Criptografía y Seguridad
Tarea 3

Ailyn Rebollar Pérez Ángela Janín Ángeles Martínez

Ejercicio 1

Se puede aplicar un test para probar un generador de números aleatorios siguiendo el siguiente teorema:

La probabilidad de que $mcd(x, y) = 1$ para dos enteros x,y escogidos al azar es de $\frac{6}{\pi^2}$

Usa este resultado para probar tres algoritmos generadores de números aleatorios:

- La función randint.
- El algoritmo RC4 usando semillas de 4 bytes (32 bits).
- El archivo /dev/urandom de tu computadora.

Para los tres casos, usa enteros en un rango de [0,n], escogiendo n con un valor de al menos $2^{8*7} - 1$ (¿cuántos bytes se necesitan para este número?). Para cada algoritmo calcula el valor de π usando 100, 1000 y 10000 parejas (x, y), de forma que puedas llenar la siguiente tabla con valores obtenidos:

Solución: Los bytes que se requieren para almacenar ese número son 7. Y de acuerdo a nuestro programa en python3 en el archivo *estimando_pi.py* que calcula el número de parejas que son primos relativos y además imprime el valor estimado de π de acuerdo al promedio calculado para despejar π:

$$\text{promedio} = \frac{6}{\pi^2}$$
$$\pi^2 = \frac{6}{promedio}$$
$$\sqrt{\pi^2} = \sqrt{\frac{6}{promedio}}$$
$$\pi = \sqrt{\frac{6}{promedio}}$$
$$\text{pero recordemos que promedio} = \frac{Parejas_{coprimos}}{Parejas_{totales}}$$

- Al ejecutar el método *obtenParejasR* (es decir usando la función randint) para 100 parejas, obtuvimos un total de 64 parejas que fueron primos relativos. Entonces el promedio es de 0.64 pues $\frac{64}{100} = 0.64$. Y el valor estimado de π es 3.0618621784789726

- Al ejecutar el método *obtenParejasR* (es decir usando la función randint) para 1 000 parejas, obtuvimos un total de 616 parejas que fueron primos relativos donde el promedio es de 0.616 y el valor estimado de π es 3.1209389196617963.

- Al ejecutar el método *obtenParejasR* (es decir usando la función randint) para 10 000 parejas, obtuvimos un total de 6063 parejas que fueron primos relativos donde el promedio es de 0.6063 donde el valor estimado de π es 3.1458053093027876.

- Al ejecutar el método $obtenParejasRC4$ (es decir usando el algoritmo RC4) para 100 parejas, obtuvimos un total de 63 parejas que fueron primos relativos donde el promedio es de 0.63 y el valor estimado de π es 3.0860669992418384.

- Al ejecutar el método $obtenParejasRC4$ (es decir usando el algoritmo RC4) para 1 000 parejas, obtuvimos un total de 614, donde el promedio es 0.614 y el valor estimado de π es 3.1260177495791734.

- Al ejecutar el método $obtenParejasRC4$ (es decir usando el algoritmo RC4) 10 000 parejas obtuvimos un total de 6072 donde el promedio es de 0.6072 y el valor estimado de π es 3.143473067309657.

- Al ejecutar el método $obtenParejasUR$ (es decir usando la función urandom) para 100 parejas, obtuvimos un total de 62 parejas donde el promedio es de 0.62 y el valor estimado de π es 3.1108550841912757.

- Al ejecutar el método $obtenParejasUR$ (es decir usando la función urandom) para 1 000 parejas, obtuvimos un total de 604 parejas que fueron primos relativos, entonces el promedio es de 0.604 y el valor estimado de π es de 3.1517891481565017.

- Al ejecutar el método $obtenParejasUR$ (es decir usando la función urandom) para 10 00 parejas, obtuvimos un total de 6084 donde el promedio es de 0.6084 y el valor estimado de π es de 3.1403714651066386.

Entonces llenando la tabla tenemos:

	randint	RC4	urandom
100	3.0618621784789726	3.0860669992418384	3.1108550841912757
1 000	3.1209389196617963	3.1260177495791734	3.1517891481565017
10 000	3.1458053093027876	3.143473067309657	3.1403714651066386.

Ejercicio 2

Supongamos que Alicia y Bartolo se quieren mandar mensajes cifrados pero solo tienen una llave secreta compartida k de 128 bits. Para enviar un mensaje m hacen lo siguiente:

- Se escoge una cadena aleatoria s de 48 bits.
- Se obtiene el mensje cifrado $c = RC4(s||k) \oplus m$.
- Se manda la pareja (s, c).

