Universidad Nacional Autónoma de México
Facultad de Ciencias
Criptografía y Seguridad
Tarea 3

Luna Vázquez Felipe Alberto
Hurtado Gutiérrez Marco Antonio

18 de abril de 2020

Ejercicios

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* $\frac{6}{\pi^2}$.
   Usa este resultado para probar tres algoritmos generadores de números aleatorios:

   - La función randint del módulo random de Python.
   - 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 $[0, n]$, escogiendo n con un valor de al menos $2^{8.7} - 1$ (¿Cuántos bytes se necesitan para guardar 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 los valores obtenidos

	randint	RC4	urandom
100			
1000			
10000			

   Respuesta:

   El Promedio se representa de la siguiente manera:

   $$Promedio = \frac{6}{\pi^2} \tag{1}$$

   Después tenemos la siguiente igualdad:

   $$\frac{\#Primos}{n} = \frac{6}{\pi^2} \tag{2}$$

   Así:

   $$1 = \frac{6n}{(\pi^2)(\#Primos)} \tag{3}$$

   Despejando π^2:

   $$\pi^2 = \frac{6n}{\#Primos} \tag{4}$$

   Finalmente:

   $$\pi = \sqrt{\frac{6n}{\#Primos}} \tag{5}$$

   Ahora, procedemos a llenar la tabla:

Randint:

```
marco@marco-X510URR:~/Escritorio/Tarea3Criptografia$ python3 randint.py
El número de primos relativos que se contaron con n =  100 fueron 59
El número de primos relativos que se contaron con n =  1000 fueron 610
El número de primos relativos que se contaron con n =  10000 fueron 6035
marco@marco-X510URR:~/Escritorio/Tarea3Criptografia$
```

Figure 1: Ejecutando el programa Ejercicio_1.py

Sustituyendo con $n = 100$ en la ecuación (5):

$$\pi = \sqrt{\frac{(6)(100)}{59}} = 3.188964021 \tag{6}$$

Sustituyendo con $n = 1000$ en la ecuación (5):

$$\pi = \sqrt{\frac{(6)(1000)}{610}} = 3.136250241 \tag{7}$$

Sustituyendo con $n = 10000$ en la ecuación (5):

$$\pi = \sqrt{\frac{(6)(10000)}{6035}} = 3.153094507 \tag{8}$$

RC4:

```
marco@marco-X510URR:~/Escritorio/Tarea3Criptografia$ python3 RC4.py
b'\x1a\xcb\x1bXym\xc4\x9cf=\r\xc9|\xfc({.F\x10G\x01\x12\xbbo\xa4\x84\xb0\
\xe3\xb4\x98\xa9HbE_m\xa9\x88mI\x9a\xba\x92\x06\xfd\x1a3y\xe8Y]~\xdfW\xe3
xc1.\xd4\xec\xc1\xf7\x1e)t\xdde\x0f\x87\xe6@\xb1l\xec_eb~W\t\xa8\xe0c|\xf
c83\x18\x98=h\x0c/d\xc4\xc3\x91\xda\x8b\xb8\xf3R\xf1\x9fx\x1fiRc\xf2]\xe5
x9e:\xf6X\xf4\x80\xa2\xe3]\x94\xf1?(N\\\x04/\xdb\xe5\xb26\xe3\xd2\xc3\xce
97#\xed\x92\x06tZS'
El número de primos relativos que se contaron con n =  100 fueron 59
El número de primos relativos que se contaron con n =  1000 fueron 577
El número de primos relativos que se contaron con n =  10000 fueron 6093
```

Figure 2: Ejecutando el programa Ejercicio_1.py obtenemos

Sustituyendo con $n = 100$ en la ecuación (5):

$$\pi = \sqrt{\frac{(6)(100)}{59}} = 3.188964021 \tag{9}$$

Sustituyendo con $n = 1000$ en la ecuación (5):

$$\pi = \sqrt{\frac{(6)(1000)}{577}} = 3.224688127 \tag{10}$$

Sustituyendo con $n = 10000$ en la ecuación (5):

$$\pi = \sqrt{\frac{(6)(10000)}{6093}} = 3.138051279 \tag{11}$$

Urandom:

```
marco@marco-X510URR:~/Escritorio/Tarea3Criptografia$ python3 urandom.py
El número de primos relativos que se contaron con n =  100 fueron 58
El número de primos relativos que se contaron con n =  1000 fueron 620
El número de primos relativos que se contaron con n =  10000 fueron 6090
```

Figure 3: Ejecutando el programa Ejercicio_1.py obtenemos

Sustituyendo con $n = 100$ en la ecuación (5):

$$\pi = \sqrt{\frac{(6)(100)}{58}} = 3.216337605 \tag{12}$$

Sustituyendo con $n = 1000$ en la ecuación (5):

$$\pi = \sqrt{\frac{(6)(1000)}{620}} = 3.110855084 \tag{13}$$

Sustituyendo con $n = 10000$ en la ecuación (5):

$$\pi = \sqrt{\frac{(6)(10000)}{6090}} = 3.138824103 \tag{14}$$

	randint	RC4	urandom
100	3.188964021	3.188964021	3.216337605
1000	3.136250241	3.224688127	3.110855084
10000	3.153094507	3.138051279	3.138824103

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 mensaje cifrado $c = RC4(s\|k) \oplus m$.
   - Se manda la pareja (s, c).

