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

Criptografía y Seguridad Computacional - IIC3253
Rúbrica Tarea 1

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:

```
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.

Corrección. Para corregir esta pregunta se evaluará su función `break_rp` con doce ejemplos distintos, los cuales se pueden encontrar en el repositorio del curso bajo `/Tareas/Tarea 1/Corrección/archivos/p1.json`. Cada ejemplo es un diccionario con los siguientes campos:

- `distance`: La función de distancia que se utiliza. Puede ser `abs`, `squared` o `quadratic`. La definición de estas funciones se explica más abajo.
- `frequencies`: El diccionario de frecuencias utilizado.
- `key`: La llave original utilizada para encriptar.
- `found_key`: La llave encontrada por el algoritmo base, definido más abajo.
- `ciphertext`: El texto cifrado.
- `found_distance`: La distancia entre el texto decriptado con `found_key` y `frequencies` de acuerdo a la función `distance`.

La función de distancia `abs` es la función `abs_distance` definida arriba. Las funciones `quadratic` y `squared` son equivalentes a `abs` pero con la diferencia de que cada valor absoluto en la suma es remplazado por su raíz o su valor al cuadrado, respectivamente.

El algoritmo base consiste en una generalización del algoritmo visto en clases para buscar llaves de distintos largos, que se encuentra en `/Tareas/Tarea 1/Solución/pregunta1.ipynb`

La asignación de puntajes será la siguiente:

- [1 punto] La función `break_rp` entregada recibe parámetros y retorna objetos del tipo esperado, pero para ninguno de los ejemplos genera una llave que, al usarla para decriptar, implique una distancia de frecuencias menor o igual a `found_distance`.
- [3 punto] La función entregada genera al menos una llave que, al usarla para decriptar, implica una distancia de frecuencias menor o igual a `found_distance`.
- [4 punto] La función entregada genera al menos seis llaves que, al usarlas para decriptar, implican una distancia de frecuencias menor o igual a `found_distance`.
- [5 punto] La función entregada genera al menos diez llaves que, al usarlas para decriptar, implican una distancia de frecuencias menor o igual a `found_distance`.
- [6 punto] La función entregada genera sólamente llaves que, al usarlas para decriptar, implican una distancia de frecuencias menor o igual a `found_distance`.

Se generarán los siguientes descuentos:

- [5 décimas] El programa no puede ser transformado a una librería de forma automática.
- [5 décimas] La función entregada genera excepciones al ejecutarse con los parámetros correspondientes a la corrección, pero el ayudante es capaz de corregirlo en tres minutos o menos.
- [1 punto] La función entregada genera excepciones al ejecutarse con los parámetros correspondientes a la corrección, pero el ayudante es capaz de corregirlo en seis minutos o menos.
- [2 puntos] La función entregada genera excepciones al ejecutarse con los parámetros correspondientes a la corrección y el ayudante no es capaz de corregirlo en diez minutos o menos, pero en la recorrección el estudiante entrega una respuesta ecenciamente equivalente a la que entregó antes que sí funciona.

Cabe mencionar que todos los parámetros que se usarán serán de los tipos esperados y presentarán consistencia en el contenido. En particular, todos los caracteres que aparecen en los textos cifrados están también en los diccionarios de frecuencia.

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 $\frac{1}{2}$.

Corrección. Esta pregunta se corrige considerando que se debe diseñar una estrategia para el adversario que le permita ganar en una ronda con una probabilidad mayor o igual a $\frac{3}{4}$.

- [3 puntos] Solo se entrega una estrategia que permite al adversario ganar con una probabilidad mayor o igual a $\frac{3}{4}$.
- [4.5 puntos] Se entrega una estrategia que permite al adversario ganar con una probabilidad mayor o igual a $\frac{3}{4}$, y se formula de manera correcta la probabilidad de que el adversario gane.
- [6 puntos] Se entrega una estrategia que permite al adversario ganar con una probabilidad mayor o igual a $\frac{3}{4}$, se formula de manera correcta la probabilidad de que el adversario gane, y se muestra que esta probabilidad es mayor o igual a $\frac{3}{4}$.

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:

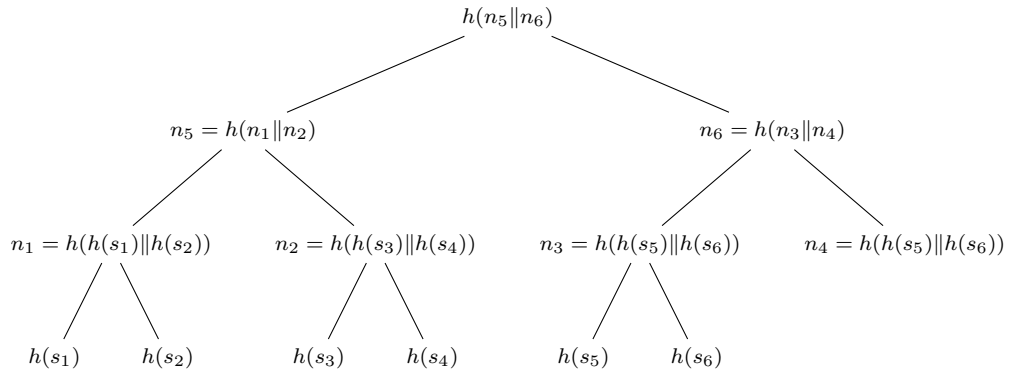

Figure 1: Árbol de Merkle para el conjunto $S = \{s_1, \ldots, s_6\}$.

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
  """
```

Corrección. TBA

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.

Corrección. Esta pregunta se corrige considerando la definición de función de hash (Gen, h) dada en clases; en particular, sabemos que si $Gen(1^n) = s$, entonces $h^s : \{0,1\}^* \to \{0,1\}^{\ell(n)}$ donde $\ell(n)$ es un polinomio fijo. Definimos entonces un juego *Hash-Pre-Img*(n) dado por los siguientes pasos:

(a) El verificador genera $s = Gen(1^n)$ y un hash $x \in \{0,1\}^{\ell(n)}$

(b) El adversario elige $m \in \{0,1\}^*$ o $m = \perp$

(c) El adversario gana el juego si alguna de las siguientes condiciones se cumple:

- $m \in \{0,1\}^*$ y $h^s(m) = x$
- $m = \perp$ y no existe $m' \in \{0,1\}^*$ tal que $h^s(m') = x$

En caso contrario, el adversario pierde.

Además, decimos que (Gen, h) es resistente a preimagen si para todo adversario que funciona como un algoritmo aleatorizado de tiempo polinomial, existe una función despreciable $f(n)$ tal que:

$$\Pr(\text{Adversario gane } \textit{Hash-Pre-Img}(n)) \quad \leq \quad f(n)$$

Considerando estos conceptos, la pregunta se corrige de la siguiente forma:

- [1.5 puntos] Solo se entrega una definición de resistencia a preimagen que es significativamente mas restringida que la anterior.
- [3 puntos] Solo se entrega una definición de resistencia a preimagen que es cercana a la anterior.
- [4.5 puntos] Se entrega una definición de resistencia a preimagen que es cercana a la anterior, y se da una idea de cómo se puede demostrar que resistencia a colisiones implica resistencia a preimagen.
- [6 puntos] Se entrega una definición de resistencia a preimagen que es cercana a la anterior, y se demuestra formalmente que resistencia a colisiones implica resistencia a preimagen.