Responde lo siguiente:

1. ¿Qué tiene que hacer Alicia para recuperar el mensaje claro cuando recibe (s, c)?

   Se manda la pareja (s, c), nos mandan la cadena aleatoria s, la cual fue concatenada con k y posteriormente cifrada mediante XOR. Dado que el algoritmo bajo las mismas entradas, devuelve la misma respuesta (esto nos asegura que podemos obtener el mensaje enviado por Bartolo bajo las mismas entradas), entonces basta con aplicar XOR a c para poder obtener m.

2. Si un adversario puede ver una lista de mensajes $(s_1, c_1), (s_2, c_2), ...$ que fueron enviados, ¿cómo puede comprobar que dos mensajes ci, cj fueron cifrados con el mismo flujo generado por RC4?

   Bajo el determinismo del algoritmo, si un adversario posee una tupla dada, (en este caso $c_i y c_j$) entonces sólo hace falta que posea un el texto claro de cada tupla (puesto que si tiene la tupla, tiene la semilla utlizada en el mensaje), aplicar XOR, y verificar que coincidan.

3. Usando la paradoja del cumpleaños, calcula aproximadamente cuántos mensajes tendría que enviar Alicia para que se repita el flujo generado por RC4.

   La paradoja del cumpleaños, modo grosso, nos indica cuántos alumnos pueden ser escogidos antes de que comiencen a repetirse sus fechas de cumpleaños. Llevando esto a nuestro problema, queremos saber entonces cuántos mensajes pueden ser enviados antes de que comiencen a repetirse nuestras cadenas s.

   Cada vez que Alicia y Bartolo se quieren enviar mensajes, toman una cadena aleatoria del conjunto s de cadenas aleatorias de 48 bits, entonces, para calcular, entonces utilizamos la operación

   $$\sqrt{N}$$

   donde N es la cardinalidad del conjunto de cadenas s. Por lo tanto, es posible enviar

   $$\sqrt{2^{48}} = 16777216$$

   mensajes, antes de que haya repeticiones.

4. Con el método de Alicia y Bartolo y tomando en cuenta lo anterior, ¿cuántos mensajes pueden cifrarse de forma segura? ¿Cuántos serían si omiten la cadena s en todo el proceso?

   De forma segura, sólo puede enviarse un sólo mensaje, donde se usa únicamente k. Debe usarse una sola vez, antes de que ésta proporcione información extra al adversario conforme realice criptoanálisis con varios textos cifrados. Esto de debe a la sencillez de XOR le proporciona, el flujo se repite.

Ejercicio 3

El profesor Bartolo guarda la tareas de sus alumnos cifradas con un cifrador de flujo. Cada tarea es un archivo de texto que comienza con lo siguiente:

No. de cuenta: XXXXXXXXXX
Tarea N
Respuestas...

con los 10 dígitos del número de cuenta del alumno, y no se incluyen otros datos personales del alumno.

Como Carlos no entregó la última tarea, maliciosamente quiere entrar a la computadora del profesor e intercambiar el número de cuenta en la tarea de Alicia por el suyo, aunque no tiene forma de conseguir la llave que usa el profesor para cifrar los archivos.

a) Suponiendo que Carlos conoce el número de cuenta de Alicia y además tiene acceso al archivo cifrado, ¿qué debe hacerle a este archivo para intercambiar su número de cuenta por el de Alicia? Así, cuando el profesor decifre el archivo que originalmente era de Alicia, ahora tendrá el número de cuenta de Carlos.
Solución:
Lo que Carlos debe hacer para poner su número de cuenta en lugar del de Alicia es primero identificar los bytes que corresponden al número de cuenta de Alicia, es decir, tomar a partir del byte 16 al 26. Después, tiene que obtener el flujo aleatorio con el que cifró la tarea el profesor y eso puede hacerlo de la siguiente manera ya que conoce parte del texto claro que es el encabezado junto con el cifrado del mismo:

$$c_1c_2...c_{10} = m_1m_2...m_{10} \oplus a_1a_2...a_{10}$$
$$a_1a_2...a_{10} = c_1c_2...c_{10} \oplus m_1m_2...m_{10}$$

Donde $a_1a_2...a_{10}$ es el flujo aleatorio, $c_1c_2...c_{10}$ es el cifrado y $m_1m_2...m_{10}$ es el mensaje en claro que sería el número de cuenta. Sucesivamente, Carlos debe tomar ese flujo aleatorio y hacer $\oplus$ con su número cuenta, de esa manera obtendría su número de cuenta cifrado y basta con que los ponga y borré los de Alicia.

b) Ejemplifica la situación usando RC4 para cifrar un archivo con número de cuenta 0123456789 y posteriormente hacer el cambio para que el número de cuenta sea 1231231231.
Solución:
Para ejemplificar ésta situación se programó un archivo en python 3 con el nombre de *num_cuenta.py* se realiza el ejercicio imprimiendo la tarea original sin modificar que está en el archivo txt *tarea.txt* y luego la ya modificada donde cambió el número por 1231231231.

