Laboratorio 5: Funciones Hash

Javiera Fuentes, Nicolás Vergara, Pablo Gajardo, Camilo Muñoz y Vicente Moya

October 18, 2024

1 Introducción

1.1 Contexto

En este informe se presenta el desarrollo y análisis de diversas tareas relacionadas con las funciones hash, dentro del marco del curso de Criptografía y Seguridad en la Información (TEL-252) de la Universidad Técnica Federico Santa María. Las funciones hash son un componente fundamental en la criptografía moderna, ya que permiten transformar grandes cantidades de datos en representaciones fijas, que facilitan tanto la verificación de integridad como la seguridad en sistemas criptográficos. Este laboratorio se enfoca en explorar los conceptos fundamentales de las funciones hash a través de ejercicios prácticos, destacando su importancia en la seguridad de los sistemas de información.

El código relacionado con las funciones hash y los ejercicios prácticos se ha desarrollado utilizando SageMath, en la plataforma CoCalc, para asegurar un entorno adecuado para cálculos simbólicos y numéricos.

1.2 Objetivos

Los objetivos del laboratorio 5 son los siguientes:

- Comprender el funcionamiento básico de las funciones hash y su relación con el cifrado RSA.
- Implementar una función hash simple utilizando números primos y generar colisiones.
- Desarrollar funciones en SageMath que apliquen el proceso de hasheo y demuestren ejemplos prácticos de colisiones.
- Evaluar la resistencia de la función hash implementada frente a ataques de colisiones.
- Analizar los resultados obtenidos y reflexionar sobre la aplicabilidad de las funciones hash en la criptografía moderna.

todo el código desarrollado para éste laboratorio puede encontrarse en el siguiente repositorio de github.

2 Metodología

2.1 Flujo del laboratorio

El flujo del laboratorio se desarrolló de la siguiente manera:

1. **Revisión teórica de la funcion Hash:** Se explicó la teoria y como opera la función Hash, asi como también la forma de romper dicha función.
2. **Demostración de ejemplos:** Se mostraron ejemplos de las salidas esperadas para las funciones pedidas en los ejercicios, asi como tambien algunos ejemplos de colisones.
3. **Desarrollo durante el laboratorio:** Se comenzó a implementar algunos de los incisos del laboratorio. Además, se resolvieron dudas surgidas durante el desarrollo.

2.2 Ejercicios ejecutados

2.2.1 Pregunta 1.a

Escribe una función que tome una longitud de bits n y genere un módulo N de longitud de bits n y g menor que N y relativamente primo a él.

```

def random_prim(bits):

    n_max = 2**(bits) - 1
    n_min = 2**(bits - 1)

    primo = random_prime(n_max, lbound=n_min)

    return primo


def generate_simple_hash_params(n):

    #p y q deben ser de la mitad de bits que N para que N nunca sobrepase la cantidad de bits
    p = random_prim(n//2)
    q = random_prim(n//2)

    N = p * q

    while True:
        g = randint(2, N - 1)
        if gcd(g, N) == 1:
            break

    return N, g

```

3 Ejercicios prácticos

3.1 Pregunta 1.b

Muestra la salida de tu función de la parte (a) para algunas salidas.

```
# /// Llamados a la funcion para algunas entradas ///

N, g = generate_simple_hash_params(16)
print ("con 16 bits, N:", N)
print ("con 16 bits, g:", g)

N, g = generate_simple_hash_params(64)
print ("con 64 bits, N:", N)
print ("con 64 bits, g:", g)

N, g = generate_simple_hash_params(128)
print ("con 128 bits, N:", N)
print ("con 128 bits, g:", g)

```

```
# ////////// Resultados obtenidos //////////

con 16 bits, N: 24883
con 16 bits, g: 23065
con 64 bits, N: 8364603345368287343
con 64 bits, g: 6270148147548365129
con 128 bits, N: 154230738257034191881780325614768136821
con 128 bits, g: 514794557359840571108419047826430930 45
```

3.2 Pregunta 1.c

Usando N, g y n como argumentos, escribe una funci´on para realizar el hasheo.

```
def generate_hasheo_simple (N, g, n):

    H = power_mod(g, n, N)

    return H
```

