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

Criptografía y Seguridad Computacional - IIC3253
Tarea 2
Plazo de entrega: martes 20 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 automaticamente 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:

```
Repositorio
├── Tarea2
│   ├── Pregunta1
│   │   └── pregunta1.pdf
│   └── Pregunta2
│       ├── pregunta2_a.pdf
│       └── pregunta2_b.py
├── .gitignore
├── README.md
└── .git
```

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 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. Sea $\{h^n\}_{n\in\mathbb{N}}$ una familia de funciones de compresión resistente a colisiones tal que $h^n : \{0,1\}^{2n} \to \{0,1\}^n$. Supondremos también que esta familia es *puzzle friendly*, lo que significa que no existe un algoritmo eficiente que es capaz de encontrar una palabra x que resuelve un puzzle $h^n(u\|x) = v$. Formalmente, un puzzle es un par $(u, v) \in \{0,1\}^{2n}$, y una solución para el puzzle es una palabra $x \in \{0,1\}^n$ tal que $h(u\|x) = v$. Si existe tal x, se dice que el puzzle (u, v) tiene solución. Con esta notación, la familia $\{h^n\}_{n\in\mathbb{N}}$ se dice *puzzle friendly* si para cada algoritmo aleatorizado $\mathcal{A}$ tal que:

   - con entrada $(u, v) \in \{0,1\}^{2n}$, el algoritmo $\mathcal{A}$ retorna una palabra $\mathcal{A}(u, v) \in \{0,1\}^n$ o el símbolo $\perp$,
   - $\mathcal{A}$ realiza $o(n \cdot 2^n)$ operaciones[1] para cada entrada $u, v \in \{0,1\}^n$,

   se tiene que la siguiente función de n es despreciable:

$$\max_{v\in\{0,1\}^n} \Pr_{u\sim\mathbb{U}(\{0,1\}^n)} \big[\mathcal{A} \text{ soluciona el puzzle } (u, v)\big],$$

   donde $u \sim \mathbb{U}(\{0,1\}^n)$ indica que u es escogido al azar con distribución uniforme desde el conjunto $\{0,1\}^n$, y $\mathcal{A}$ soluciona el puzzle (u, v) si $\mathcal{A}(u, v) \in \{0,1\}^n$ y $h(u\|\mathcal{A}(u, v)) = v$ en caso de que el puzzle (u, v) tenga solución, y $\mathcal{A}(u, v) = \perp$ en caso de que el puzzle (u, v) no tenga solución.

   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) A escoge con distribución uniforme $s \in \{0,1\}^n$, y se lo envía a B.

   (2) 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 $P = \{00000, 00001, 00010, \ldots, 10110, 10111, 11000\}$ y $m = 15$.

      (2.1) A escoge m palabras distintas $x_1, \ldots, x_m$ desde el conjunto P, calcula $a_i = h(s\|x_i)$ para cada $i \in \{1, \ldots, m\}$, y envía $(1, a_1), \ldots, (m, a_m)$ a B.

      (2.2) B escoge m palabras distintas $y_1, \ldots, y_m$ desde el conjunto P, calcula $b_i = h(s\|y_i)$ para cada $i \in \{1, \ldots, m\}$, y envía $(1, b_1), \ldots, (m, b_m)$ a A.

   (3) 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 $\{1, \ldots, m\}^2$ definido por $1 < 2 < \cdots < m$.

      (3.1) A establece como clave x_k.

      (3.2) B establece como clave y_ℓ (que debería ser igual a x_k).

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

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