Ayudantía 6

IIC3253 - Criptografía y Seguridad Computacional

Christian Klempau

1 HMAC

1.1 JWT

JSON Web Token (JWT) es un estándar del Internet, utilizado para compartir información y autenticar usuarios, de forma segura. Se construye de la siguiente manera:

Header:

```
{
    "alg": "HS256",
    "typ": "JWT"
}
```

Body:

```
{
  "lat": " -19.949156",
  "lng": "-69.633842",
  "name": "Chris Klempau",
  "url_base64": "aHR0cHM6Ly93d3cueW91dHViZS5jb2
                 0vd2F0Y2g/dj1kUXc0dzlXZ1hjUQ=="
}
```

Signature:

$$Message = base64(Header) + "." + base64(Body)$$

$$Signature = \text{HMACSHA256}(Message, Secret)$$

Y el JWT final es:

$$Header_{b64}.Body_{b64}.Signature_{b64}$$

Por ejemplo, algo del estilo:

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4
gRG9lIiwiaXNTb2NpYWwiOnRydWV9.
4pcPyMD09olPSyXnrXCjTwXyr4BsezdI1AVTmud2fU4
```

¿Qué hace el receptor? Verifica que la firma (*signature*) del JWT sea efectivamente: $Signature_{JWT} = \text{HMACSHA256}(Message_{JWT}, Secret)$. En ese caso, puede asumir que el mensaje fue construido por alguien con acceso a la llave $Secret$ y se considera válido.

En el caso de un atacante, al no tener acceso a la llave *Secret*, no puede construir un JWT válido.

Puedes probar JWTs en: `https://jwt.io/`

2 Teoría de números

2.1 Máximo común divisor

2.

1. Defina un algoritmo para calcular el máximo común divisor de dos números enteros positivos a y b, dado que:

$$MCD(a,b) = \begin{cases} a & \text{si } b = 0 \\ MCD(b, a \bmod b) & \text{si } b > 0 \end{cases}$$

2. ¿Cuál es la complejidad temporal (peor caso) del algoritmo?

1.

```
# Recursivo
# input:    dos números a y b
# output:   máximo común divisor (a, b)

GCD(a, b):
    if b == 0 then
        return a
    else
        return GCD(b, a mod b)
    end
```

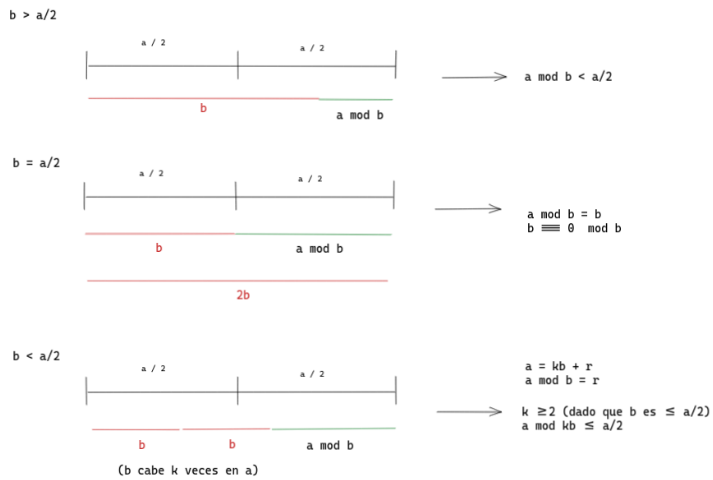
```
# Iterativo
# input:    dos números a y b
# output:   máximo común divisor (a, b)

GCD(a, b):
    while b != 0 do
        r <- a mod b
        a <- b
        b <- r
    end
    return a
```

2.

Nos basamos en que:

$$a \bmod b \leq \frac{a}{2}$$

Lo podemos ver gráficamente, por casos:



Dado que $a \bmod b \leq \frac{a}{2}$, podemos decir que en cada paso, el algoritmo reduce el input a la mitad o menos.

Por lo tanto, el algoritmo es $\in O(\log a)$, con $a \geq b$.

¿Por qué? Recuerde la demostración de binary search: el input se reduce a la mitad (en nuestro caso, reducimos a la mitad o menor) en cada iteración, por lo que la complejidad es $\in O(\log n)$

2.2 Algoritmo extendido de Euclides

Dado el algoritmo extendido de Euclides:

```
# input:    dos números a y b
# output 1:       ?
# output 2 y 3:   ?

ExtendedGCD(a, b):
    r_old, r := a, b
    s_old, s := 1, 0
    t_old, t := 0, 1

    while r != 0 do
        q  := old_r // r

        s_old, s := s, s_old - q * s
        t_old, t := t, t_old - q * t
        r_old, r := r, s * a + t * b
    end

    return r_old, s_old, t_old
```

Nota: recuerde la identidad de Bézout:

$$MCD(a, b) = s \cdot a + t \cdot b$$

1. ¿Qué significan los valores retornados?

   Output 1: $r\_old = GCD(a, b)$

   Output 2: $s\_old =$ coeficiente de a en la identidad de Bézout

   Output 3: $t\_old =$ coeficiente de b en la identidad de Bézout

2. ¿Cuál es el resultado de ejecutar el algoritmo, con $a = 240$ y $b = 46$?

