Laboratorio 6: RSA

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

November 11, 2024

1 Introducción

1.1 Contexto

El laboratorio 6 forma parte del curso de *Criptografía y Seguridad en la Información (TEL-252)* de la Universidad Técnica Federico Santa María. El objetivo principal de esta sesión es profundizar en el conocimiento y aplicación del algoritmo RSA, uno de los pilares fundamentales en la criptografía moderna. El laboratorio abarca desde la generación de claves públicas y privadas, hasta la implementación de funciones para cifrar, descifrar y firmar digitalmente mensajes. Adicionalmente, se evalúa la seguridad de firmas y se practican conceptos de factorización y exponentes para un entendimiento completo del proceso.

1.2 Objetivos

Los objetivos específicos de este laboratorio son:

- Comprender y aplicar los fundamentos matemáticos del algoritmo RSA.
- Implementar en SageMath funciones para la generación de claves, cifrado y descifrado RSA.
- Realizar el proceso de firma y verificación de mensajes utilizando RSA.
- Evaluar y validar la seguridad de firmas digitales mediante ejercicios prácticos.
- Simular y verificar el correcto funcionamiento de todas las operaciones implementadas mediante ejemplos concretos.

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

2 Metodología

2.1 Flujo del laboratorio

- En este laboratorio, se explican los conceptos clave de RSA, incluyendo la generación de claves/firmas digitales, encriptación y desencriptación, por parte del profesor y del ayudante. Se mencionan los objetivos específicos y el formato de entrega, aclarando que no habrá un avance al final de la clase.
- El desarrollo del laboratorio consiste en el uso de lenguaje Sage. Donde se debe desencriptación de un texto cifrado y viceversa, como tambien el cifrado de números y la generación de claves/firmas. Además, se implementan funciones de validación de firmas RSA. Tambien se verifican los resultados de los ejercicios.
- Finalmente se logro la implementación y pruebas de funciones RSA, desarrollando soluciones colaborativas y comparando resultados para asegurar la validez de las implementaciones.

2.2 Ejercicios ejecutados

Para éste laboratorio no se desarrollaron ejercicios durante la clase.

3 Ejercicios prácticos

Pregunta 1

Use Sage para responder las siguientes preguntas. Muestre toda su entrada/salida de Sage:

(a) **Desencriptación de un mensaje RSA**

Supongamos que la clave pública RSA se factoriza como $p = 6569$ y $q = 8089$, y el exponente público es $e = 11$. Para desencriptar el mensaje cifrado $c = 28901722$, utilizamos el siguiente código en Python:

```
p = 6569
q = 8089
n = p * q
e = 11
c = 28901722

def descifrado(n, e, c):
    d = valor_d(e, phi_euler(n))
    return pow(c, d) % n

mensaje_descifrado = descifrado(n, e, c)
print("Mensaje en claro:", mensaje_descifrado)
```

Esto calculará el valor de d usando la función `valor_d` y luego aplicará la fórmula de desencriptación para obtener el mensaje en claro.

(b) **Cifrado de un número**

Para cifrar el número 449 y enviarlo a alguien con clave pública $N = 37617577$ y $e = 529$:

```
n = 37617577
e = 529
m = 449

def cifrado(n, e, m):
    return pow(m, e) % n

mensaje_cifrado = cifrado(n, e, m)
print("Mensaje cifrado:", mensaje_cifrado)
```

Este código cifra el mensaje $m = 449$ utilizando la clave pública proporcionada.

(c) **Cálculo del exponente público**

Dado que los factores primos del módulo son $p = 1723$ y $q = 5381$, y que el exponente privado es $d = 223$, podemos calcular el exponente público e:

```
p = 1723
q = 5381
d = 223
n = p * q

def valor_e(d, phi):
    # Se determina el valor k de: d * e = 1 + k * phi(n)
    x = 1
    while True:
        if (1 + x * phi) % d == 0:
            return (1 + x * phi) / d
        x += 1

e = valor_e(d, phi_euler(n))
print("Exponente público e:", e)
```

Esto calculará el valor del exponente público e.

(d) **Generación de claves RSA y encriptación/desencriptación**

Para generar un par de claves públicas/privadas RSA y realizar una encriptación y desencriptación, usamos el siguiente código:

```
def generador_RSA(p, q):
    e, d = generador_valores(p, q)
    n = p * q
    return (e, n), (d, n)

p = 61
q = 53
publica, privada = generador_RSA(p, q)
print("Clave pública:", publica)
print("Clave privada:", privada)

mensaje = 30
mensaje_cifrado = cifrado(publica[1], publica[0], mensaje)
mensaje_descifrado = descifrado(privada[1], privada[0], mensaje_cifrado)
print("[{0}] -> [{1}] -> [{2}]".format(mensaje, mensaje_cifrado, mensaje_descifrado))
```

Esto generará un par de claves y cifrará el mensaje $m = 30$, para luego desencriptarlo y recuperar el mensaje en claro.

Pregunta 2

Use Sage para resolver los siguientes problemas: En las partes (a)-(c), determine si las siguientes firmas son buenas o malas.

