PONTIFICIA UNIVERSIDAD CATOLICA DE CHILE
ESCUELA DE INGENIERIA
DEPARTAMENTO DE CIENCIA DE LA COMPUTACION

Criptografía y Seguridad Computacional - IIC3253
Tarea 2
Plazo de entrega: 15 de junio

Instrucciones

Cualquier duda sobre la tarea se deberá hacer en los *issues* del repositorio del curso. Si quiere usar alguna librería en sus soluciones debe preguntar primero si dicha librería está permitida. El foro es el canal de comunicación oficial para todas las tareas.

Entrega. Para entregar esta segunda tarea usted deberá utilizar el mismo repositorio que usó para la tarea 1. Al entregar esta segunda tarea, la estructura de archivos de su repositorio se deberá ver exactamente de la siguiente forma:

```
Repositorio
├── Tarea 1
│   └── ...
├── Tarea 2
│   ├── requirements.txt
│   ├── Pregunta 1
│   │   └── pregunta1.pdf
│   └── Pregunta 2
│       └── pregunta2.ipynb
├── .gitignore
└── README.md
```

Además, deberá considerar lo siguiente:

- El archivo `requirements.txt` deberá especificar todas las librerías que se necesitan instalar para ejecutar todas las respuestas en código de su tarea. Este archivo debe seguir la especificación de Pip, es decir se debe poder ejecutar `pip install -r requirements.txt` suponiendo una versión de `Pip` mayor o igual a 20.0 que apunta a la versión 3.9 de Python.
- La solución de cada problema de programación debe ser entregada como un Jupyter Notebook (esto es, un archivo con extensión `ipynb`). Este archivo debe contener comentarios que expliquen claramente el razonamiento tras la solución del problema, idealmente utilizando

markdown. Más aun, su archivo deberá ser exportable a un módulo de python utilizando el comando de consola

```
jupyter-nbconvert --to python preguntaX.ipynb
```

Este comando generará un archivo `preguntaX.py`, del cual se deben poder importar las funciones que se piden en cada pregunta.

- Para cada problema cuya solución se deba entregar como un documento (en este caso la pregunta 1), usted deberá entregar un archivo `.pdf` que, o bien fue construido utilizando LaTeX, o bien es el resultado de digitalizar un documento escrito a mano. En caso de optar por esta última opción, queda bajo su responsabilidad la legibilidad del documento. Respuestas que no puedan interpretar de forma razonable los ayudantes y profesores, ya sea por la caligrafía o la calidad de la digitalización, serán evaluadas con la nota mínima.

Preguntas

1. En clases se presentó en detalle la estructura del protocolo criptográfico simétrico DES (Data Encryption Standard). En particular, vimos que DES es una red de Feistel de 16 pasos, cada uno de los cuales consiste en una red de sustitución/permutación de una sola ronda. El único punto que no se discutió en detalle fue la forma en la que se derivan las sub-llaves: en cada paso de la red de Feistel es necesario aplicar una función $f(k_i, R_i)$, donde f es la red de sustitución/permutación, R_i es un estado interno de la red y k_i es la *sub-llave* correspondiente al paso i. Esta sub-llave se deriva de forma determinista en base a la llave inicial de acuerdo a lo que se conoce como el *key schedule*.

   (a) Investigue y describa en detalle el *key-schedule*, es decir, explique cómo se deriva la llave correspondiente a cada ronda en base a la llave original.

   (b) Considerando que el *key-schedule* divide la llave en dos mitades que son prácticamente *independientes*, demuestre que si en vez de 16 rondas DES utilizara sólo 3 rondas, entonces existiría un ataque de texto plano que recupera la llave sin ejecutar la red de sustitución/permutación más de 2^{30} veces.

2. En esta pregunta usted deberá implementar en Python el protocolo criptográfico de clave pública RSA. Para esto deberá implementar las siguientes funciones, sólo pudiendo suponer como dadas las funciones aritméticas básicas para números enteros (suma, resta, multiplicación, división y resto).

   - Una función que implemente el test de primalidad de Rabin-Miller, y una función que utilice este test para generar un número primo con una cantidad de dígitos dada como parámetro:

```python
def rabin_miller(n: int, k: int) -> bool:
    # Argumentos:
    #   n: int - n >= 1
    #   k: int - k >= 1
```

```
    # Retorna:
    #   int - True si n es un numero primo, y False en caso contrario.
    #         La probabilidad de error en el test debe ser menor o
    #         igual a 2**(-k)

def generar_primo(l: int) -> int:
    # Argumentos:
    #   l: int - l >= 1
    # Retorna:
    #   int - numero primo con al menos l digitos. La probabilidad de
    #   error en la generacion debe ser menor o igual a 2**(-100)
```

- Una función que implemente el algoritmo extendido de Euclides:

```
def alg_ext_euclides(a: int, b: int) -> (int, int, int):
    # Argumentos :
    #   a: int
    #   b: int - a >= b >= 0 y a > 0
    # Retorna :
    #   (int, int, int) - maximo comun divisor MCD(a,b) entre a y b,
    #   y numeros enteros s y t tales que MCD(a,b) = s*a + t*b
```

- Una función que implemente el algoritmo de exponenciación rápida en módulo un número:

```
def exp_mod(a: int, b: int, n: int) -> int:
    # Argumentos:
    #   a: int - a >= 0
    #   b: int - b >= 0
    #   n: int - n > 0
    # Retorna:
    #   int - a**b en modulo n
```

- Una función que retorne el inverso de un número a en módulo un número n:

```
def inverso(a: int, n: int) -> int:
    # Argumentos:
    #   a: int - a >= 1
    #   n: int - n >= 1, a y n son primos relativos
    # Retorna:
    #   int - inverso de a en modulo n
```

- Una función que permite crear claves públicas y privadas de un cierto largo:

```
def generar_clave(l: int):
    # Argumentos:
    #   l: int - largo de las claves a ser generadas
    # Retorna:
    #   genera una clave privada (d,N) y una clave publica (e,N) tales
    #   que d, e y N tienen al menos l digitos. La clave privada debe
    #   ser almacenada en un archivo private_key.txt en el formato:
    #   d
    #   N
    #   La clave publica debe ser almacenada de la misma forma en
    #   en un archivo public_key.txt
```

- Un par de funciones que permitan cifrar y descifrar mensajes utilizando claves públicas y privadas ya generadas:

```
def enc(m: int) -> int:
    # Argumentos:
    #   m: int
    # Retorna:
    #   int: cifrado de m de acuerdo con la clave publica almacenada
    #   en public_key.txt

def dec(m: int) -> int:
    # Argumentos:
    #   m: int
    # Retorna:
    #   int: descifrado de m de acuerdo con la clave privada
    #   almacenada en private_key.txt
```