3.3 Pregunta 1.d

Calcula el hash simple con N = 600107, g = 154835, n = 239715.

```
# /// Llamados a la funcion para algunas entradas ///

print("Hasheo (Pregunta d)")
print(generate_hasheo_simple(600107, 154835, 239715))

# ////////// Resultados obtenidos //////////

Hasheo (Pregunta d)
565219
```

3.4 Pregunta 1.e

Calcula el hash simple con N = 548155966307, g = 189830397891, n = 44344313866.

```
# /// Llamados a la funcion para algunas entradas ///

print("Hasheo (Pregunta e)")
print(generate_hasheo_simple(548155966307, 189830397891, 44344313866))

# ////////// Resultados obtenidos //////////

Hasheo (Pregunta e)
499803692828
```

3.5 Pregunta 1.f

Calcula el hash simple con N = 604766153, g = 12075635, n = 443096843.

```python
# /// Llamados a la funcion para algunas entradas ///

print("Hasheo (Pregunta f)")
print(generate_hasheo_simple(604766153, 12075635, 443096843))

# ////////// Resultados obtenidos //////////

Hasheo (Pregunta f)
290266411

```

3.6 Pregunta 1.g

Escribe una función que cree una colisión dado p y q. Demuestra que tu función funciona para algunos ejemplos.

```python
def buscar_colision(p, q):
    N = p * q
    phi_N = (p - 1) * (q - 1)

    while True:
        g = randint(2, N - 1)
        if gcd(g, N) == 1:
            break

    colision_encontrada = False

    while not colision_encontrada:
        m = randint(2, N - 1)
        n = randint(2, N - 1)

        hash_m = power_mod(g, m, N)
        hash_n = power_mod(g, n, N)

        if hash_m == hash_n:
            colision_encontrada = True

    return (g, m, n, hash_m, hash_n)

print("prueba para g:")

p = 61  # Primer número primo
q = 53  # Segundo número primo

g, m, n, hash_m, hash_n = buscar_colision(p, q)
print(f"g = {g}, m = {m}, n = {n}")
print(f"Hash de m: {hash_m}")
print(f"Hash de n: {hash_n}")
print(f"¿Colisión encontrada? {hash_m == hash_n}")

# ////////// Resultados obtenidos //////////

prueba para g:
```

```
g = 690, m = 2593, n = 373
Hash de m: 2916
Hash de n: 2916
Colisión encontrada: True
```

4 Conclusiones

El laboratorio permitió profundizar en el estudio de las funciones hash, un pilar esencial en la criptografía moderna. A través del desarrollo en SageMath, en la plataforma CoCalc, se realizaron ejercicios prácticos que facilitaron la comprensión de los mecanismos detrás de las funciones hash y su importancia en la seguridad de la información.

En primer lugar, la implementación de una función hash básica evidenció cómo el proceso de hasheo está directamente vinculado con la teoría de los números primos, y cómo la dificultad para factorizar el módulo N juega un papel clave en la seguridad criptográfica, similar a lo que ocurre en el cifrado RSA. Se evidenció que la generación de colisiones es un problema real y relevante en criptografía, pues permite que dos valores distintos generen el mismo hash, lo que pone en riesgo la integridad de los datos.

El uso de SageMath fue de gran ayuda para manejar de manera eficiente los cálculos con números grandes y operaciones modulares, lo que permitió una implementación rápida y precisa de los algoritmos de hasheo. Asimismo, se comprendió que la seguridad de una función hash no solo reside en la complejidad matemática subyacente, sino también en su capacidad para evitar colisiones y otros tipos de ataques.

En resumen, este laboratorio sirvió como una experiencia enriquecedora, proporcionando un equilibrio entre la teoría y la práctica de las funciones hash. Los ejercicios realizados no solo reforzaron los conceptos aprendidos, sino que también ofrecieron una visión más clara de los desafíos que enfrentan las funciones hash en aplicaciones criptográficas reales. Esta experiencia pone de manifiesto la necesidad de seguir investigando y mejorando estas funciones para asegurar su fiabilidad en sistemas criptográficos.

References

[1] W. Stallings, *Cryptography and Network Security: Principles and Practice*, 7th ed. London, UK: Pearson, 2017.