Writeup P3 Tarea 1

Writeup P3 Tarea 1

Nombre: Benjamín Aguilar Osorio
Equipo: L05 1NCR31BL35

Persona a cargo de la pregunta **Benjamín Aguilar Osorio**

Descripción general del problema

Descripción general: Se trata de un problema de criptografía asimétrica donde se intercepta la comunicación entre dos servidores. El primer servidor recibe un mensaje en hexadecimal y retorna una firma RSA del mensaje, y el segundo servidor recibe el output del primer servidor y entrega una respuesta dependiendo del primer mensaje.

El problema se resuelve cuando el segundo servidor recibe la firma del mensaje "¿cuál es la flag?", por lo que el primer servidor nunca entregará la firma para este mensaje.

Metodología de resolución

A continuación se darán los **pasos** que se llevaron a cabo para resolver el problema:

1. Inicialmente, para tener una idea del funcionamiento de los servidores, se trabaja entregando input a ambos. A partir de esto, se puede determinar que el servidor 1 recibe un texto en hexadecimal y entrega una llave RSA de este, mientras que el servidor 2 recibe una llave en RSA y entrega una respuesta dependiente de esta.
2. Luego, teniendo una idea inicial del funcionamiento de ambos servidores, se revisó el código de ambos servidores con el fin de comprobar el funcionamiento de la creación de la llave en RSA.
3. A partir del análisis del código base de ambos servidores, se puede determinar que estos no ocupan padding al momento de crear la llave de RSA, por lo que es posible realizar un blinding attack si se consigue la llave pública de estos.
4. Para poder conseguir la llave pública del RSA, se barajó la posibilidad de hacer un ciclo con el fin de encontrar el valor de n, donde cuando x = x, implicaría que se encontró el valor de n-1. Esta idea fue desechada cuando en el foro se mencionó que la llave pública se encontraba en el archivo del GitHub.
5. Teniendo la llave pública RSA codificada, se buscaron maneras de poder decodificar esta. Inicialmente se intentó decodificar mediante el uso de la web **CyberChef**, pero fallando en el intento. Finalmente, se decodificó la llave utilizando el siguiente código encontrado en Stack Overflow, pudiendo conseguir así el valor n y e de la llave pública::

```
>>> from Crypto.PublicKey import RSA
>>> f = open(".env.example","r")
>>> key = RSA.importKey(f.read())
>>> print key.n
>>> print key.e
```

6. Teniendo los valores de n y e, se prosiguió al cálculo del blinding attack. Para esto, inicialmente se hizo una transformación de hex a int y de int a hex mediante el uso de CyberChef, fallando miserablemente. Por lo que, tomando en consideración el hint dado por el profesor en el foro del curso, se volvió a analizar el código del servidor, obteniendo así las siguientes funciones para pasar código de hexadecimal a int y viceversa:

```
def to_int(x):
    return int.from_bytes(bytes.fromhex(x), byteorder='big')
def to_hex(x):
    return x.to_bytes((x.bit_length() + 7) // 8, byteorder='big').hex()
```

7. Teniendo una forma correcta de pasar de int a hex y viceversa, se volvió a realizar el blinding attack, fallando nuevamente. Luego de estar analizando el proceso de ataque un tiempo, se logró identificar que el problema recaía en el uso del valor añadido **R** y su inverso **R_inverso**, por lo que se decidió investigar más acerca de los requisitos de este valor.
8. Luego de una revisión en distintos foros, se llegó a la conclusión de que el error recaía en el cálculo de **R_inverso**, puesto que inicialmente este solamente era calculado como $\frac{1}{R}$, para luego ser reemplazado por $[R^{-1}]_n$, entregando así un valor entero que para una base n, sería el inverso de R.
9. Finalmente se realizó el blinding attack como sigue a continuación, obteniendo la flag CC5325{r54_p4dd1ng_4lw4ys!1111}

```
n=25701462380404112520675474334978405838467466733073287037745487711512577893
14309226085807877075390422649337132909474338162513127713861898809498194024967
78065159911928293653613443489182125450211932310041754139156427303612304578227
73332483606910374135368842026843812807536324608309043799418085861579164527150
68194756531920751245578022336159111596533207135267795115581170809102773059427
27166399104888841564261527991409742790762761003221759248126272900138853384531
88176348601179561381390467849510880038514920587183722986235566233843856011630
14549706159960945366782016978396456056176530580192034501828886217450706430376
34411421
e=3
r=3
r_inverso = pow(r,-1,n)
hex_real = "6375616c206573206c6120666c61673f"

int_real = to_int(hex_real)
fake_int = int_real * pow(r,e)
```

```
fake_hex = to_hex(fake_int)

fake_signature =
7923808055879179336134907151756940626306566805782271213038421628391309566789
7099879305598415834389613611018794014872768665443888594124974141322667138771
9292663638841353704932261590514717460238610433561741413260342907406619734024
7668000047357223051672753767399193445919538220057295760003362591750711688391
9691356702235171428319912319141300646387587784669709340041284029789191558927
3153448907460835635607135482697083770575953007605504991151707584645543260314
3053061653487607015162783915046948373040896628303024001522638330040866980428
4912154939213946810398268582544791419521333126352386989417936163133903122193
76147203
real_siganture = (fake_signature * r_inverso)%n
```