Práctica 6
Facultad de Ciencias
Criptografía y Seguridad

José de Jesús Galaviz Casas
galaviz@ciencias.unam.mx

Edgar Omar Arroyo Munguía
omar.am@ciencias.unam.mx

Luis Fernando Yang Fong Baeza
fernandofong@ciencias.unam.mx

29 de Enero 2020

1. Implementación de RSA

Tendrán que utilizar el script de la práctica 4, que genera números primos de tamaño arbitrario (al menos 100 dígitos) para generar a $n = pq$ con p y q primos, que tomarán como base, posteriormente, deberán generar una llave privada y una llave pública, generar un archivo llamado *pub_key*, recordando que la llave pública consta de (n, e), de igual manera, se debe de generar un *priv_key* para poder reutilizar siempre las llaves. de manera que al momento de cifrar un mensaje completamente arbitrario, se pueda decifrar guardando ambos archivos en sistema ejecutando su script.

Recordando el algoritmo de RSA, se tienen que escoger 3 números, (n, e) que son publicados, conocidos como la llave pública, de manera que si alguien nos quiere enviar un mensaje, supongamos que quiero enviar el mensaje `NO DERRAMES LA COCA`, asumiendo que A = 1, B = 2, ..., sucesivamente, ' '= 28 :

$$14|16|28|4|5|19|19|1|13|5|20|28|12|1|28|3|16|3|1$$

De manera que a cada uno de estos números hay que aplicarle el algoritmo de RSA, que consta de:

$$m_i^e \ mod \ n$$

y enviar esa información a nuestro receptor, en este caso, regresarlo en nuestra función de encripción y desencripción.

En cuanto al algoritmo de decifrado, simplemente vamos a recibir una lista de números en $\mathbb{Z}_n$ y aplicarles lo siguiente a cada número:

$$c_i^d \ mod \ n$$

Recordando que pudimos haber calculado a d de manera eficiente y rápida y que solo el programa lo conoce, pero pudiendo publicar la *pub_key* en cualquier lado, puesto que $(\phi(n), e) = 1$

2. Consideraciones

El tamaño de p y q, tiene que ser al menos de 100 dígitos, esto no debería de ser problema puesto que el generador de primos, soporta este número de dígitos, quien guste hacerlo de más dígitos, es bajo su propia decisión.

Para propósitos de pruebas, se ha agregado una función completamente auxiliar y privada, que se llama *__phi__* que regresa exactamente $\phi(n)$, esto no debería exisitir en realidad, puesto que es solo para verificar que el algoritmo sea correcto.

No tienen que hacer el mapeo que se muestra en los ejemplos, es solo para mostrar la claridad y funcionamiento del algoritmo, pueden utilizar `ord`.

Les recomiendo que primero vean las pruebas para que sepan qué atributos se espera que tenga su clase de RSA.

La verificación de los archivos no está incluída en las pruebas, sin embargo, sí se evaluará la generación de éstos archivos, no deben llevar ningún formato en particular, con que se pueda ver que en efecto corresponde a una llave pública y una privada respectivamente.

Como también necesitan encontrar el inverso multiplicativo de un número modulo n, se sugiere que también utilicen lo que ya se programó de las prácticas pasadas, como lo fue en cifrado afin, sin embargo, son consideraciones de cada equipo o persona.

3. Puntos extra

Existe otra forma de hacer RSA más eficiente y es con algo que traducido sería *esquema de relleno*, aunque en inglés se conoce como *padding scheme*, en lo que consiste es crear un número más grande en lugar de hacerlo para cada individuo, siguiendo el ejemplo de arriba, si suponemos que el mensaje es otra vez `NO DERRAMES LA COCA` y por fijar llaves públicas y privadas aleatorias y para fines didácticos, supongamos que $n = 71 * 101$, como $\phi(n) = 7000$, entonces solo tengo que escoger un número que sea primo relativo con él, viendo su factorización en primos, tenemos que $7000 = 2^3 * 5^3 * 7$, entonces puedo tomar como llave pública a 27 y es primo relativo de manera trivial, calculo su inverso *mod* $\phi(n)$ que es 5963, entonces una vez formada la llave pública que es $(7171, 27)$ y la llave privada que es $(7171, 5963)$, procedo de la siguiente manera.

14|16|28|4|5|19|19|1|13|5|20|28|12|1|28|3|16|3|1 una vez traducido el mensaje, entonces la idea es, tomar un cierto número de número b de dígitos siempre y cuando no pasen de n, el problema es cuando tenemos números de un solo dígito, puesto que si tomo 4 dígitos, en el caso de la parte 3|16|3, ¿cómo reconstruir el mensaje original?, se tendrían que hacer muchas suposiciones, entonces, para evitar problemas, se escriben todos esos números como $0x$, en caso de estar al inicio, simplemente representarían a un dígito más pequeño y en caso de estar en medio, nos sirve para diferenciar una vez que se haya desencriptado el mensaje, he aquí un ejemplo con $b = 4$

$$1416^{27} \ mod \ 7171$$
$$2804^{27} \ mod \ 7171$$
$$0519^{27} \equiv 519^{27} \ mod \ 7171$$
$$1901^{27} \ mod \ 7171$$

.
.
.

Y al momento de desencriptar, recibo exactamente un número y simplemente lo elevo a la llave privada, para que podamos obtener el número original.

$$
\begin{aligned}
1416^{27*5963} \ mod\ 7171 \\
2804^{27*5963} \ mod\ 7171 \\
0519^{27*5963} \equiv 519^{27*5963} \ mod\ 7171 \\
1901^{27*5963} \ mod\ 7171 \\
. \\
. \\
.
\end{aligned}
$$

Puesto que $(\phi(7171), 27) = 1$, entonces $27^{-1} \equiv 5963 \ mod\ \phi(7171)$, y solo tendría que tomar los primeros 2 dígitos para la primer letra, luego los siguientes 2 para la segunda, luego los primeros dos de la siguiente para obtener la siguiente y así sucesivamente. Obviamente el punto extra, está en implementar un *padding scheme* dado un criptosistema de RSA, con su llave pública y su llave privada.

Para quienes decidan implementar el *padding scheme* hay que marcar como `true` una variable de clase y tienen que escoger un tamaño adecuado para particionar los mensajes, si no, hay que marcar esta variable de clase como `false` para que las pruebas corran de manera normal.