

Informe Tarea 4 - Cripografia y seguridad en redes

José Proboste
Profesor: Roni Berezin
Ayudante: Javier Valenzuela

22 de Junio 2020

Índice

1. Introducción	3
2. RSA	4
3. Cifrado y Descifrado	7
4. Servidor HTTP	8
5. Test local	10
6. Preguntas	11
7. Conclusión	13

1. Introducción

Para esta tarea se implementará el algoritmo RSA, el cual es uno de los sistemas criptográficos más utilizados, ya que se considera muy seguro y robusto si se utilizan los parámetros adecuados, los cuales son utilizar numero primos muy grandes, lo suficientes para volver a este algoritmo casi imposible de invertir en un tiempo prudente. El lenguaje que utilizaré será Python, en el cual programaré el algoritmo RSA y un cifrador y descifrador basados en RSA para probar el correcto funcionamiento el algoritmo. Luego, la información se enviara a un servidor por medio de Postman, el cual retornará su llave publica y un mensaje, cifraré el mensaje y lo retornare al servidor para comprobar su eficacia. Paralelamente a estas acciones, interceptaré el trafico del servidor utilizando Wireshark para comprobar la seguridad del mismo.



Figura 1: Logo de RSA

2. RSA

Como ya mencioné, para programar el algoritmo RSA utilizaré Python, pero primero se debe comprender el funcionamiento de RSA, el cual detallaré a continuación:

1. Seleccionar 2 números primos, idealmente estos deberían ser muy grandes, pero por temas de tiempo y recursos, solo me limitaré a utilizar numero primos menores a 100. Los números a utilizar se seleccionarán de la siguiente manera:

```
p = sympy.randprime(0,100)
q = sympy.randprime(0,100)
```

Figura 2: Elección de números primos

2. Luego se selecciona un n que corresponde a la multiplicación aritmética de p y q y un ϕ que corresponde al resultado a la función de euler que se calcula como: $\phi(n) = (p-1) \cdot (q-1)$

```
n=p*q
fi = (p-1)*(q-1)
```

Figura 3: Calculo de n y ϕ

3. Siguiente se escoge un numero e menor que ϕ y coprime de este, para la elección de este numero, elegiré a todos los números coprimos de ϕ de manera iterativa partiendo en 2 y si su máximo común divisor es igual a 1, significa que estos números son coprimos y lo agrego a un arreglo, luego avanzo al siguiente hasta llegar a $\phi-1$, finalmente entre todos estos números, selecciono uno al azar y le asigno la variable e . La variable e también debería ser grande, pero al acotar p y q esta también se acota.

```

def mcm(a, b):
    while b != 0:
        a, b = b, a % b
    return a
for i in range(2, fi):
    if mcm(fi, i) == 1:
        arr_e.append(i)

e = random.choice(arr_e)

```

Figura 4: Elección de la variable e

4. Finalmente se determina la ultima variable d que se calcula mediante aritmética modular con tal de que satisfaga la igualdad $e \cdot d = 1 \cdot \text{mod}(\phi)$ esto se calcula con el inverso modular de $e \cdot \text{mod}(\phi)$ de tal manera que se busca el numero entre 2 y ϕ que al multiplicarse por e y aplicarles el modulo de m resulte 1, el valor que cumpla todas estas condiciones será asignado como d.

```

def inversoMod(a, m) :
    for i in range(2, m):
        if ((a * i) % m == 1):
            return i
d = inversoMod(e, fi)

```

Figura 5: Elección de la variable d

5. Como resultado de todas estas operaciones, estamos en condiciones de formular las llaves publicas y privadas, las cales corresponden a las siguientes:
- public key = (n,e)
 - private key = (d,p,q) o (n,d)

```
p = 71
q = 2
n = 142
fi = 70
e = 31
d = 61
public key = (142,31)
private key = (61,71,2)
private key = (142,61)
```

Figura 6: Ejemplo de ejecución del programa

3. Cifrado y Descifrado

Para el cifrado y descifrado del mensaje se usarán dos pequeños algoritmos los cuales deben cumplir las siguientes condiciones:

- **Cifrador:** El algoritmo de cifrado recibe la llave publica y el mensaje para retornar el mensaje cifrado, para esto debe cumplir la siguiente condición $c = m^e \bmod(n)$ lo cual resolví de la siguiente manera:

```
print "ingrese la llave publica"
n = input("Ingrese n: ")
e = input("Ingrese e: ")
m = input("Ingrese el mensaje: ")
c = pow(m,e,n)
print("-----")
print "mensaje cifrado: " + str(c)
print("-----")
```

