Tarea 2

Rebollar Pérez Ailyn Ángeles Martínez Ángela Janín

Ejercicios

1. Alicia y Bartolo escogen un espacio de claves $\mathcal{K}$ que contiene 2^{56} claves. Supón que Eva tiene una computadora que puede revisar 10^{10} claves por segundo.

   a) ¿Cuántos días le tomaría a Eva revisar todas las claves de $\mathcal{K}$?

   A la computadora de Eva le tomaría 7205759.40379 segundos revisar el espacio de claves K, este resultado fue obtenido a partir de:

   $$\frac{2^{56}}{10^{10}}$$

   ahora, consideremos que un día tiene 24 horas, cada hora tiene 60 minutos y cada minuto, 60 segundos. Entonces:

   $$\frac{7205759,40379}{86,400} = 83,3999930995\ días$$

   b) Si Alicia y Bartolo cambian su esquema por uno con un conjunto más grande, con 2^B claves, ¿qué tan grande debe ser B para que la computadora de Eva tarde 100 años revisando todas las claves? (Puedes suponer que un año tiene 365,25 días.)

   Para calcular el valor de B primero obtuvimos los segundos que tendrían los 100 años y aplicamos una regla de 3.
   (60)(60)(24) = 86400 seg - 1 día
   3155760000 seg - 365.25 (100) = 36525 días

   Ahora tenemos que despejar B:
   $2^B = 3155760000(10^{10})$
   $log_2(2^B) = log_2(3155760000) + log_2(10^10)$
   $B = 64.7746$
   Por lo tanto B debe ser 64.7

2. ¿Los siguientes esquemas de cifrado son perfectamente seguros? Explica.

a) Los mensajes claros son $\mathcal{M} = \{0, 1, \ldots, 9\}$. El algoritmo KeyGen devuelve una clave al azar del conjunto $\mathcal{K} = \{0, 1, \ldots, 10\}$. La función $\mathsf{Enc}_k(m)$ calcula $(k + m)$ mód 10, y $\mathsf{Dec}_k(c)$ devuelve $(c - k)$ mód 10.

Es posible ver que $|M| < |K|$, además, por medio del enunciado no podemos asegurar que $|C| = |M|$. Por lo tanto, no podemos asegurar que una llave diferente sea dada para cada mensaje que vaya a ser cifrado. Por lo tanto, no es un cifrado perfectamente seguro, pues incluso, existe un conjunto mayor de llaves que de posibles mensajes, por lo tanto, es posible que la llave se repita si es dada de forma aleatoria.

b) El algoritmo de desplazamiento (César) para mensajes de tamaño uno sobre el alfabeto `ABC...Z` de 26 letras.

Debido a que para el adversario, conocer el texto cifrado de tamaño 1 de un texto cifrado con César no le proporciona información respecto del texto plano, pues aunque intente por fuerza bruta decifrar el mensaje probando con todo el alfabeto, una única letra no significa mucho para que realice el criptoanálisis y la probabilidad se mantiene.
Por lo tanto, sí es un cifrado perfectamente seguro, si podemos asegurar que cada llave se usará solamente una vez, y además, éste puede tener como máximo 26 textos claros y textos cifrados.

c) One-time pad para mensajes de longitud ℓ, usando únicamente llaves distintas de $k = 0^\ell$. Esto es para evitar que un mensaje cifrado sea exactamente el mensaje claro.

OTP sin la modificación de usar únicamente llaves distintas de $k = 0^\ell$, cumple seguridad perfecta. Sin embargo, para que un cifrado sea perfectamente seguro, debe cumplirse que $M = K = C$, donde M es el conjunto de posibles mensajes, K el conjunto de posibles llaves y C el conjunto de posibles cifrados.
Particularmente para OTP, entonces, $K = M = C = \{0, 1\}^n$.
Particularmente, para OTP usando llaves distintas de $k = 0^l$, tenemos $|K| = |M| - 1 < |M|$, por lo tanto, no es perfectamente seguro.

