

Universidad Nacional Autónoma de México
Facultad de Ciencias
Criptografía y Seguridad
Tarea 4

Luna Vázquez Felipe Alberto
Hurtado Gutiérrez Marco Antonio

31 de abril de 2020



Ejercicios

- Sea $\mathcal{F} : \mathcal{K} \times \{0,1\}^n \rightarrow \{0,1\}^n$ una función de cifrado por bloques (una permutación pseudoaleatoria). Considera las siguientes formas de usar \mathcal{F} para cifrar un bloque para $m \in \{0,1\}^n$ con la clave k .

(a) $c = \mathcal{F}_k(m)$.

(b) Se elige $r \leftarrow \{0,1\}^n$ y el cifrado es $c = (r, \mathcal{F}_k(r) \oplus m)$.

¿Cuál de las dos formas consideras que es más segura? ¿Por qué? ¿Preferirías una de estas formas o algún modo de operación de los vistos en clase?

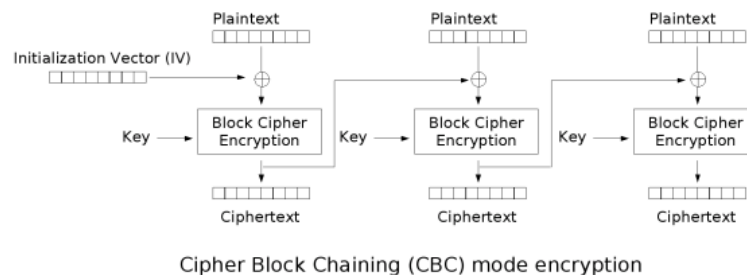
Respuesta:

Considero más segura a (b) que (a) en virtud que el cifrado de a solo utiliza la función de cifrado por bloques con el mensaje, y (b) elige a r con un rango y su cifrado es una tupla con esa r pseudoaleatoria y con la función de cifrado por bloques con r y además hace un XOR con m que genera flujo pseudoaleatorio. Por lo que b es más seguro.

Yo utilizaría un cifrado *CTR* ya que simula un cifrado de flujo. Es decir, se usa un cifrado de bloque para producir un flujo pseudo aleatorio conocido como *keystream*. Este flujo se combina con el texto plano mediante XOR dando lugar al cifrado que se parece a b pero prefiero utilizar un cifrado conocido.

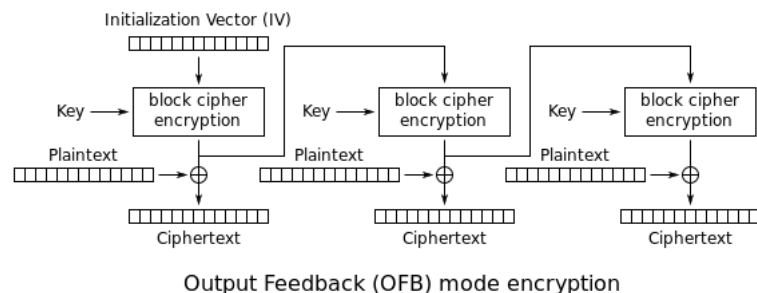
- Es posible que cuando se manda un criptotexto $c = c_1, c_2, c_3 \dots$ en la transmisión se pierda el bloque c_2 , así que se recibe el mensaje $c' = c_1, c_3, c_4, \dots$. Describe el efecto que causa un bloque perdido en el criptotexto, cuando se descifra usando los modos de operación *CBC*, *OFB* y *CTR*.

