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

Criptografía y Seguridad Computacional - IIC3253
Rúbrica Tarea 2

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 lugar de 16 rondas DES utilizara sólo 3 rondas, entonces el protocolo se podría atacar en no más de 2^{30} pasos.

Corrección. La corrección de cada item se llevará a cabo de acuerdo a lo especificado en las tablas que se presentan a continuación.

(a)

0 puntos	En base a la respuesta no se puede obtener nada de información correcta sobre el funcionamiento del key-schedule de DES.
1 punto	Se puede obtener cierta información correcta sobre el funcionamiento del key-schedule de DES, pero no es suficiente como para entenderlo desde una mirada de alto nivel.
2 puntos	Permite tener una mirada de alto nivel de cómo funciona el key-schedule de DES, pero es muy informal y se salta ciertos detalles relevantes como las rotaciones o las funciones PC_1 y/o PC_2.
2.9 puntos	Explica el key-schedule de DES con todo detalle pero tiene algún error menor, por ejemplo menciona que las rotaciones son iguales en todas las rondas.
3 puntos	Explica el key-schedule de DES con todo detalle y sin errores.

(b)

0 puntos	O bien de la respuesta no se puede deducir un ataque concreto, o bien propone un ataque claramente incorrecto que no llevará a la recuperación de la llave, o bien propone un ataque que utiliza la red de sustitución/permutación más de 2^{50} veces.
1 punto	Presenta la idea de un ataque que podría funcionar, pero a muy alto nivel y sin presentar ningún detalle. Lo descrito en la respuesta es insuficiente para llevar a cabo el ataque.
2 puntos	Presenta un ataque que permitiría efectivamente obtener la llave usando la red de sustitución/permutación no más de 2^{40} veces, pero la descripción es de alto nivel y faltan detalles.
2.5 puntos	Presenta un ataque que permitiría efectivamente obtener la llave usando la red de sustitución/permutación no más de 2^{32} veces. La descripción es suficiente para que alguien entienda cómo llevar a cabo el ataque aunque falten detalles menores.
2.9 puntos	Presenta en todo detalle un ataque que permitiría efectivamente obtener la llave usando la red de sustitución/permutación no más de 2^{32} veces.
3 puntos	Presenta en todo detalle un ataque que permitiría efectivamente obtener la llave usando la red de sustitución/permutación no 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 Miller-Rabin, y una función que utilice este test para generar un número primo con una cantidad de dígitos dada como parámetro:

```python
def miller_rabin(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:

```python
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 >= 2, 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 - 0 <= m <= N-1, suponiendo que la clave publica
    #   almacenada en public_key.txt es (e,N)
    # Retorna:
    #   int: cifrado de m de acuerdo con la clave publica almacenada
    #   en public_key.txt

def dec(m: int) -> int:
    # Argumentos:
    #   m: int - 0 <= m <= N-1, suponiendo que la clave privada
    #   almacenada en private_key.txt es (d,N)
    # Retorna:
    #   int: descifrado de m de acuerdo con la clave privada
    #   almacenada en private_key.txt
```

Corrección. Para corregir esta pregunta se realizarán dos grupos de tests, cada uno con 20 elementos. El primer grupo de tests estará basado en la generación de primos de 100, 200 y 300 dígitos, para lo cual se va a considerar un tiempo máximo de 5 segundos. El segundo grupo de tests estará basado en la generación de claves de 200, 400 y 600 dígitos, y su utilización para cifrar y descifrar mensajes, para lo cual se va a considerar un tiempo máximo de 10 segundos. Para ambos grupos de tests se dará el siguiente puntaje:

1 punto	El código entrega entre 20% y 49% de respuestas correctas.
2 puntos	El código entrega entre 50% y 94% de respuestas correctas.
3 puntos	El código entrega entre al menos un 95% de respuestas correctas.

Así, por ejemplo, si para el primer grupo de test obtiene 7 repuestas correctas, y para el segundo obtiene 15 repuestas correctas, su puntaje para el primer grupo será de 1 punto (35% de respuestas correctas), su puntaje para el segundo grupo será de 2 puntos (75% de respuestas correctas), y su puntaje final será de 3 puntos.