PRÁCTICA2: CRIPTOGRAFÍA

Álvaro Palomar y Pablo Rodríguez: GP12B

ÍNDICE

CAMBIOS MODULAR.PY

-En general no teníamos casi ningún fallo en el modular por lo que no nos hemos centrado mucho en él, aun así, los nuevos cambios implementados en el modular son:

- Inicialmente, un par de paréntesis en la función Euler que hacía que la función siempre devolviera 0. (Esos paréntesis fueron una errata que no sabemos en qué momento llegaron ahí y que hizo que nos dejara de funcionar).

- Asimismo, hemos modificado la estructura de un par de excepciones, ya que estas capturaban el error y devolvían un string, en lugar de un texto proveniente de una excepción.

- También, hemos modificado en alguna operación su estructura para que quede correctamente según la guía PEP-8.

- Por último, hemos cambiado la tipificación de alguna función para su correcta ejecución en las aulas virtuales, al no tener la versión de Python más actualizada.

GENERACIÓN DE CLAVES

-El .py de generación de usuarios se ha estructurado con un par de funciones y un main así como importación de las librerías necesarias para llevar a cabo este programa, Las funciones son existentes dentro de este registrarusuario.py son:

- pedir_datos(): Esta función recoge los datos que tienen que ser dados por el usuario, primero pedirá el nombre donde se aceptara cualquier tipo de texto comprobando si exista algún usuario con ese nombre, en caso de ser así pedirá si al usuario que indique si quiere cambiar sus contraseñas y en caso de querer tendrá 3 intentos para introducir la clave privada y en caso de no querer volverá a pedir el nombre para, después pedirá los dos números que representan los límites entre los números que pueden estar los primos (Con un while nos aseguramos de que lo que introducimos son números así como que el límite inferior es menor que el superior) y por último pedirá la cantidad de dígitos de padding (También asegura con un while que lo que se introduce es un número).

- crear_ficheros(): Esta función recibe el nombre del usuario, los datos generados en la función del rsa generar_claves() y por último los digitos de padding. Primero, dicha función comprueba si existe un directorio llamado usuarios y si no es así lo crea; después, genera un fichero de claves públicas donde se introducirán las claves a las que pueden tener acceso el resto de usuario y otro fichero de claves privadas, donde se incluirá la clave privada de dicho usuario.

- Main: En el main primero llamaremos a la función pedir_datos(), los cuales guardaremos en diferentes variables, después llamamos a la función del rsa.py que genera las claves a partir de los datos introducidos y por último llamamos a la función crear_ficheros() la cual nos termina de crear los ficheros.txt que necesitamos del usuario.

-A la hora de generar claves de mas de 20 cifras el programa es poco eficiente ya que tiene que generar una lista de todos los primos entre esos números y es muy lento.

ATAQUES RSA

romper_clave(): Para conocer la clave privada d tal que de = 1 (mod phi(n)), basta con pasar e al otro lado de dicha ecuación y, por tanto, calcular la inversa de e (mod phi(n)), lo cual puede resultar sencillo si los primos de los que proviene n son conocidos, pero largo y costoso si no lo son. Como la clave d es privada, debemos calcular aplicar la función Euler a n para sacar el módulo de la congruencia, en lugar de multiplicar (primo1 -1) * (primo2-1) y, por tanto, la eficiencia de la función depende, en gran parte, de nuestro Euler en el fichero "modular" y, por ende, del factorizar, para finalmente aplicar la función inversa_mod_p con e y módulo phi(n).

ataque_texto_elegido(): Tras investigar en internet, nos hemos dado cuenta de que la manera más optimizada para realizar esta función no es hacerlo de la manera más obvia a priori ya que, como hemos visto en romper_clave(), puede ser muy costoso para valores de n muy grandes. Por ello, hemos optado por hacer un diccionario cuyas claves son los valores ASCII del alfabeto estándar, incluyendo las mayúsculas y minúsculas con tildes, cifradas con la n y la e dadas y sus valores el carácter que representa cada uno realmente, para posteriormente recorrer la lista de los números cifrados y atribuirles su carácter real.

Esta función puede descifrar cualquier texto cifrado dado, siempre y cuando los caracteres reales estén dentro del alfabeto estándar y, como se pide en la práctica, no tenga dígitos de padding. Así, hemos logrado descifrar el texto del "Criptograma X.txt" en décimas de segundo:

"Yo he visto cosas que vosotros no creeríais. Atacar naves en llamas más allá de Orión. He visto rayos-C brillar en la oscuridad cerca de la Puerta de Tannhäuser. Todos esos momentos se perderán en el tiempo, como lágrimas en la lluvia."

BIBLIOGRAFÍA

https://en.wikipedia.org/wiki/Chosen-plaintext_attack

https://codedinsights.com/