

FIRMAS Y CERTIFICADOS DIGITALES: PROYECTO FINAL

JUAN CAMILO TAMAYO MOLINA

JUAN DAVID GARCÍA REYES

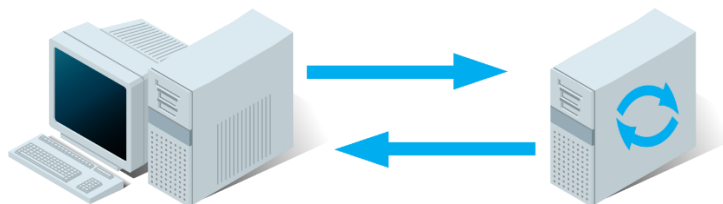
RESUMEN

En el siguiente documento se explica el funcionamiento de dos algoritmos desarrollados para el curso de firmas y certificados digitales, así como su respectivo código fuente y ejemplos de ejecución de los mismos. Los algoritmos tratados son: Algoritmo de diffie-hellman con implementación de firma basada en algoritmo DSA; y criptoanálisis del cifrado por transposición de julio cesar.

DIFFIE-HELLMAN CON FIRMA BASADA EN ALGORITMO DSA

En este algoritmo se realiza una simulación de la interacción de un cliente y un servidor; cada uno posee su respectivo juego de llaves públicas y privadas, cuya estructura se basa en el estándar DSA o Digital Signature Algorithm por sus siglas en inglés, presentadas a través de un archivo en formato pem, en caso de que el usuario desee proveerlos, o en su defecto, se genera un juego de llaves para ambos. posterior a esto, la ejecución se realiza tal cual el algoritmo tradicional de diffie-hellman:

El cliente, en este caso, el usuario, provee al script con los parámetros siguientes:



valores cliente	valores públicos	Valores servidor
a = 17	p = 17 g = 3	

1. un número privado "a"
2. un número primo "p", el cual se recomienda y sea mayor a 1024 bits
3. un número generador "g", que se recomienda y sea una raíz primitiva del módulo "p" seleccionado anteriormente
4. Un archivo con extensión .pem correspondiente al juego de llaves (pública y privada) del cliente (opcional).
5. Un archivo con extensión .pem correspondiente al juego de llaves (pública y privada) del servidor (opcional).

Cabe destacar que tanto "p" como "g" serán de carácter público; es decir, conocido tanto por el cliente como por el servidor.

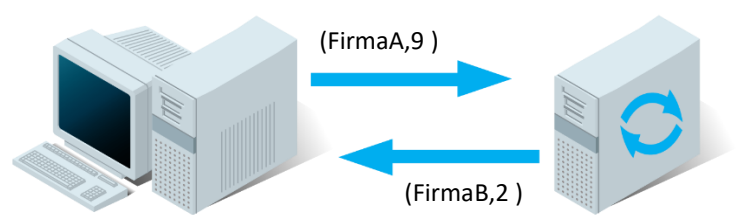
El servidor realizará el proceso de elegir un número privado "b", ya que los otros parámetros de carácter público ya fueron establecidos por el cliente.

valores cliente	valores públicos	Valores servidor
a = 17	p = 17 g = 3	b=772238

Tanto el cliente como el servidor calcularán un número público, elevando el número generador "g" a la potencia de su número privado correspondiente, y luego obteniendo el modulo con el número primo público.

valores cliente	valores públicos	Valores servidor
a = 17 A = 9 $A = g^a \text{ mod } p$	p = 17 g = 3	b=772238 B = 2 $B = g^b \text{ mod } p$

Antes de enviar los parámetros como en el algoritmo diffie-hellman tradicional, tanto el cliente como el servidor firman una cadena o String, generando un digest o un compendio, compuesto por el numero público de ambas partes, es decir, el numero público del cliente y el número público del servidor son concatenados de tal manera que la cadena resultante es igual a "ab", luego, esta cadena es firmada con la llave privada respectiva de cada uno, y posteriormente, se envía en conjunto al número público de cada uno.



valores cliente	valores públicos	Valores servidor
Ca = C("92") a = 17 A = 9	p = 17 g = 3	Cb = C("92") b=772238 B = 2

*Ca y Cb corresponden a la cadena firmada por cada terminal, y la función "C" corresponde al proceso de "digerir" la cadena para firmarla con la llave privada de cada uno. Posteriormente, ambas partes reciben el número y llave públicos del otro, Con el fin de proceder a validar las firmas pertinentes.