(a) CBC



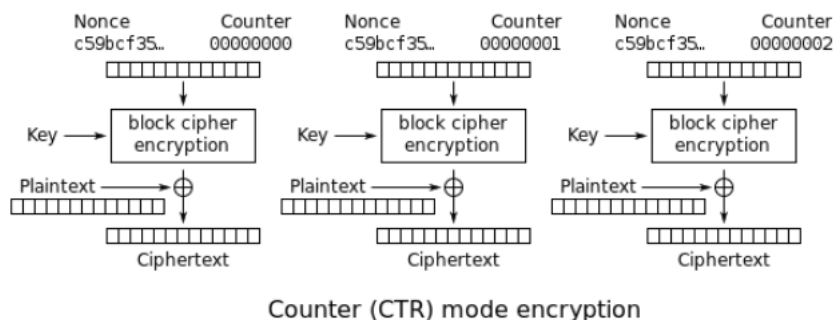
Como podemos ver en el esquema si se pierde c_2 del mensaje $c = c_1, c_2, c_3$ como para hacer el cifrado de cada bloque se depende del bloque cifrado anterior (o del vector de inicialización si se trata del primer bloque) y el texto claro del bloque actual a cifrar, lo que va a suceder al momento de descifrar los bloques es que sólo c_1 se descifra correctamente pero a partir de ahí el descifrado será incorrecto.

(b) **OFB**



Como se ilustra en el esquema en el caso del modo OFB cada bloque depende del *block cipher encryption* anterior por lo que al perderse c_2 del mensaje se tiene una reacción en cadena hacia los bloques siguientes y se afecta el descifrado.

(c) **CTR**



Dado que en este modo se depende no de algún resultado anterior si no de un contador el cual es muy difícil que sea alterado, es decir, cada bloque es independiente por lo que sólo se afectaría a c_2 en particular al momento de hacer el descifrado.

3. (2.5 pts.) Existen varias formas de usar un cifrador de bloques para construir funciones hash.

(a) ¹ Describe como hacer una función hash usando el método de Davies-Meyer.

Respuesta:

La función de compresión-bloque de longitud única Davies-Meyer alimenta cada bloque de mensaje (m_i) como la clave para un cifrado de bloque. Se alimenta el valor hash anterior (H_{i-1}) como el texto en claro a cifrar. El texto cifrado salida es entonces también XOR (\oplus) con el anterior valor hash (H_{i-1}) para producir el siguiente valor hash (H_i). En la primera ronda cuando no hay valor de hash anterior que utiliza un valor inicial pre-especificado constante (H_0).

Formalmente Davies-Meyer se puede describir como:

$$H_i = E_{m_i}(H_{i-1} \oplus H_{i-1}). \quad (1)$$

El esquema tiene la tasa (k es el tamaño de clave):

$$R_{DM} = \frac{k}{(1)(n)} = \frac{k}{n}. \quad (2)$$

Si el cifrado de bloque utiliza por ejemplo claves de 256 bits a continuación, cada bloque de mensaje (m_i) es una de 256 bits trozo del mensaje. Si el mismo cifrado de bloques

¹https://es.qwe.wiki/wiki/One-way_compression_function

utiliza un tamaño de bloque de 128 bits a continuación, los valores de entrada y de hash de salida en cada ronda es de 128 bits.

Las variaciones de este método reemplazan *XOR* con cualquier otra operación de grupo, tales como la adición de enteros sin signo de 32 bits.

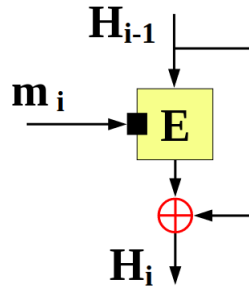


Figure 1: Davis-Meyer

- (b) Recuerda que en DES existen llaves k para las que es fácil encontrar m tal que $DES_k(m) = m$. Muestra cómo usar esta propiedad para encontrar una colisión en la construcción de Davies-Meyer aplicada a DES_k .

Respuesta:

En DES a la salida de la permutación expansiva se le hace un XOR con los bits de la clave de ronda y el resultado de eso se pasa a la etapa de substitución, aplicando la propiedad anterior y al hacer Davies-Meyer con la misma k encontraríamos un flujo pseudoaleatorio idéntico por lo que harían una colisión.