```
q_i+1 = r_i-1 // r_i
s_i+1 = s_i-1 - q_i * s_i
t_i+1 = t_i-1 - q_i * t_i
r_i+1 = s_i * a + t_i * b
```

index i	quotient q_{i-1}	Remainder r_i	s_i	t_i
0		240	1	0
1		46	0	1
2	240 ÷ 46 = 5	240 - 5 × 46 = 10	1 - 5 × 0 = 1	0 - 5 × 1 = -5
3	46 ÷ 10 = 4	46 - 4 × 10 = 6	0 - 4 × 1 = -4	1 - 4 × -5 = 21
4	10 ÷ 6 = 1	10 - 1 × 6 = 4	1 - 1 × -4 = 5	-5 - 1 × 21 = -26
5	6 ÷ 4 = 1	6 - 1 × 4 = 2	-4 - 1 × 5 = -9	21 - 1 × -26 = 47
6	4 ÷ 2 = 2	4 - 2 × 2 = 0	5 - 2 × -9 = 23	-26 - 2 × 47 = -120

```
r == 0 --> END:

r_old = 2
s_old = -9
t_old = 47
```

2.3 Invertibilidad dado GCD

Demuestre el siguiente teorema:

Un número a es invertible en módulo n si y solo si $GCD(a, n) = 1$.

Demostración

1. $GCD(a, n) = 1 \rightarrow a$ es invertible en módulo n.

   Por identidad de Bézout, existen $s, t \in \mathbb{Z}$ tales que:

$$GCD(a, n) = s \cdot a + t \cdot n = 1$$

Aplicamos $\bmod n$ a ambos lados:

$$s \cdot a + t \cdot n \equiv 1 \bmod n$$

$$s \cdot a \equiv 1 \bmod n$$

Por lo tanto, existe un s que es el inverso modular de a, por definición de inverso modular.

2. a es invertible en módulo $n \to GCD(a, n) = 1$.
Por definición de inverso modular, existe un b tal que:

$$a \cdot b \equiv 1 \bmod n$$

Restamos 1 a ambos lados:

$$a \cdot b - 1 \equiv 0 \bmod n$$

Y 0 en mundo módulo n es equivalente a αn:

$$a \cdot b - 1 \equiv \alpha n \bmod n$$

Reordenamos, y construimos la igualdad:

$$a \cdot b - \alpha n = 1$$

Consideremos el divisor c de a:

$$c | a$$

Como c divide a a, también divide a ab.
Por otro lado, dado que estamos en mundo módulo n:

$$c|(-\alpha n) \to c|0 \text{ , que se cumple siempre.}$$

Como c divide a ambos por separado, divide a su suma.
Por lo tanto, c es un divisor común de a y n.
Luego, por igualdad (parte derecha), c divide a 1.
Entonces, $c = 1 = GCD(a, n)$.

2.4 Demostración alternativa complejidad GCD

Nota:
Teorema de Lamé: La cantidad de pasos que GCD debe realizar es exactamente n, donde $a > b > 0$, tal que:

$$a = F_{n+2}$$

$$b = F_{n+1}$$

Donde F_k es el k-ésimo número de Fibonacci.

Fórmula de Binet:

$$F_k = \frac{\phi^k - \varphi^k}{\sqrt{5}}$$

donde ϕ es el *golden ratio*:

$$\phi = \frac{1+\sqrt{5}}{2}$$

$$\varphi = \frac{1-\sqrt{5}}{2} \approx -0.61$$

Reescribimos b como:

$$b \geq F_{n+1}$$

Notemos que $\varphi^k \approx 0$, para $k >> 0$.
Por lo tanto:

$$b \geq F_{n+1} = F_n + F_{n-1} = \frac{\phi^n - \varphi^n}{\sqrt{5}} + F_{n-1}$$

$$= \frac{\phi^n}{\sqrt{5}} + \underbrace{(F_{n-1} - \frac{\varphi^n}{\sqrt{5}})}_{>0}$$

Entonces:

$$b \geq \frac{\phi^n}{\sqrt{5}}$$

Multiplicando por $\sqrt{5}$ a ambos lados:

$$\sqrt{5}b \geq \phi^n$$

Aplicando logaritmo natural a ambos lados:

$$\ln(\sqrt{5}b) \geq n\ln(\phi)$$

Por lo tanto:

$$n \leq \frac{\ln(\sqrt{5}b)}{\ln(\phi)}$$

$$n \leq \frac{\ln\sqrt{5} + \ln b}{\ln(\phi)} = c_1 + c_2 \cdot \ln b$$

Concluyendo la complejidad temporal $n \in \mathcal{O}(\log b)$

Nota: una complejidad logarítmica bajo el número, es equivalente a una complejidad lineal, bajo la cantidad de dígitos del número.