Ejercicio 4

En la práctica, es muy común usar la biblioteca OpenSSL para varias tareas relacionadas con criptografía, como generar llaves, cifrar, codificar, entre otras. También hay una aplicación que se ejecuta con el comando openssl, que normalmente se incluye en distribuciones de Linux y MacOS.
Investiga cómo usar openssl para cifrar archivos con los cifradores de flujo RC4 y ChaCha20 usa además una cadena llamada vector de inicialización (IV), para este ejercicio usa una cadena de 16 bytes cero. Obten el cifrado de los siguientes mensajes con las respectivas llaves y escribe el resultado en Base64.

1. m = Este es un mensaje secreto
   k = Una llave muy larga de 32 bytes (ASCII)

   Obteniendo el cifrado y pasándolo a Base64:

   - **RC4:**
     U2FsdGVkX19fPOoZfIJnezuALY8vyb10vMdDzzq250cuDbvxpNLnxAjHNg==
   - **Chacha20:**
     U2FsdGVkX19GKj0aVCuARqhQrQyB2HIt5G6JQFP3/6rEjL+Kcxus0M5qvw==

```
angelajan@angelajan:~/Documentos/Criptologia/Tareas/Tarea4$ openssl enc -rc4 -base64 -e -in ejercicio1.txt -out ejercicio1_rc4.txt -k 556e6120
6c6c617665206d7579206c617267612064652033322206279746573
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
angelajan@angelajan:~/Documentos/Criptologia/Tareas/Tarea4$ openssl enc -iv 00000000000000000000000000000000 -chacha20 -base64 -e -in ejercici
o1.txt -out ejercicio1_chacha.txt -k 556e61206c6c617665206d7579206c617267612064652033322206279746573
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
```

2. m = 060606060606 (son 6 bytes hexadecimal)
   $k = 00^{16}$ (llaves de 16 bytes cero)

   Obteniendo el cifrado y pasándolo a Base64:

   - **RC4:**
     U2FsdGVkX1+cRodKknw0vJTdM9puJzd0732s1m8=
   - **Chacha20:**
     U2FsdGVkX18AbGg/n2kFrgjEDbTifGZujT9IVkY=

```
angelajan@angelajan:~/Documentos/Criptologia/Tareas/Tarea4$ openssl enc -rc4 -base64 -e -in ejercicio2.txt -out ejercicio2_rc4.txt -k "0000000
00000000000000000000000"
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
angelajan@angelajan:~/Documentos/Criptologia/Tareas/Tarea4$ openssl enc -iv 00000000000000000000000000000000 -chacha20 -base64 -e -in ejercici
o2.txt -out ejercicio2_chacha.txt -k "00000000000000000000000000000000"
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
```

3. m = Este es un mensaje secreto060606060606 (ASCII, al final hexadecimal)
   k = Una llave muuy larga de 32 bytes

   Obteniendo el cifrado y pasándolo a Base64:

- **RC4:**
  U2FsdGVkX18UCX6PHfGg/inQm9AcrfyXlCUeEfyRUg46XJKpmfMg8cTdkvbaUSH+ jcFAxPgS/Q==
- **Chacha20:**
  U2FsdGVkX18QusILsCZUBspeCrHj9dJYiZBH1K1ETODyt2Al080khU2I7RAWvqjW yzpTZzzUIg==

```
angelajan@angelajan:~/Documentos/Criptologia/Tareas/Tarea4$ openssl enc -rc4 -base64 -e -in ejercicio3.txt -out ejercicio3_rc4.txt -k 556e6120
6c6c617665206d757579206c617267612064652033322206279746573
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
angelajan@angelajan:~/Documentos/Criptologia/Tareas/Tarea4$ openssl enc -iv 00000000000000000000000000000000 -chacha20 -base64 -e -in ejercici
o3.txt -out ejercicio3_chacha.txt -k 556e61206c6c617665206d757579206c617267612064652033322206279746573
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
angelajan@angelajan:~/Documentos/Criptologia/Tareas/Tarea4$
```