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

Criptografía y Seguridad Computacional - IIC3253
Tarea 3
Plazo de entrega: 20 de julio

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 tercera tarea usted deberá utilizar el mismo repositorio que usó para las dos tareas anteriores. Al entregar esta tercera tarea, la estructura de archivos de su repositorio se deberá ver exactamente de la siguiente forma:

```
Repositorio
├── Tarea 1
│   └── ...
├── Tarea 2
│   └── ...
├── Tarea 3
│   ├── 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 mencionaron ciertas consecuencias negativas que conlleva el autentificar y autorizar a usuarios en la Web en base a sus nombres de usuario (o correos electrónicos) y contraseñas.

   (a) Explique en detalle cuáles son estas consecuencias, y describa casos hipotéticos que evidencien que son negativas.

   (b) Diseñe una alternativa para autentificar y autorizar usuarios en la web que **no** traiga consigo las consecuencias negativas mencionadas. Explique cómo funcionaría su sistema de autentificación/autorización, y qué problemas prácticos podría traer consigo. Finalmente, explique qué medidas tomaría para prevenir dichos problemas.

2. En esta pregunta usted deberá implementar en Python el protocolo de firma digital de Schnorr. Suponga como dados un número primo p y un par de elementos $g, q \in \{1, \ldots, p-1\}$ tal que q es el orden del sub-grupo generado por g en el grupo $\mathbb{Z}_p^* = (\{1, \ldots, p-1\}, \cdot)$. Vale decir, se deben cumplir las siguientes condiciones:

   $$\begin{aligned} g^i &\not\equiv g^j \mod p \qquad \text{para cada } i, j \in \{1, \ldots, q\} \text{ con } i \neq j, \\ g^q &\equiv 1 \mod p. \end{aligned}$$

   Ademas, suponga que el espacio M de posibles mensajes corresponde con los strings de Python (tipo `str`), y que $h : M \to H$ es una función de hash con H un subconjunto finito de $\mathbb{N}$. En el protocolo criptográfico ElGamal, la clave privada de un usuario A es un número $x \in \{1, \ldots, q-1\}$ generado al azar con distribución uniforme, y la clave pública de A es $y = g^x \bmod p$. En el protocolo de firma digital de Schnorr, A utilizada su clave privada x

para firmar un mensaje $m \in M$ de la siguiente forma. A genera $k \in \{1, \ldots, q-1\}$ al azar con distribución uniforme, y luego calcula:

$$\begin{aligned} r &= g^k \bmod p, \\ e &= h(r\|m), \\ s &= k - x \cdot e, \end{aligned}$$

donde $r\|m$ es la concatenación del número r visto como un string con m (por ejemplo, se tiene que `3341`$\|$`hola` = `3341hola`). Entonces la firma de Schnorr de m es definida como el par (e, s).

Un usuario B puede verificar que una firma (e, s) para un mensaje m fue generada por A de la siguiente forma. Utilizando la clave pública y de A, el usuario B calcula $r' = (g^s \cdot y^e) \bmod p$, y luego verifica si $h(r'\|m) = e$.

Para implementar el protocolo de firma digital de Schnorr, primero deberá crear un archivo `grupo.txt` para almacenar los valores de p, g y q que definen el grupo a ser utilizado. En particular, estos valores debe ser tomados desde aquí: `https://datatracker.ietf.org/doc/html/rfc5114#section-2.3`. Después 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 permite crear claves públicas y privadas de acuerdo al protocolo criptográfico ElGamal.

```
def generar_clave_ElGamal():
    # Retorna :
    #   Genera una clave privada y una clave publica segun el protocolo
    #   critografico ElGamal, para el grupo almacenado en grupo.txt.
    #   Almacena la clave privada en private_key.txt, y la clave publica
    #   en public_key.txt.
```

- Una función que retorna una firma de un mensaje utilizando el protocolo de Schnorr.

```
def firmar_Schnorr(m: int) -> (int,int):
    # Argumentos :
    #   m: int - mensaje
    # Retorna :
    #   (int,int) - firma de Schnorr (e,s) del mensaje m segun la clave
    #   privada almacenada en private_key.txt, para el grupo almacenado
    #   en grupo.txt
```

- Una función que verifica si un par (e, s) es un firma de un mensaje m de acuerdo al protocolo de Schnorr.

```
def verificar_firma_Schnorr(m: int, firma: (int,int)) -> bool:
    # Argumentos :
    #   m: int - mensaje
    #   firma: (int,int) - firma de Schnorr (e,s) para m
    # Retorna :
    #   bool - retorna True si para el usario con clave publica
    #   almacenada en public_key.txt, el par (e,s) es una firma de
    #   Schnorr correcta para el mensaje m, en el grupo almacenado en
    #   grupo.txt. En caso contrario retorna False.
```

- Una función que implementa MD5 interpretando la salida como un número natural.

```python
def md5(m: str) -> int:
    # Argumentos :
    #   m: str - mensaje
    # Retorna :
    #   int - valor de la funcion de hash MD5 aplicada sobre m
```