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

Criptografía y Seguridad Computacional - IIC3253
Tarea 1
Plazo de entrega: 28 de abril

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.

Configuración inicial. Para entregar las tareas usted deberá crear un repositorio en GitHub y darle acceso como colaboradores a los ayudantes del curso, cuyos usuarios son `@johntrombon` y `@jnhasard`. También deberá responder este formulario, en el que se le pedirá una **llave simétrica** que será utilizada para encriptar sus notas y correcciones usando el esquema AES. En el repositorio del curso se publicará además el valor de hash de su llave utilizando SHA256. Si usted pierde dicha llave podrá recuperarla incurriendo en una penalización de dos décimas en el promedio de sus tareas. Si otra persona descubre su llave antes de que se publiquen las notas de la tercera tarea, el promedio de tareas de dicha persona aumentará en cinco décimas, mientras que el suyo disminuirá en cinco décimas. Recomendación: use un administrador de contraseñas.

Entrega. Al entregar esta tarea, su repositorio se deberá ver exactamente de la siguiente forma:

```
Repositorio
├── Tarea 1
│   ├── Pregunta 1
│   │   ├── requirements.txt
│   │   └── pregunta1.ipynb
│   ├── Pregunta 2
│   │   ├── requirements.txt
│   │   └── pregunta2.ipynb
│   ├── Pregunta 3
│   │   └── pregunta3.pdf
│   └── Pregunta 4
│       └── pregunta4.pdf
├── .gitignore
├── README.md
└── .git
```

Deberá considerar lo siguiente:

- El archivo `requirements.txt` dentro de la carpeta de una pregunta deberá especificar todas las librerías que se necesitan instalar para ejecutar el código de su respuesta a dicha pregunta. Este archivo debe seguir la especificación de Pip, es decir se debe poder ejecutar el comando `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. Si su respuesta no requiere librerías adicionales, este archivo debe estar vacío (pero debe estar en su repositorio).

- 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 y clases que se piden en cada pregunta. Por ejemplo, para la Pregunta 1, luego de ejecutar este comando, se debe poder importar desde otro archivo python (ubicado en el mismo directorio) la función `break_rp` simplemente con `from pregunta1 import break_rp`.

- Para cada problema cuya solución se deba entregar como un documento (en este caso las preguntas 2 y 4), 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. Dado un alfabeto $\Sigma = \{\sigma_0, \ldots, \sigma_{N-1}\}$ definiremos el esquema criptográfico $\text{RP}^{\Sigma,\ell}$ (Repeated Pad sobre Σ con llaves de largo ℓ), como (Gen, Enc, Dec) donde:

   - Gen es la distribución uniforme sobre Σ^ℓ.
   - Dado $m = \sigma_{m_0}, \ldots, \sigma_{m_{n-1}} \in \Sigma^*$ y $k = \sigma_{k_0}, \ldots, \sigma_{k_{\ell-1}} \in \Sigma^\ell$, el texto cifrado $c = Enc(k, m)$ se define como $c = \sigma_{c_0}, \ldots, \sigma_{c_{n-1}}$ donde $c_i = (\, m_i + k_{(i \bmod \ell)}\,) \mod N$.
   - Dado $c = \sigma_{c_0}, \ldots, \sigma_{c_{n-1}} \in \Sigma^*$ y $k = \sigma_{k_0}, \ldots, \sigma_{k_{\ell-1}} \in \Sigma^\ell$, el texto plano $m = Dec(k, c)$ se define como $m = \sigma_{m_0}, \ldots, \sigma_{m_{n-1}}$ donde $m_i = (\, c_i - k_{(i \bmod \ell)}\,) \mod N$.

   En clases se mostró una forma de decriptar mensajes suficientemente largos encriptados con $\text{RP}^{\Sigma,\ell}$, suponiendo que los mensajes originales estaban escritos en inglés y ℓ era un valor conocido. Para esto se utilizó la frecuencia de caracteres del inglés, además de una noción que definía intuitivamente cuánto dista un string de seguir dicha frecuencia. En esta pregunta se deberá generalizar lo hecho en clases para intentar decriptar $\text{RP}^{\Sigma,\ell}$ suponiendo lo siguiente:

   - El largo de la llave es desconocido.
   - El mensaje original está en un idioma arbitrario, para el cual la frecuencia es conocida.
   - La noción de cuánto dista un string de seguir una frecuencia de caracteres es arbitraria.

   En concreto, deberá escribir una función que reciba (1) un texto cifrado, (2) una frecuencia de caracteres, y (3) una función que indica cuánto dista un string de seguir una frecuencia de caracteres, y retorne la llave que se utilizó para encriptar el texto cifrado.

   La entrega de esta pregunta deberá seguir las instrucciones indicadas más arriba, entregando un Jupyter Notebook que defina una función como la siguiente:

```python
def break_rp(
    ciphertext: str,
    frequencies: {str: float},
    distance: (str, {str: float}) -> float,
) -> str:
    """
    Arguments:
        ciphertext: An abritrary string representing the
                    encrypted version of a plaintext.
        frequencies: A dictionary representing a character
                     frequency over the alphabet.
        distance: A function indicating how distant is a string
                  from following a character frequency
    Returns:
        key: A guess of the key used to encrypt the ciphertext, assuming
             that the plaintext message was written in a language in which
             letters distribute according to frequencies.
    """
```

   Para probar su respuesta se recomienda utilizar la distancia que definimos en clases, dada por la siguiente función:

```python
def abs_distance(string: str, frequencies: {str: float}) -> float:
  """
  Arguments:
    string: An abritrary string
    frequencies: A dictionary representing a character frequency
  Returns:
    distance: How distant is the string from the character frequency
  """
  return sum([
    abs(frequencies[c] - string.count(c) / len(string))
    for c in frequencies
  ])
```

Puede suponer que el alfabeto es el conjunto de llaves del diccionario `frequencies`, y que el largo de la llave es a lo más el largo de `ciphertext` dividido en 50.