Figura 7: Algoritmo de cifrado

- **Descifrador:** Por otro lado el algoritmo de descifrado es similar al de cifrado pero en este caso se recibe la llave privada acotada y el mensaje cifrado de tal manera que se cumpla la siguiente condición $m = c^d \bmod(n)$ cuyo codigo es el siguiente:

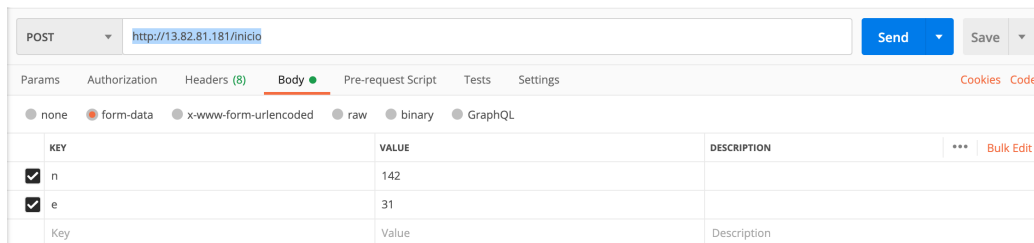
```
print "ingrese la llave privada"
n = input("Ingrese n: ")
d = input("Ingrese d: ")
c = input("Ingrese el mensaje cifrado: ")
m = pow(c,d,n)
print("-----")
print "mensaje descifrado: " + str(m)
print("-----")
```

Figura 8: Algoritmo de descifrado

Estos algoritmos nos ayudarán para comprobar la eficacia de nuestro algoritmo, además de para cifrar el mensaje recibido desde el servidor.

4. Servidor HTTP

Luego de la obtención de todos los datos al ejecutar el algoritmo se procede a trabajar con Postman para enviar los datos al servidor cuya URL es `http://13.82.81.181/inicio`, aquí debemos enviar un POST en cuyo Body se encuentren los datos de la llave publica:



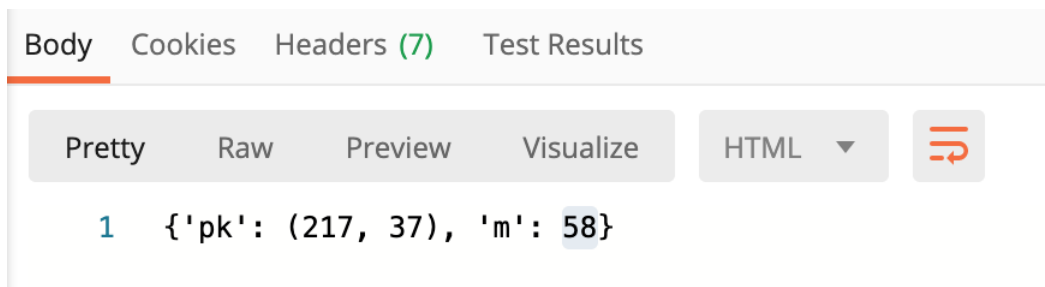
The screenshot shows the Postman interface for a POST request. The URL is `http://13.82.81.181/inicio`. The 'Body' tab is selected, and the 'form-data' type is chosen. A table with two rows is visible, representing the form data:

KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/> n	142	
<input checked="" type="checkbox"/> e	31	

Below the table, there are labels 'Key' and 'Value' with a 'Description' column.

Figura 9: Post al servidor con la llave publica

Luego de enviar los datos, el servidor retornará su llave pública junto a un mensaje numérico, de la siguiente forma:



The screenshot shows the Postman interface displaying the response body. The 'Body' tab is selected, and the 'Pretty' view is chosen. The response is a JSON object:

```
1  {'pk': (217, 37), 'm': 58}
```

Figura 10: Retorno del servidor

Luego utilizando el algoritmo de cifrado, se cifra el mensaje utilizando la llave publica del servidor y el mensaje:

```
ingrese la llave publica
Ingrese n: 217
Ingrese e: 37
Ingrese el mensaje: 58
-----
mensaje cifrado: 170
-----
```

Figura 11: Mensaje cifrado

Siguiente se prueba este mensaje en la URL <http://13.82.81.181/test> de la siguiente manera:

The screenshot shows a REST client interface with a POST request to <http://13.82.81.181/test>. The 'Body' tab is selected, and the data is in 'form-data' format. The form contains three fields: 'n' with value '142', 'e' with value '31', and 'm' with value '170'.

KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/> n	142	
<input checked="" type="checkbox"/> e	31	
<input checked="" type="checkbox"/> m	170	

Figura 12: Envío del mensaje cifrado al servidor

Como resultado el servidor retorna un mensaje de felicitaciones, certificando la eficacia en la obtención de la llave publica.