3. Definimos Π como una versión modificada de one-time pad, donde $\mathcal{M} = \{0, 1\}^\ell$, pero ahora $\mathcal{K}$ son las cadenas de ℓ bits con un número par de unos; el cifrado y descifrado son iguales que en one-time pad. Construye un adversario

$\mathcal{A}$ tal que $\Pr[\mathsf{PrivK}_{\mathcal{A},\Pi} = 1] = 1$, es decir, un adversario que siempre gana el juego $\mathsf{PrivK}_{\mathcal{A},\Pi}$.

El adversario $\mathcal{A}$ como sabe que el cifrado y descifrado se realiza con one-time pad entonces sabe que puede armar el siguiente sistema de ecuaciones para deducir cuál es la llave:

$$k_0 = m_0 \oplus \text{c} \ldots (1)$$
$$k_1 = m_1 \oplus \text{c} \ldots (2)$$

Sin embargo, $\mathcal{A}$ debe de tomar algo muy importante para elegir correctamente la llave, recordar que k debe tener un número par de unos, así que supongamos que el adversario tiene una función llamada *parUnos(llave)* donde recibe una llave y cuenta el número de unos, si tiene un número par de unos regresa true y false en otro caso.

Entonces un pseudo-código que tiene $\mathcal{A}$ para saber qué mensaje elegir es:

if *parUnos*(k_0):
  return m_0
else:
  return m_1

Podemos ver que ésta implementación nos garantiza que el adversario pueda elegir siempre el mensaje que fue cifrado con una llave correcta, también notemos que cada mensaje m_0 y m_1 tienen siempre diferente llave porque sino, implicaría que $m_0 = m_1$ lo cual no sería propio del cifrado one-time pad porque una llave con dos diferentes mensajes nos estaría llevando al mismo cifrado lo cual no sería conveniente y no se deja la decisión al azar.

Entonces podemos asegurar que $\mathcal{A}$ siempre gana el juego de $\mathsf{PrivK}_{\mathcal{A},\Pi}$ porque cuando checa el número par de unos de la llave correcta, la probabilidad de que escoja es de 1 mientras que la probabilidad de que escoja a la otra llave es de 0.

4. Sea Π el esquema de Vigenère, donde $\mathcal{M} = \{\mathtt{a}, \mathtt{b}, \ldots, \mathtt{z}\}^3$, la clave se genera escogiendo aleatoriamente un número $t \in \{1, 2, 3\}$ y luego se escoge una clave aleatoria de tamaño t.

   En el juego $\mathsf{PrivK}_{\mathcal{A},\Pi}$ un adversario $\mathcal{A}$ entrega $m_0 = \mathtt{aab}$ y $m_1 = \mathtt{abb}$. Cuando se le da un texto cifrado $c = c_1c_2c_3$, devuelve 0 si $c_1 = c_2$ y 1 en caso contrario.

Construye un mejor adversario que $\mathcal{A}$, es decir, un adversario $\mathcal{A}'$ tal que $\Pr[\mathsf{PrivK}_{\mathcal{A}',\Pi} = 1] > \Pr[\mathsf{PrivK}_{\mathcal{A},\Pi} = 1]$.

5. Como hemos visto, un archivo cifrado normalmente está formado por bytes que no tienen una representación como texto. Una forma muy común de almacenar bytes arbitrarios como una cadena de texto consiste en usar la codificación Base64.

   Recuerda que no es lo mismo cifrar (encriptar) que codificar, pues la codificación solamente sirve para representar de distintas formas la información, no tiene el propósito de ocultarla o proteger su integridad.