Restricción. Si se utiliza la función `abs_distance` definida arriba, un texto cifrado con 1000 caracteres y un alfabeto de 30 caracteres, su función no puede demorar más de 10 segundos en su propio computador.

2. Considere un esquema criptográfico (Gen, Enc, Dec) definido sobre los espacios $\mathcal{M} = \mathcal{K} = \mathcal{C} = \{0,1\}^n$. Suponga además que Gen no permite claves cuyo primer bit sea 0, y que el resto de las claves son elegidas con distribución uniforme. Demuestre que este esquema no es una pseudo-random permutation con una ronda, si $\frac{3}{4}$ es considerada como una probabilidad significativamente mayor a $\frac{1}{2}$.

3. Un árbol de Merkle es una estructura de datos que utiliza una función de hash criptográfica h para representar un conjunto de strings $S = \{s_1, \ldots, s_m\}$. La estructura es un árbol binario en el que cada nodo es un string definido recursivamente: Las hojas son $h(s_1), \ldots, h(s_m)$ y cada nodo interior n se define como $h(n_1 \| n_2)$ donde n_1 y n_2 son los hijos de n y $\|$ representa la concatenación de strings. Cuando la cantidad de nodos en un nivel es impar se duplica el último nodo de dicho nivel, lo que implica que todos los nodos interiores tienen exactamente dos hijos. La Figura 1 muestra gráficamente la construcción de un árbol de Merkle:

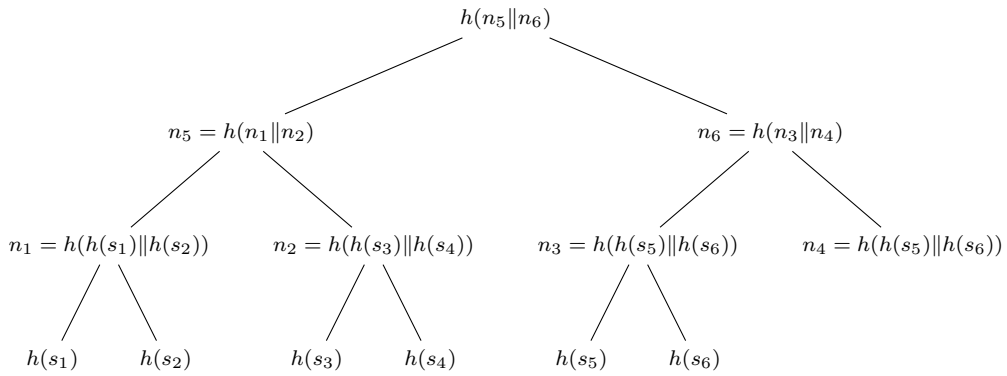

Figure 1: Árbol de Merkle para el conjunto $S = \{s_1, \ldots, s_6\}$.

La principal propiedad de un árbol de Merkle es que si una persona posee un hash que representa la raíz del árbol, la podemos convencer rápidamente de que un elemento es una de las hojas del árbol. Para esto, basta con compartir con esa persona dicho elemento, su hermano y los hermanos de todos sus ancestros, indicando para cada uno si es hermano derecho (d) o izquierdo (i). Por ejemplo, supongamos que alguien tiene la raíz del árbol que se muestra en la Figura 1. Para convencer a dicha persona de que s_5 es parte del conjunto S basta con enviarle el valor s_5 junto con $(h(s_6), d)$, (n_4, d) y (n_5, i). Teniendo estos valores, esta persona puede:

- Computar $n_3 = h(h(s_5)\|h(s_6))$.
- Computar $n_6 = h(n_3\|n_4)$.
- Computar $h(n_5\|n_6)$ y verificar que el resultado sea igual a la raíz del árbol.

En esta pregunta usted deberá programar una clase que permita crear árboles de Merkle para conjuntos arbitrarios de strings, usando una función de hash arbitraria. Deberá seguir las instrucciones indicadas más arriba, entregando un Jupyter Notebook que define una clase como la siguiente:

```
class MerkleTree:

  def __init__(self, strings: [str], hash_func: (str) -> str) -> MerkleTree:
  """
  Arguments:
    strings: The set of strings S
  """

  def get_root(self) -> string:
  """
  Returns:
    root: Root of the Merkle Tree
  """

  def get_proof_for(self, item: str) -> None || [(str, "d"|"i")]:
  """
  Returns:
    result: None if the item is not part of the leafs of the tree
            A list with the necessary info to prove that the
            item is part of the leafs of the tree
  """
```

Además de esta clase, usted deberá programar una función que reciba una *prueba* de que un elemento es parte de un árbol de Merkle, y retorne `True` si la prueba es correcta y `False` de lo contrario. Esta función también deberá estar escrita en su Jupyter Notebook y deberá tener la siguiente firma:

```
def verify(root: string, item: str, proof: [(str, "d"|"i")]) -> bool:
  """
  Arguments:
    root: The root of a merkle tree
    item: An abritrary string
```

```
    proof: An alleged proof that item is part of a Merkle
           tree with root root
  Returns:
    correct: whether the proof is correct or not
  """
```

4. Considere el juego *Hash-Col*(n) mostrado en clases para definir la noción resistencia a colisiones. Utilizando este tipo de juegos, defina la noción de resistencia a preimagen para una función de hash (Gen, h). Además, demuestre que si (Gen, h) es resistente a colisiones, entonces (Gen, h) es resistente a preimagen.