(a) **Verificación de la firma**

Para determinar si la firma es buena, calculamos la clave privada d usando la función `valor_d` y luego ciframos el valor a firmar con d para verificar si coincide con la firma dada.

```
m = 821         # valor a firmar
c = 8674413     # firma
n = 13962799    # N
e = 3           # exponente publico

valor_constante = valor_d(e, phi_euler(n))
print("Clave publica : Pr[{0},{1}] -> Es buena la firma {3}: {2}".format(
    valor_constante, n, check_firma(n, valor_constante, m, c), m))
```

Esto verifica si la firma 8674413 es válida para el valor 821 con los parámetros dados.

(b) **Verificación de la firma**

```
m = 2478
c = 27535246
n = 34300129
e = 61

valor_constante = valor_d(e, phi_euler(n))
print("Clave publica : Pr[{0},{1}] -> Es buena la firma {3}: {2}".format(
    valor_constante, n, check_firma(n, valor_constante, m, c), m))
```

Verifica si la firma 27535246 es válida para el valor 2478.

(c) **Verificación de la firma**

```
m = 419
c = 2607727
n = 5898461
e = 23

valor_constante = valor_d(e, phi_euler(n))
print("Clave publica : Pr[{0},{1}] -> Es buena la firma {3}: {2}".format(
    valor_constante, n, check_firma(n, valor_constante, m, c), m))
```

Verifica si la firma 2607727 es válida para el valor 419.

(d) **Cálculo y verificación de la firma**

Para este inciso, calculamos la firma de 521 usando los factores primos $p = 3181$ y $q = 2677$ y el exponente público $e = 163$. Luego verificamos si la firma es correcta.

```
p = 3181
q = 2677
e = 163
m = 521
```

```
d = valor_d(e, phi_euler(3181 * 2677))

firma = cifrado((p * q), d, m)
firmar = descifrado((p * q), d, firma)

print("firma generada (encriptado) : {0} y su confirmacion (descifrado): {1}".format(firma,
```

Este código genera la firma para el mensaje 521 y la verifica desencriptándola para comprobar la validez.

Pregunta 3

El propósito de esta pregunta es implementar funciones de cifrado y descifrado RSA con Sage.

(a) **Generación de claves RSA**

Para generar las claves RSA, necesitamos seleccionar dos valores primos p y q, y calcular los exponentes e y d (público y privado, respectivamente). La función `generador_RSA` utiliza estos valores para crear un par de claves.

```
factores_primos = lambda valor:[base for base, exponente in factor(valor)]

def phi_euler(numero):
    valor = numero
    for x in factores_primos(numero):
        valor *= (1 - 1 / x)
    return valor

def generador_valores(p, q):
    phi = phi_euler(p * q)
    while True:
        e = randint(2, phi - 1)
        if gcd(phi, e) == 1 and e != valor_d(e, phi):
            d = valor_d(e, phi)
            return e, d

def generador_RSA(p, q):
    e, d = generador_valores(p, q)
    n = p * q
    return (e, n), (d, n)
```

(b) **Función de cifrado RSA**

La función de cifrado toma el mensaje m, el módulo n, y el exponente público e y devuelve el mensaje cifrado.

```
def cifrado(n, e, m):
    return pow(m, e, n)
```

(c) **Función de descifrado RSA**

La función de descifrado toma el mensaje cifrado c, el módulo n, y el exponente privado d y devuelve el mensaje en claro.

```
def descifrado(n, d, c):
    return pow(c, d, n)
```

(d) **Simulación de cifrado y descifrado**

Para demostrar que las funciones funcionan correctamente, generamos un par de claves, ciframos un mensaje y luego lo desciframos.

```
p = 61
q = 53
n = p * q

publica, privada = generador_RSA(p, q)
d = valor_d(publica[0], phi_euler(n))

print("Clave pública : Pu{0}\nClave privada : Pr{1}".format(publica, privada))

mensaje = 30
mensaje_cifrado = cifrado(n, publica[0], mensaje)
mensaje_descifrado = descifrado(n, d, mensaje_cifrado)

print("[{0}] -> [{1}] -> [{2}]".format(mensaje, mensaje_cifrado, mensaje_descifrado))
```

Este bloque de código muestra la clave pública y privada generadas, cifra el mensaje $m = 30$ y luego lo descifra para recuperar el mensaje original, confirmando así la funcionalidad de las funciones de cifrado y descifrado.

Pregunta 3

El propósito de esta pregunta es implementar funciones de cifrado y descifrado RSA con Sage.