Ambas partes reciben el número y llave públicos del otro, Con el fin de proceder a calcular la clave acordada cuyo cálculo se obtiene con la formula $B^a \text{ mod } p$, en el caso del cliente, y $A^b \text{ mod } p$ en el caso del servidor. El resultado debe ser el mismo para ambas partes, confirmando que se tuvo éxito en la operación.

valores cliente	valores públicos	Valores servidor
Ca = C("92") a = 17 A = 9 Ka = 4 $B^a \text{ mod } p$	p = 17 g = 3	Cb = C("92") b=772238 B = 2 Kb = 4 $A^b \text{ mod } p$

Si al momento de validar la firma, la cadena firmada no concuerda con la verificación que realiza cada uno, será arrojado un error de verificación y se abortará la "conexión" o "transacción".

valores cliente	valores públicos	Valores servidor
Ca = C("92") a = 17 A = 9 Ka = 4 Va = Ps(Ca, FirmaB)	p = 17 g = 3	Cb = C("92") b=772238 B = 2 Kb = 4 Vb = Pc(Cb, FirmaA)

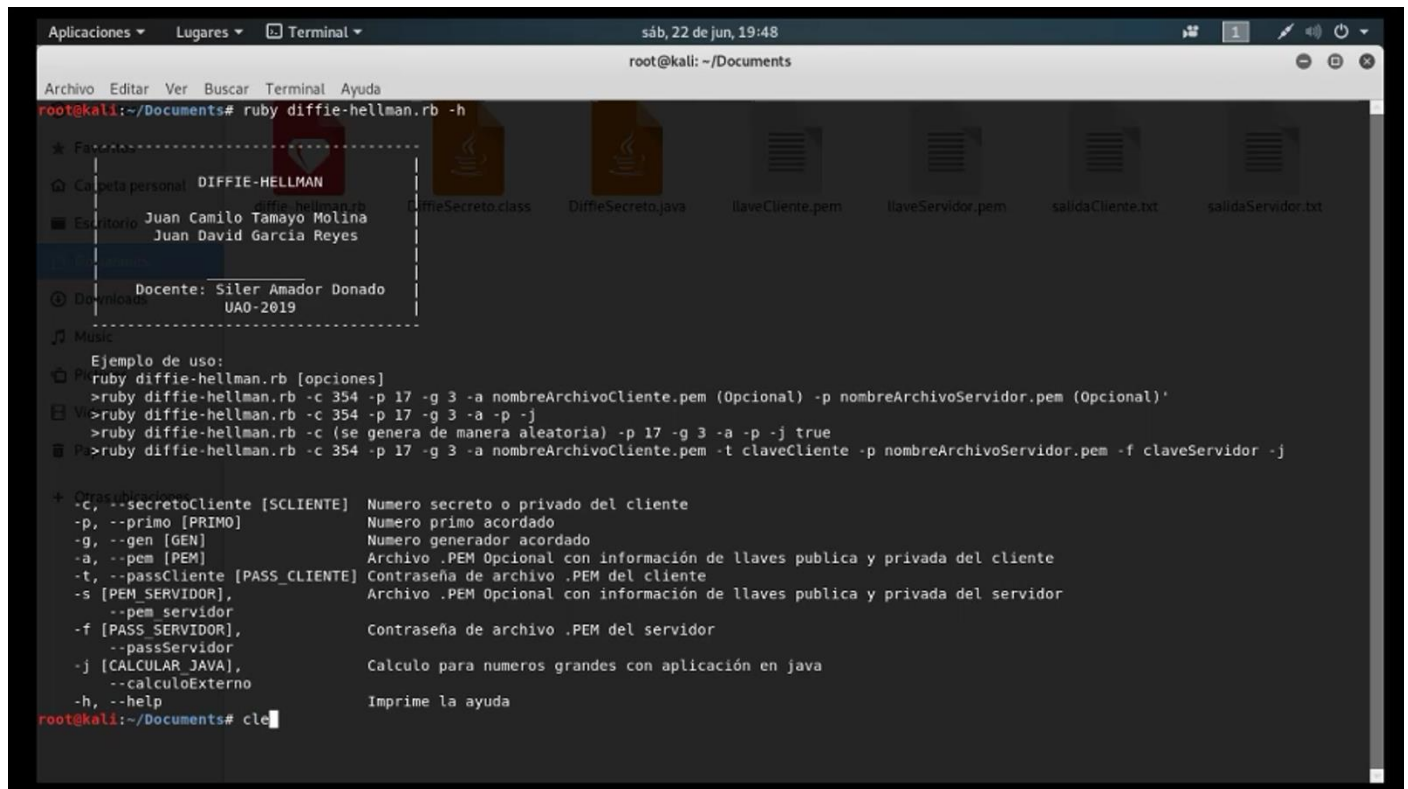
*Va y Vb corresponden a la validación de las firmas por medio de una función valida a través de la llave pública del servidor si su digest generado de la cadena concatenada, concuerda con esta y con la firma recibida, El servidor realiza la misma tarea, tomando la llave pública del cliente y validando con su digest y con la firma recibida.