   Responde lo siguiente.

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

   Respuesta:

   $RC4$ al concatenar s con k siempre nos va a dar la misma semilla esto quiere decir que es un algoritmo determinista, entonces por eso solo basta hacer el XOR con c y regresamos m para recuperar el mensaje claro al recibir (s, c).

   (b) 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 $RC4$?

   Respuesta:

   Tendría que recuperar la lista de mensajes claros como se menciono en el inciso a), después sólo basta hacer XOR s_1 con m y XOR s_2 con m y verificar si el flujo generado es el mismo.

(c) Usando la paradoja del cumpleaños calcula aproximadamente cuántos mensajes tendría que enviar Alicia para que se repita el flujo generado por $RC4$.

Respuesta:

Primero tomemos en cuenta estos datos que son relevantes para el desarrollo del problema:

i. Tenemos una llave secreta compartida k de 128 bits.
ii. Se escoge una cadena aleatoria s de 48 bits.
iii. Con la Paradoja del cumpleaños tenemos que calcular :

$$k = 2^{\frac{n+1}{2}} \sqrt{\ln(\frac{1}{1-\mathcal{P}_k})} \tag{15}$$

Entonces tenemos que considerar sumar los bits de i y ii para obtener la semilla, así:
128 bits + 48 bits = 176 bits. Sustituyendo $\frac{177}{2}$ tomando el piso obtenemos 2^{88} que es aproximadamente cuántos mensajes tendría que enviar Alicia para que se repita el flujo generado por $RC4$.

(d) 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?

Respuesta:

$RC4$ es un algoritmo determinista es decir dada una entrada el algoritmo va devolver siempre la misma salida, por . salida, entonces si omitimos la cadena s ya solo quedaría la llave k entonces solo tendríamos el flujo del XOR y este flujo se repetiría como en el algoritmo de one time pad. Por lo que Alicia y Bartolo usan s concatenada con k para generar una semilla aleatoria. Entonces podemos concluir que solo un mensaje puede cifrarse de forma segura ya que si solo utilizas k sería darle mucha ventaja al adversario porque se repite el flujo.

3. El profesor Bartolo guarda las tareas de sus alumnos encriptadas 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 ultima 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 encriptar 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 descifre el archivo que originalmente era de Alicia ahora tendrá el número de cuenta de Carlos.

Respuesta:

Cómo Carlos conoce el número de cuenta de Alicia y el formato que tienen todas las tareas, Carlos debe encontrar la manera de reemplazar los bytes cifrados del número de cuenta de Alicia por los de su número de cuenta.

Para lograr esto Carlos hace un cifrado XOR entre los bytes generados por $RC4$ de la tarea de Alicia y la parte en claro de la tarea de Alicia (formato), el resultado le daría parte del flujo generado por $RC4$ así el ya podría identificar el texto que quiere reemplazar (ya que se sabe el formato exacto de cada tarea). Por último debe hacer XOR entre los bytes generados por $RC4$ y su número de cuenta (en claro) y sustituir los bytes del número de cuenta de Alicia por los suyos.

(b) Ejemplifica la situación usando $RC4$ para encriptar un archivo con número de cuenta 0123456789 y posteriormente hacer el cambio para que el número de cuenta sea 1231231231.

El código que se deja en la carpeta "$src/Ejercicio - 3.py$" hace lo siguiente:

```
if __name__ == "__main__":
    tarea_alicia = b"""No. de cuenta: 0123456789\nTarea 3\nRespuestas...
                    """
    print(tarea_alicia.decode("utf-8"))

    num_cta_carlos = b'1231231231'
    tarea_alicia_cifrada = rc4_encrypt(b'0123456789', tarea_alicia)
    num_cta_carlos_int = bytes_to_int(num_cta_carlos)
    tarea_alicia_cifrada_int = bytes_to_int(tarea_alicia_cifrada)

    flujo_aleatorio = []
    for i in range(len(tarea_alicia_cifrada_int)):
        flujo_aleatorio.append(tarea_alicia[i] ^ tarea_alicia_cifrada_int[
    i])
    reemplazo = []

    c = 15
    for i in range(len(num_cta_carlos_int)):
        reemplazo.append(num_cta_carlos_int[i] ^ flujo_aleatorio[c])
        c += 1
    tarea_alicia_cifrada_int[15:25] = reemplazo

    decrypt = rc4_encrypt(b'0123456789', tarea_alicia_cifrada_int)
    print(decrypt.decode("utf-8"))
```