The screenshot shows the 'Body' tab of a REST client displaying the response '1 Lo lograste!'. The interface includes tabs for 'Body', 'Cookies', 'Headers (6)', and 'Test Results'. Below the tabs are buttons for 'Pretty', 'Raw', 'Preview', and 'Visualize', along with a dropdown menu set to 'HTML' and a refresh icon.

```
1 Lo lograste!
```

Figura 13: Mensaje de felicitaciones

5. Test local

Para comprobar la eficacia total de los datos, usaremos los algoritmos de cifrado y descifrado para comprobar tanto el funcionamiento de la llave publica como de la llave privada. Para eso utilizaremos la misma salida de la figura 6 y como mensaje utilizaremos $m=12$, obteniendo lo siguiente:

```
ingrese la llave publica
Ingrese n: 142
Ingrese e: 31
Ingrese el mensaje: 12
-----
mensaje cifrado: 18
-----
```

Figura 14: Prueba de los datos cifrado

Luego utilizaremos intentaremos descifrar el mensaje antes cifrado con el algoritmo de descifrado, entregando nuestra llave privada y el mensaje cifrado anteriormente, asi:

```
ingrese la llave privada
Ingrese n: 142
Ingrese d: 61
Ingrese el mensaje cifrado: 18
-----
mensaje descifrado: 12
-----
```

Figura 15: Prueba de los datos descifrado

De esta manera se puede comprobar que el algoritmo funciona a la perfección y que todos los datos están correctos.

6. Preguntas

1. ¿Es posible capturar la clave pública?

Si, ya que al trabajar con un servidor con protocolo HTTP lo vuelve vulnerable a interceptaciones en la comunicación y ataques man in the middle, lo cual demostraré utilizando el programa Wireshark para realizar esto, Paralelamente a las operaciones para enviar los datos al servidor, interceptaré el trafico generado entre mi ip y la del servidor. Luego de realizar todas las operaciones y filtrar el trafico a solo HTTP, se obtiene lo siguiente:

28	4.476318	192.168.0.9	13.82.81.181	HTTP	650	POST /inicio HTTP/1.1
32	4.641942	13.82.81.181	192.168.0.9	HTTP	381	HTTP/1.1 200 OK (text/html)
55	7.435889	192.168.0.9	13.82.81.181	HTTP	751	POST /test HTTP/1.1
56	7.582310	13.82.81.181	192.168.0.9	HTTP	306	HTTP/1.1 200 OK (text/html)

Figura 16: Trafico interceptado

Luego al analizar las dos primeras operaciones, es posible detectar todos los datos, tanto los enviados como los recibidos, lo queda demostrado aqui:

```
"n" . . . . 1 42 . . . . .
-----
----- 41148485
09113283 58656392
. . Conten t-Dispos
ition: f orm-data
; name=" e" . . . . 31
```

Figura 17: Datos enviados

```
▼ Line-based text data: text/html (1 lines)
{'pk': [217, 37], 'm': 58}
```

Figura 18: Datos recibidos

De esta manera pudimos identificar los datos enviados (n=142, e=31) y los datos recibidos ('pk': [217, 37], 'm': 58), quedando demostrado que es posible obtener la llave publica y el mensaje.

2. **¿Es posible obtener información del mensaje cifrado en el segundo paso con Wireshark?** De la misma manera que realicé la pregunta 1, responderé a la pregunta 2, pero esta vez utilizando los 2 últimos resultados interceptados, los cuales contienen lo siguiente:

```
rm-data;  name="n
" . . . 142  . . -----
-----
-----29  41928516
68816942  464083 . .
Content-  Disposit
ion: for  m-data;
name="e"  . . . 31 . .
-----
-----  ----2941
92851668  81694246
4083 . . Co ntent-Di
spositio n: form-
data; na me="m" . .
. . 170 . . -----
```

Figura 19: Datos recibidos

```
▼ Line-based text data: text/html (1 lines)
Lo lograste!
```

Figura 20: Datos recibidos

Obteniendo tanto los datos enviados al servidor (n=142, e=31, m=170) y el mensaje de retorno, por lo que queda demostrado que si es posible capturar el mensaje cifrado.

7. Conclusión

Con este trabajo pudimos profundizar mucho más en como trabaja el algoritmo de cifrado RSA, analizar sus pros y contras e implementarlo en un servidor HTTP, detectando sus falencias en temas de seguridad. Además del uso de diversas herramientas, como son Postman y Wireshark que nos ayudaron en el trabajo de implementar este algoritmo.