



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

Criptografía y Seguridad Computacional - IIC3253

Tarea 2

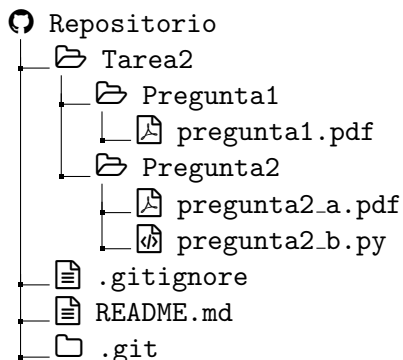
Plazo de entrega: miércoles 14 de mayo

Instrucciones

Cualquier duda sobre la tarea se deberá hacer en los *issues* del repositorio del curso. Los issues son el canal de comunicación oficial para todas las tareas.

Configuración inicial. Para esta tarea utilizaremos *github classroom*. Para acceder a su repositorio privado debe ingresar al siguiente link, seleccionar su nombre y aceptar la invitación. El repositorio se creará automáticamente una vez que haga esto y lo podrá encontrar junto a los repositorios del curso. Para la corrección se utilizará Python 3.11.

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



Para cada problema cuya solución se deba entregar como un documento (en este caso la pregunta 1), deberá entregar un archivo `.pdf` que, o bien fue construido utilizando $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$, 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. Sea $\{h^n\}_{n \in \mathbb{N}}$ una familia de funciones de compresión resistente a colisiones tal que $h^n : \{0, 1\}^{2n} \rightarrow \{0, 1\}^n$. Supondremos también que esta familia es *puzzle friendly*, lo que significa

que no existe un algoritmo eficiente que, dados $u, v \in \{0, 1\}^n$, construye $x \in \{0, 1\}^n$ tal que $h^n(u\|x) = v$. Formalmente, la familia $\{h^n\}_{n \in \mathbb{N}}$ se dice *puzzle friendly* si para cada algoritmo aleatorizado $\mathcal{A} : \{0, 1\}^{2n} \rightarrow \{0, 1\}^n$ que funciona en tiempo $o(2^n)$, se tiene que la probabilidad de que $h(u, \mathcal{A}(u, v)) = v$ es despreciable.¹

A partir de la familia $\{h^n\}_{n \in \mathbb{N}}$, definimos el protocolo **EstablecerClave**(1^n) que permite a dos usuarios A y B establecer una clave de n bits en un canal público, sin la necesidad de juntarse físicamente.

EstablecerClave(1^n)

- (1) Sea P el conjunto de las primeras n^2 palabras en $\{0, 1\}^n$, ordenadas por orden lexicográfico (definido por $0 < 1$), y sea $m = n \cdot \lceil \log n \rceil$. Por ejemplo, si $n = 5$, entonces $P = \{00000, 00001, 00010, \dots, 10110, 10111, 11000\}$ y $m = 15$.
 - (1.1) A escoge m palabras distintas u_1, \dots, u_m desde el conjunto P , calcula $a_i = h(0^n\|u_i)$ para cada $i \in \{1, \dots, m\}$, y envía $(1, a_1), \dots, (m, a_m)$ a B .
 - (1.2) B escoge m palabras distintas v_1, \dots, v_m desde el conjunto P , calcula $b_i = h(0^n\|v_i)$ para cada $i \in \{1, \dots, m\}$, y envía $(1, b_1), \dots, (m, b_m)$ a A .
- (2) Sea $I = \{(i, j) \mid a_i = b_j\}$. Si $I = \emptyset$, entonces el protocolo falla. En otro caso, sea (k, ℓ) el menor elemento en I en el orden lexicográfico sobre $\{1, \dots, m\}^2$ definido por $1 < 2 < \dots < m$.
 - (2.1) A establece como clave u_k .
 - (2.2) B establece como clave v_ℓ (que debería ser igual a u_k).

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 $u_k \neq v_\ell$ (las claves secretas establecidas por A y B son distintas). Demuestre que existe una función despreciable $f(n)$ tal que:

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

- (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 sea \mathcal{A} un algoritmo aleatorizado que usa un atacante para descubrir la clave compartida entre A y B . Demuestre que si \mathcal{A} funciona en tiempo $o(n^3)$, entonces la probabilidad de que la repuesta entregada por \mathcal{A} sea correcta es una función despreciable (como función del largo n de la clave).

¹Recuerde que una función $f(n)$ es $o(g(n))$ si se cumple que $(\forall c \in \mathbb{R}, c > 0)(\exists n_0 \in \mathbb{N})(\forall n \in \mathbb{N}, n \geq n_0)(f(n) \leq c \cdot g(n))$.

2. Sea (Gen, h) una función de hash tal que $Gen(1^n) = n$ y $h^n : \{0, 1\}^* \rightarrow \{0, 1\}^n$. 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 $\mathcal{A} : \{0, 1\}^* \rightarrow \{0, 1\}^*$ 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.

- (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 $y = \text{SHA256}(\text{alg}(x))$.