(a) **Generación de claves RSA**

Para generar las claves RSA, necesitamos seleccionar dos valores primos p y q, y calcular los exponentes e y d (público y privado, respectivamente). La función `generador_RSA` utiliza estos valores para crear un par de claves.

```
factores_primos = lambda valor:[base for base, exponente in factor(valor)]

def phi_euler(numero):
    valor = numero
    for x in factores_primos(numero):
        valor *= (1 - 1 / x)
    return valor

def generador_valores(p, q):
    phi = phi_euler(p * q)
    while True:
        e = randint(2, phi - 1)
        if gcd(phi, e) == 1 and e != valor_d(e, phi):
            d = valor_d(e, phi)
            return e, d

def generador_RSA(p, q):
    e, d = generador_valores(p, q)
    n = p * q
    return (e, n), (d, n)
```

(b) **Función de cifrado RSA**

La función de cifrado toma el mensaje m, el módulo n, y el exponente público e y devuelve el mensaje cifrado.

```
def cifrado(n, e, m):
    return pow(m, e, n)
```

(c) **Función de descifrado RSA**

La función de descifrado toma el mensaje cifrado c, el módulo n, y el exponente privado d y devuelve el mensaje en claro.

```
def descifrado(n, d, c):
    return pow(c, d, n)
```

(d) **Simulación de cifrado y descifrado**

Para demostrar que las funciones funcionan correctamente, generamos un par de claves, ciframos un mensaje y luego lo desciframos.

```
p = 61
q = 53
n = p * q

publica, privada = generador_RSA(p, q)
d = valor_d(publica[0], phi_euler(n))

print("Clave pública : Pu{0}\nClave privada : Pr{1}".format(publica, privada))

mensaje = 30
mensaje_cifrado = cifrado(n, publica[0], mensaje)
mensaje_descifrado = descifrado(n, d, mensaje_cifrado)

print("[{0}] -> [{1}] -> [{2}]".format(mensaje, mensaje_cifrado, mensaje_descifrado))
```

Este bloque de código muestra la clave pública y privada generadas, cifra el mensaje $m = 30$ y luego lo descifra para recuperar el mensaje original, confirmando así la funcionalidad de las funciones de cifrado y descifrado.

Pregunta 4

El propósito de esta pregunta es implementar funciones de Sage para crear y verificar firmas RSA. Para estas preguntas, se puede usar cualquier respuesta de las preguntas anteriores.

(a) **Función para firmar un mensaje con RSA**

La siguiente función toma un mensaje y una clave privada RSA para generar una firma válida del mensaje.

```
def firma_rsa(mensaje, clave_privada):
    d, n = clave_privada
    firma = power_mod(mensaje, d, n)
    return firma
```

(b) **Función para verificar una firma RSA**

La función de verificación toma una firma RSA y el hash del mensaje, y determina si la firma es válida usando la clave pública.

```
def verificar_rsa(firma, hash_mensaje, clave_publica):
    e, n = clave_publica
    hash_calculado = power_mod(firma, e, n)
    if hash_calculado == hash_mensaje:
        verificar = True
    else:
        verificar = False
    return verificar
```

(c) **Simulación de firma y verificación**

Se muestra un ejemplo de firma y verificación, incluyendo casos válidos y no válidos para demostrar el funcionamiento correcto de las funciones.

```
p = 6569
q = 8089

# Claves en formato de tuplas
clave_publica, clave_privada = generador_RSA(p, q)

# Mensaje a firmar
mensaje = 12345  # Supongamos que este es el hash de un mensaje original

# Caso 1: Firma y verificación válidas
firma_valida = firma_rsa(mensaje, clave_privada)
es_valida = verificar_rsa(firma_valida, mensaje, clave_publica)
print(f"Caso válido - Firma generada: {firma_valida}")
print(f"¿La firma es válida? {es_valida}")

# Caso 2: Firma incorrecta
firma_incorrecta = firma_valida + 1  # Alteramos la firma
es_valida_incorrecta = verificar_rsa(firma_incorrecta, mensaje, clave_publica)
print(f"Caso inválido - Firma alterada: {firma_incorrecta}")
print(f"¿La firma es válida? {es_valida_incorrecta}")

# Caso 3: Hash incorrecto
hash_incorrecto = mensaje + 1  # Alteramos el hash
es_valida_hash_incorrecto = verificar_rsa(firma_valida, hash_incorrecto, clave_publica)
print(f"Caso inválido - Hash alterado: {hash_incorrecto}")
print(f"¿La firma es válida? {es_valida_hash_incorrecto}")
```

En el código anterior: - En el **Caso 1**, la firma es válida y la verificación es exitosa. - En el **Caso 2**, se altera la firma y la verificación falla, como se espera. - En el **Caso 3**, se altera el hash y la verificación también falla, confirmando el correcto funcionamiento de la función de verificación.

4 Conclusiones

En este laboratorio, se implementaron y analizaron diversas funciones relacionadas con el cifrado, descifrado y verificación de firmas en el esquema RSA. A través de los ejercicios, se comprendió la importancia de los parámetros de seguridad, como la selección adecuada de los exponentes y los números primos en la generación de claves RSA. Además, se demostró cómo el uso de claves públicas y privadas garantiza la autenticidad e integridad de los mensajes mediante la firma digital. La implementación de funciones de verificación permitió identificar si una firma o hash eran válidos, lo cual es fundamental para prevenir manipulaciones y ataques de fuerza bruta. En resumen, este laboratorio brindó una comprensión sólida sobre los fundamentos y la aplicación de RSA en criptografía, subrayando la relevancia de las técnicas de seguridad en la protección de la información.

References

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