||\, H^n(R \,||\, sal) \,||\, \ldots$$

la sal debe ser sistemática y conocida por todos los dispositivos del grupo.

Pondremos a trabajar AES en algún modo que no sea ECB para ocultar las repeticiones. Por eficacia de electrónica, es conveniente usar un modo que no requiera tener el algoritmo de descifrado; es decir, que se pueden usar CFB, OFB o CTR, de los más habituales. Hay que evitar CBC, que requiere que se implemente al algoritmo de descifrar.

Un protocolo no es PFS si cuando un atacante consigue conocer una clave K(i), ese conocimiento le lleva a poder calcular todas las claves futuras.

En nuestro caso, si el atacante conociera K(n) para un valor n, puede calcular las claves K(m) para m > n. Pero no puede calcularla para valores m < n. Basta que el master siempre vaya usando valores n decrecientes y que los dispositivos recuerden el último valor usado, negándose a volver a usar un valor superior.

Con el esquema propuesto es imposible evitar que, si el atacante descubre una K, averigüe secretos anteriores. Para ello haría que usar otro modo de calcular K. Por ejemplo, podríamos usar

$$K = HMAC(R, n)$$

que hace todas las claves incalculables unas de otras. En este caso conviene evitar que n se repita, bien usando valores siempre crecientes, o usando un número aleatorio impredecible con suficientes bits para evitar repeticiones.

Si M se ve suplantado por M', podría ordenar valores de n crecientes, lo que destruiría la PFS. En el párrafo anterior se indica cómo protegerse de ello. También podría publicitar un n=0, bloqueando el sistema. Es muy difícil protegerse de ataques de denegación de servicio como el expuesto; pero el secreto no peligra.

Para autenticar M, lo más fácil es usar los mismos algoritmos de hash que ya tenemos implementados y emitir HMAC(R, n).

Aunque costoso en electrónica y complicado de hacer, también podríamos introducir un mecanismo de firma para que M radiara el valor n firmado digitalmente. Funcionar, funciona; pero causaría un incremento en las necesidades de los dispositivos IoT que tendrían que incluir un algoritmo de verificación de firma. Como algoritmos posibles de firma se podría usar un ECDSA con 256 bits o más.

3. Ejercicio 3 (valor: 1/3)

Nuestro dispositivo probablemente almacene en algún sitio el dato R para poder detener y arrancar el dispositivo sin tener que volver a cargarlo.

Si R están en memoria permanente (tipo flash o EEPROM, o similar) basta leerla.

Un diseño cuidadoso del dispositivo habrá previsto estos robos o pérdidas, y el dato estará cifrado. No se necesita un cifrado muy sofisticado; pero hay unos mínimos.

La clave para descifrar debe estar cableada en el programa. Basta desensamblarlo o acceder al código fuente para descubrir la clave. Alternativamente, podemos provocar una caída del ejecutable (por ejemplo, provocando un *buffer overflow*) para buscar entre la RAM del vaciado de memoria. Podemos buscar el valor R directamente, o la clave para recuperar R del almacenamiento externo.

En general, se puede intentar reemplazar el programa original por una versión nuestra que revele R en cuanto lo descifre.

Todos los problemas de manipulación del programa se pueden evitar si el dispositivo dispone de un chip criptográfico (tipo SIM, tarjeta EMV, DNI electrónico, etc.) que guarde el secreto en su interior y que realice las operaciones criptográficas internamente.

Probablemente sea un esfuerzo desmesurado; pero cabe la opción de encontrar un ataque por *buffer overflow* del tipo *stack smash* para hacer que el programa original funcione incorrectamente y pueda saltar a alguna rutina de *debugging* que revele R.