En esta pregunta usted va a analizar la complejidad de este protocolo, para lo cual va a considerar las operaciones entre bits (suma, resta, comparación, etc.) como las operaciones básicas en los algoritmos, las cuales tienen costo 1. Por ejemplo, verificar si $u = v$ para dos palabras $u, v \in \{0,1\}^n$ toma tiempo n ya que se deben realizar n operaciones de comparación entre bits. En el análisis a realizar a continuación debe suponer que $h^n(u\|v)$ se calcula en tiempo $O(n)$, lo cual es cierto para las funciones de hash usuales.

(a) La llamada **EstablecerClave**(1^n) falla tanto si no se tiene un par (i,j) tal que $a_i = b_j$ como si $x_k \neq y_\ell$ (las claves secretas establecidas por A y B son distintas). Demuestre que existe una función despreciable $f(n)$ tal que:

$$\Pr(\textbf{EstablecerClave}(1^n) \text{ falle}) \quad \leq \quad f(n).$$

(b) Suponga que **EstablecerClave** no falla. Demuestre que A y B establecen una clave compartida en tiempo $O(n^2 \cdot \log^2 n)$.

(c) Suponga que **EstablecerClave** no falla, y que un atacante trata de descubrir la clave compartida entre A y B. Suponga que el atacante es exitoso en el sentido de que logra construir un algoritmo (no aleatorizado) $\mathcal{A}$ que dado $s, u_1, u_2, v \in \{0,1\}^n$ tal que $|\{u \in \{0,1\}^n \mid u_1 \leq u \leq u_2\}| = n^2$, genera $u \in \{0,1\}^n$ que satisface $h(s\|u) = v$ y $u_1 \leq u \leq u_2$ siempre que dicho u exista, y retorna $\perp$ si dicho u no existe. Para la construcción anterior $\mathcal{A}$ realiza $o(n^3)$ operaciones, donde $\leq$ es el orden lexicográfico sobre $\{0,1\}^n$ definido por $0 < 1$. Demuestre que esto lleva a una contradicción puesto que implicaría que la familia de funciones $\{h^n\}_{n\in\mathbb{N}}$ no es puzzle friendly.

2. Sea (Gen, h) una función de hash tal que $Gen(1^n) = n$ y $h^n : \{0,1\}^* \to \{0,1\}^n$. El siguiente juego es utilizado para definir la propiedad de que (Gen, h) es resistente a modificaciones en la pre-imagen.

PreImageModification(1^n)

- El atacante define un algoritmo de tiempo polinomial $\mathcal{A} : \{0,1\}^* \to \{0,1\}^*$ tal que para todo $x \in \{0,1\}^*$: x es un prefijo de $\mathcal{A}(x)$ y el largo de $\mathcal{A}(x)$ es mayor al largo de x.
- El atacante envía $\mathcal{A}$ al verificador.
- El verificador selecciona $x \in \{0,1\}^n$ y envía $h^n(x)$ al adversario.
- El verificador selecciona al azar $b \in \{0,1\}$.
  - Si $b = 0$, el verificador computa $y = h^n(\mathcal{A}(x))$.
  - Si $b = 1$, el verificador elige al azar $y \in \{0,1\}^n$.
- El verificador envía y al adversario.
- El adversario elige $b' \in \{0,1\}$, y gana si $b = b'$.

Decimos que una función de hash es resistente a modificaciones de pre-imagen si es que no existe un adversario que funcione en tiempo polinomial (en n) y que gane el juego **PreImageModification**(1^n) con una probabilidad no despreciable.[2]

[2] Al igual que para los juegos vistos en clases, esto significa que el adversario no puede ganar **PreImageModification**(1^n) con una probabilidad $\frac{1}{2} + f(n)$, donde $f(n)$ es una función no despreciable.

(a) Demuestre que las funciones de hash basadas en la construcción de Merkle-Damgård vista en clases no son seguras frente a modificaciones de pre-imagen. En particular, para esta construcción considere la función de padding vista en clases, la cual es definida de la siguiente forma. Dado un mensaje m, sea $\ell = |m| \bmod n$, y sea $m_1 \in \{0,1\}^n$ la representación como string binario del número $|m| \bmod 2^n$. Si $\ell = 0$, entonces $Pad(m) = m \| m_1$. Y si $\ell > 0$, entonces $Pad(m) = m \| 10^{n-\ell-1} \| m_1$.

(b) Programe en Python un adversario que gane este juego para la función `SHA256`. Específicamente, deberá entregar un archivo `pregunta2_b.py` que contenga dos funciones:

- `alg(bytes) -> bytes`. Esta función representa el algoritmo $\mathcal{A}$ que utilizará el adversario para ganar el juego definido más arriba para el caso de `SHA256`.
- `adv(z: bytes, y: bytes) -> bool`. Esta función representa a su adversario que, habiendo recibido $z = h(x)$ e y (teniendo $x, y, z \in \{0,1\}^{256}$), deberá retornar verdadero si y sólo si $y = \texttt{SHA256}(\texttt{alg}(x))$.