Y estos son los resultados:

```
TAREA ALICIA (TEXTO CLARO)
No. de cuenta: 0123456789
Tarea 3
Respuestas...

TAREA ALICIA CIFRADA
8d639aeb7f54fa3a3e49734a62f559b6ba19befa388758f72f69e8fa3a299428a67b60fd36194db94b
194798e13e0a037810797c448903a66bd51da35ad4d782a4428b003d653c56224935421be6283d07b0
3aea966cedbc2429603d55661e7c7339fcadb141cd0c0fad6b80d52b451b

XOR TAREA ALICIA CLARO Y TAREA ALICIA CIFRADS (FLUJO A LEATORIO)
c30cb4cb1b31da594b2c1d3e03cf79868b2b8dce0db16fcf1663c8da1a09b408865b40dd16396d996b
3967b8c11e2a232c710b1925a930ac4bf53d837af4f7a28462ab201d451c76026915623bc67a5874c0
4f8fe5188ccf0a074e3775463e5c5319dc8d9161ed2c2f8d4ba0f50b653b
```

```
XOR NUM CTA CARLOS Y FLUJO ALEATORIO (REEMPLAZO)
b7b918bcfc3e805dfc27

RESULTADO CIFRADO
8d639aeb7f54fa3a3e49734a62f559b7b918bcfc3e805dfc2769e8fa3a299428a67b60fd36194db94b
194798e13e0a037810797c448903a66bd51da35ad4d782a4428b003d653c56224935421be6283d07b0
3aea966cedbc2429603d55661e7c7339fcadb141cd0c0fad6b80d52b451b

TAREA ALICIA (TEXTO CLARO)
No. de cuenta: 1231231231
Tarea 3
Respuestas...
```

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 en MacOS.

   Investiga cómo usar openssl para encriptar archivos con los cifradores de flujo RC4 y ChaCha20. ChaCha20 usa además una cadena llamada vector de inicialización (IV), para este ejercicio usa una cadena de 16 bytes cero. Obtén el cifrado de los siguientes mensajes con las respectivas llaves y escribe el resultado en Base64.

   (a) m = Este es un mensaje secreto, k = Una llave muuy larga de 32 bytes (ASCII).

   (b) m = 060606060606 (son 6 bytes en hexadecimal), $k = 00_{16}$ (llave de 16 bytes cero).

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

   En OpenSSL se ocuparon las siguientes banderas:

   (a) *enc*: Especifica el tipo de cifrado a usar, que en nuestro caso ocupamos *rc*4 y *chacha*20.

   (b) *base*64: Especifica que el resultado del cifrado se va a escribir en el archivo de salida en Base64.

   (c) *iv*: Vector de inicialización, necesario como parámetro de ChaCha20.

   (d) *e*: Indica que se cifrarán los datos de entrada.

   (e) *in*/*out*: Ruta del archivo de entrada/salida.

   (f) *k*: Llave que se ocupará para cifrar.

   A continuación se muestran los resultados:

```
== RC4 ==

openssl enc -rc4 -base64 -e -in /E4_in.txt -out /E4_out.txt
-k "Una llave muuy larga de 32 bytes"

R = U2FsdGVkX1/BMU0zHrcSiyybZ8OFB/P2gKZ48XmdQu9h8X7t1UYCUwDLUQ==

openssl enc -rc4 -base64 -e -in /E4_in.txt -out /E4_out.txt
-k "00000000000000000000000000000000"
R = U2FsdGVkX1/TJaTaEM8/dibGsAFNVLk/q5We2oQ=
```

```
openssl enc -rc4 -base64 -e -in /E4_in.txt -out /E4_out.txt
-k "Una llave muuy larga de 32 bytes"
R= U2FsdGVkX1+jAq68XJnS3YxVOhbCzZjYQMy
mMVi7qHQEwA/fgxpXUlOq9nR4zuspfWWno7kI6g==

== SHA SHA 20 ==
openssl enc -iv 00000000000000000000000000000000 -chacha20 -base64
-e -in /E4_in.txt -out /E4_out.txt -k "Una llave muuy larga de 32 bytes"
R = U2FsdGVkX19UYG5nPrLz32W7C8/MRMIxCUh4YP+oatJYUUjA8AKGTTaFqQ==

openssl enc -iv 00000000000000000000000000000000 -chacha20 -base64
-e -in /E4_in.txt -out /E4_out.txt -k "00000000000000000000000000000000"
R = U2FsdGVkX1/eHExKQlnfwmIWVR27eXebg2GO1p4=

openssl enc -iv 00000000000000000000000000000000 -chacha20 -base64
-e -in /E4_in.txt -out /E4_out.txt -k "Una llave muuy larga de 32 bytes"
R = U2FsdGVkX1/Agf9pWmnUAMRJtEZ88TY
6IynqDV1JW5EjfSco/EP5cOeg6Bw3P5BqDtKmf+tQzw==
```