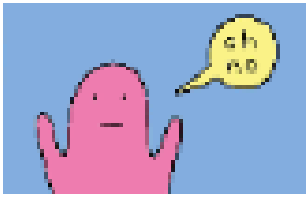
   Para usar Base64 existen programas o bibliotecas en cualquier lenguaje de programación. Busca alguno y decodifica el siguiente contenido, que originalmente es una imagen en formato PNG.

```
iVBORw0KGgoAAAANSUhEUgAAAFEAAAAzCAIAAABgOXMwAAAAAXNSROIArs4c6QAAAARnQU1BAACx
jwv8YQUAAAAJcEhZcwAADsMAAA7DAcdvqGQAAAn5SURBVGhD5ZoHVFRXGsfvvW/eMHQUBRUrYjTL
GkuMGNfusa/ILpYYG8paWAvrMWuOOUYl6AaNikmM5WStMa6KGLtYYhdjY1UELERZK9gow8zAvHL3
m3l3hFFmIIODcvI7R+Z+990n7z/3fuXeB45NvIN+ZxD2+XuCIIpZs5oBz00ppgjDP1MbetiV8sBf
JGaZb6hmYE5KWru0UPuIw0SWJXeNpnfEbMy7s8t2wfMTMynimPXWA5Mj6J/vWj6jMPtKWtpSjv7K
IS8qS3oq9un1Qz5pMnhyHO/uw0bbADTfokjFrLcbWMMHNi3rHHT3n9EfenmJ2DoYwdWLFx9vO/Dw
QoZ333GzsG2frTbzDDO8c8U8P+OxXXujcGk5ijZwagWKJNmr34TzPYZ9jnHZuqpN3C549qDrB893
7YtC1vO3YPF5mUjMADBSqQrC2qGivBzW8wrVRvPJjdFxn/cEvcRqltHqFfsItV7kFE2aGLx5yWhb
Kal6aE7/5cjo8GAsGaAtybjBu1/GLTmvXCoWpXMpxsXxpynilR6AEEkjGDCSmWONaK4GiSr1+NZJ
kX9U2lHRCWmXpn30cTeCasDDh4d39vGUIsa1+Oqb08oAhcNH59zNvMwMa4hA1Kz5tkJFQ6dOgRQz
p+0UOmzrtqxpO74T5WfgvgOC/JoG+Xt5+EhGrTJAoYZn3uXj+5hhDY7ZkUXAA95isrPS+7U41ruz
L7Mpuv8gj1PhunW8wdLpBTc3NSzjAm2xl6dGGWKCku7DM/pGzmVmKaqBPxuK9JgWMgPAqH59HOUw
4O7Gm+tIbCXYjK2ZrAaaKQRqWnYOUoABMqdjhgXKawS56vIz5YiQfQ+qQtHyXVfKdzQu3i6u9ZiB
kCjjgkJNOs/XU9MfmUsRPC9mf/jQHWk3n7IRZo6dymnXdSAzrCGVehxrVKKUtO7L/8RGXFk+t8vD
zAdrFm6cM3xD7Dg3XExlx38PRkV7diUzw5Sr3CZMSfT25qdM24EQr9MZebVrwuYRiT9dEcWS5TAx
clOd5m2YYc1rqDOh8Rfr806umu2nO8XOGNHArbZgmVgOY5FK4zfGuHfv4/GHrvWCgpX+CkNz7t8W
MhbGx/VWEVg1JmSKW7ZZnnZ58tNclPLfzE4dGsWvTunWse6vd4SRQwMJMdUhj3NOgyLPhOWvLHMh
cz2GTavMCpcwepB+8v6e9cs6DB7WorMH71ba8OwrD+HQVt0bFdNtW77lmjT18a3LrpUHodKz25eb
kh2zZ3YEZ2G98B9i1DI4oFEjr3xd7qVf7oWENPb388x5lN/mPd+aNV1hgIxxwtYrvu1m866eyi0v
Ucl5lh/euHBtddzuMTFUthdmAK1eOzcvtemAvzPbLsUG3d7v53UKzl4a19+8kpDOgARR8PFkxRaG
uEVEImNqKTyhh5qSLr1z98HYT7V9I+ZiUvZcQq/jnqaS8fP9P24ZPLNcwYCnm6fvtdtigc3S3wKE
YSk1KW56hGbJot6ghWIh/utTE2demfpZ+pzYw0oNDfKwxLOQDJgFo7MXMnv229q6+8e2BAOOzzMU