4. El ejercicio cuatro no se realizará
5. (2.5 pts) Extrae el archivo *mis archivos.zip*, que contiene el directorio *Mis archivos*. Desde este directorio ejecuta *juego.py* con Python 3.

Este programa es un ransomware de juguete. Haz un programa que funcione como "vacuna" para el ransomware, es decir, que revierta los cambios hechos por *juego.py*. Cuando se ejecute tu programa en un directorio donde anteriormente se ejecutó el ransomware, se recuperarán (exactamente) los archivos originales.

Estudiando a fondo el programa *juego.py* se descubrió que esencialmente el cifrado que hace el programa consiste en hacer un XOR entre cada uno de los bytes de el archivo original y un arreglo aleatorio de bytes de tamaño 16. En pseudocódigo digamos que hace lo siguiente:

```
k := random_bytes(16)
bytes_enc := []
for i in bytes_original
    bytes_enc.add(bytes_original[i] xor k[i mod 16])
//end for
```

Posterior a eso k se convierte a un entero y se le aplica la siguiente operación:

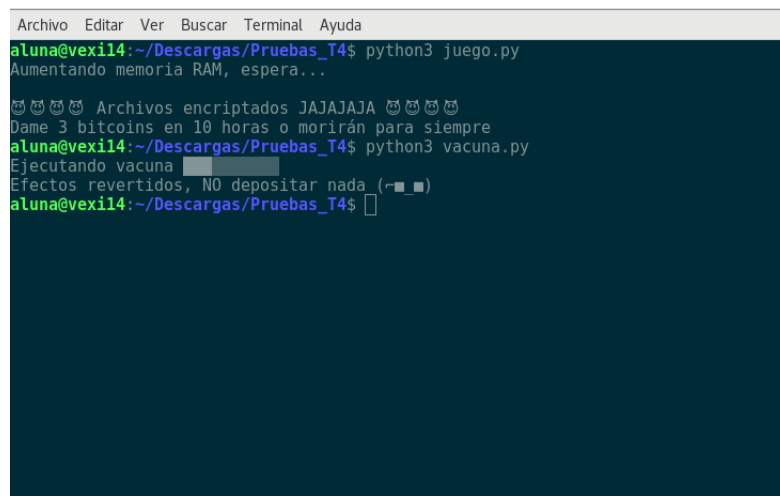
```
k = d * k % (1 << c)
```

Donde c y d son enteros generados aleatoriamente y " $<<$ " es la operación de desplazamiento de bits. Finalmente se escribe en el archivo llamado *.xyz* el resultado de concatenar c , d y k convertidos a bytes.

Para revertir los cambios primero se logró obtener k , d y c del archivo .xyz, para esto se notó que sus longitudes son constantes, por lo que se recupero fácilmente a partir de allí. Después para revertir la operación se hizo lo siguiente:

$$k = (\text{inverso} * k) \% (1 \ll c)$$

Donde inverso es el inverso modular de d con $1 \ll c$, esta idea se saco del cifrado afín visto anteriormente, y por último se volvió a aplicar el código mostrado al inicio, ya que es un *XOR* la manera de revertirlo es volviendo a aplicar esa misma operación.



```
Archivo Editar Ver Buscar Terminal Ayuda
aluna@vexi14:~/Descargas/Pruebas_T4$ python3 juego.py
Aumentando memoria RAM, espera...

🐛🐛🐛 Archivos encriptados JAJAJAJA 🐛🐛🐛
Dame 3 bitcoins en 10 horas o morirán para siempre
aluna@vexi14:~/Descargas/Pruebas_T4$ python3 vacuna.py
Ejecutando vacuna ██████████
Efectos revertidos, NO depositar nada (-█-█)
aluna@vexi14:~/Descargas/Pruebas_T4$ █
```

Figure 2: Ejecución de vacuna.