8Kum9708aoGqwrdDlA369m/R685h27cUPr975oe5R/YMIFjU6aWneU9HRe5uP2Be7eadKaZJa2KG
9y8O7dnU189Dbb3rz9cW5RYW9Rp0fMzc9bz65VxdGsfPPWFVNz51NLJt75c2OnYgCK+4+XNuzyFe
tRqwrlJgJB5P+Casi35wWLMli5IMxdSgap4rNfQLbBfQpLl5CEWyfPtasqR96u+eaXhyDeIZz1FB
hE+ilevpuNbt+wyFZGYebBPHz8Ou7vl3jHdgHc+azK4YOqMu6nlaSL8JzLYAS2DB2HYL5w+Z98V2
TDw+WXFQgPISHozaXBEC4dWSSIkgQ8lCECcTbHtwaSCOOyJYhYtrpaW+EAxZfn3KQWi89A3D+t95
oyS1ApwEpUoZngYju/914m0aGr3y9D9WHROJGoOv2tXAy4Lp3JOqCOVVEl9BwYBNR7dP3OSxs/uP
ZAbURgRtun5S/bJklGcoXH1uLzPMaNxdR7btyAxrPgwd79vgXcK5MNtpOKKZIMlTynItKrmXUHxg
dGyxyeGsqKHx2BcxnxlmOIou7YT66U3ikGajfuv4f2GruEl5sWw3IeBqpZAgpchGZNkMvxEc0Xz2
RJJ/EWy6HQkEJiRBKnWOU/XYOyxKxUumDryw73tmv4AaTFsmR5FVSEWNzHgT2NRsNGpvr51/deKc
cL10IznRdJpoAfKCY9FeQYZ8iqwClanwStmffvYnJBawLmdiU7MnLZjUrIOHwA8K+hN37SwuKjlt
8qa8qdJ3FDXiYP/ADDOp5w8My86bruMSFO2GKpr1Og2bml2oXN/HDxoykuI/GJN8aJPSDymx8zst
vL3YOUxpJDVZfW73rDOrP9u/elzCooiEhTEHN7ySvyCvyrjUKTwsGZ+LydODgoN8GOr3HpEKp1mH
salZJ+Az6SlKW5bFPoHvKW1ORlf3biZlnQGoBDqhY+j8sKgFg6LWDJu5fujMmD5jXnWCdwIaY67k
9/4YP2v0+91gW2wyJNirlL9dqSQ2NedR1OeGPKUNW//s4OckyDEmJGQoLnNlm/c6UGkhLMgYnpya
```

```
fryKm4t7yeslJOnunA1W+ytGkUmw45GigtjUDN+3xLGMAoUhJiI2O6GR8DaOVJwSVRnnDq7vOVm2
BEijmsrOT2M2NXt5+uYFBjADoYZuvoLeFMbgBnjASs2FZZLhQ8h/HFSvoWLCpt9bVaMKDtttaoa5
PZN1EOpFhb+O7rItPhoaLpws4kq9r37hF+AJLZ/keqjYXjfl3q2QjyKVtlOxs7YJ715Lh1nxAAlG
zsmSREEw6r1dKvW6p3FAQyqazrf0+VpZV5ICr+Xdd2vajhnOxI5m1HXQqE8Or1WmBeb7xJSVKccT
jmz+alLHP5v7HKSWq0/qoW2wCVwbOyIqhB1Bw25UU6u2C18VNak9zUjlcTTjPG95W+9ahN6/dyv7
yIkanIfS4xjeourg9q/PbF++JWyqClzIjITxujsZHj6Wl1LOxK5mhNqHT0198j9mIDS11aBz0csc
312YgdvTJ6wMzS9qq/F/kbNkTpZd+DJT4GunHM2tug05mplS8iim19ivIbC6EH54217UfP6uELp0
xsCITOsFOCdSjmYJC7DqKi/SPiDOPqdFKtOJfBVQzt8Bumg8Ant0c/qXj7Go9oaoxkwnQ+xKRoTj
N1xIhmqB2c5hb0bylCWbnL6cLJR/7ikTdTkOUGlyC59pBadvp15QvpyWnfpvvH7KqRMt1PJX8fbe
PLxeytdcs2HzGXu/A79nthM4oyt0ca1Uzv9NlK8Z3Cw8anG6/jGzXzewIX8sGypz8PJbqZCrBnXs
eyjV6nXEaOTSqKhlm1EVIPR/XQXt9t25L+8AAAAASUVORK5CYII=
```

También podemos escribir la representación hexadecimal de los bytes originales. Entre Base64 y la representación hexadecimal ¿cuál ocupa más espacio? ¿Por qué?

La imagen obtenida es la siguiente (también se encuentra en la carpeta, junto con el script que fue usado para obtenerla):



Base64 es más eificiente en cuanto a espacio frente a a representación hexadecimal. Esto de debe a una razón: el charset. Base64 tiene un charset de 64 caracteres, mientras que la representación hexadecimal (como su nombre lo indica), tiene 16 caracteres. Por lo tanto, las representaciones en Base64 son más cortas.
De hecho, la ineficiencia en espacio de hexadecimal se encuentra en que el tamaño de los datos aumenta en un 100 % al ser codificado. Por otra parte, base64 aumenta solamente 33 % de los datos al codificar.

6. Para cifrar un mensaje usando One-time pad necesitamos una cadena aleatoria de bytes. Supongamos que Bartolo quiere enviar un archivo a Alicia, entonces Juan Aleatorio le proporciona el archivo `cinta_aleatoria.txt`, que contiene una cadena de bytes k en Base64. Bartolo usa k como llave para cifrar el archivo `imagen.png` y envía el resultado c a Alicia.

   Alicia, que también posee k, al recibir el mensaje cifrado c revisa los primeros bytes y nota algo extraño: c no parece algo aleatorio, sino que es un mensaje de un formato muy particular.

Cuando Alicia descifra c obtiene el mensaje correcto (el archivo `imagen.png`), y entonces se da cuenta de que la llave no es una cadena tan arbitraria de bytes, al parecer Juan Aleatorio conocía el mensaje que Bartolo quería enviar.

a) Cifra `imagen.png` con la cadena contenida en `cinta_aleatoria.txt` (revisa que esta llave tiene la misma longitud que el mensaje) para obtener el mensaje c y descubre por qué Alicia notó algo extraño.

R= Para cifrar la imagen se escribió un script llamado *ejercicio7.py* en python 2 donde se implementó la función *encriptaImagen()*, *guardaCifrado(cifrado)*, *imprimeBytes(cifrado, num)* y *checaLongitud(llave,imagen)* con las cuales ciframos la imagen, checamos la longitud del mensaje y la llave pero se implementó la función *imprimeBytes(cifrado, num)* donde teniendo el cifrado que recibió Alicia podemos imprimir o ver los primeros bytes que fueron 255 y 251 los cuales pertenecen a la cabecera de un formato .mp3, por eso es que no parece aleatorio el cifrado.

b) Suponiendo que k efectivamente es una cadena aleatoria de bytes, ¿cuál es la probabilidad de que $m \oplus k$ sea el c que recibió Alicia?

R= La probabilidad que m $\oplus$ k sea c está dada por lo siguiente:

- Recordemos que la longitud de la imagen a encriptar o cifrar es de 400663 bytes, por lo que de alguna manera se requiere cubrirlos con caracteres diferentes y 1 byte funciona para cifrar 256 caracteres diferentes, entonces tendríamos que 256^{400663} para tener diferentes combinaciones de cubrir la imagen.

- Pero recordemos que lo que se busca en one-time pad es que un mensaje cifrado con la llave te lleve a un sólo cifrado, es decir que sea una función biyectiva, entonces sólo una combinación de las 256^{400663} debe dar el mensaje en claro.

Entonces tomando en cuenta estos dos puntos la probabilidad es de $\frac{1}{256^{400663}}$

c) ¿Qué hizo Juan Aleatorio para crear el archivo `cinta_aleatoria.txt`?

R = Lo que hicimos fue en la función *guardaCifrado(cifrado)* guardar el cifrado con formato .mp3, reproducimos el audio y era una canción, así que el cifrado efectivamente pertenecía a un audio, entonces como

sabemos el cifrado de one-time pad consiste en hacer XOR del mensaje con la llave.

$$c = m \oplus k$$
$$k = m \oplus c \quad \text{despejando a k.}$$

Así que Juan Aleatorio debió conocer la imagen que era el mensaje a cifrar y el cifrado que era el audio para poder despejar la llave.