La razón de ser de la modificación se da a causa de la simpleza del algoritmo de de diffie-hellman trivial. Aunque se pueda dificultar la tarea de determinar la clave acordada frente a un tercero, aún existe la posibilidad de ejecutar un ataque de tipo "hombre en el medio", ya que con el algoritmo no se tiene conocimiento de con quién se está intercambiando claves, solo se sabe que nadie hasta el momento ha calculado la clave acordada a través de "snooping" o fisgoneo en la red.

Utilizando el estándar DSA, se tiene una capa extra de seguridad, ya que se añade un mecanismo de autenticación en el cual se valida tanto la llave acordada como la firma de las partes involucradas, para hacer la comunicación más segura.

A continuación se podrá visualizar un ejemplo del funcionamiento paso a paso del script.

Al escribir el comando `"ruby diffie-hellman.rb"` junto a la bandera `"-h"` se nos mostrará la ayuda del script, con ejemplos pertinentes:



A continuación pondremos a prueba el script escribiendo la siguiente línea “`ruby diffie-hellman.rb -c -p 17 -g 3 -a -t -s -f -j`” lo cual hará lo siguiente:

- Creará de manera aleatoria un numero privado para el cliente (-c sin argumentos).
- Asignará el valor de 17 al primo a utilizar (-p).
- Asignara el valor de 3 al número generador (-g).
- Generará de manera automática un juego de llaves para el cliente (-a sin argumentos).
- Generará una clave predeterminada para el juego de llaves del cliente (-t sin argumentos).
- Generará de manera automática un juego de llaves para el servidor (-s sin argumentos).
- Generará una clave predeterminada para el juego de llaves del servidor (-f sin argumentos).
- Omitirá la utilización de un programa externo en java para cálculo de números de mayor tamaño

Al ejecutarlo se mostrará la siguiente pantalla:

```
Aplicaciones ▾ Lugares ▾ Terminal ▾ sáb, 22 de jun, 19:49 root@kali: ~/Documents
Archivo Editar Ver Buscar Terminal Ayuda
root@kali:~/Documents# ruby diffie-hellman.rb -c -p 17 -g 3 -a -t -s -f -j
numero secreto invalido o no asignado, será asignado uno al azar
AVISO: No fue otorgada una ruta de archivo PEM o el indicado no existe, se procede a
configurar un juego de llaves por defecto cliente
AVISO: No fue otorgada una contraseña para el archivo .PEM del cliente, se utilizará la clave por
defecto en caso de que se haya proveído uno
AVISO: No fue otorgada una ruta de archivo PEM o el indicado no existe, se procede a
configurar un juego de llaves por defecto para el servidor
AVISO: No fue otorgada una contraseña para el archivo .PEM del servidor, se utilizará la clave por
defecto en caso de que se haya proveído uno
AVISO: Los parametros otorgados serán calculados con el script de ruby
Secreto cliente: 997119
Secreto servidor: 687797
Se recomienda un primo mayor a 1024 bits (es decir, 2 elevado a la 1024) para mayor seguridad
Modulo Primo: 17
Numero generador: 3
Generando llaves para el cliente...
Generando llaves para el servidor...
calculando valor público para el cliente...
calculando valor público para el servidor...
VALOR PUBLICO DEL CLIENTE: 6
VALOR PUBLICO DEL SERVIDOR: 5
descifrando y computando el secreto compartido desde el cliente...
descifrando y computando el secreto compartido desde el servidor...
validado servidor!! (Firma validada)
validado cliente!! (Firma validada)
VALOR SECRETO CONOCIDO (CALCULO CLIENTE): 7
VALOR SECRETO CONOCIDO (CALCULO SERVIDOR): 7
Tiempo de ejecución: 1.896471215
root@kali:~/Documents#
```

De los valores anteriores se destacan los valores públicos del cliente y el servidor, y sus valores secretos conocidos o llave acordada.

Cabe destacar que la aplicación mostrará un aviso pertinente por cada bandera a la cual se le digite un parámetro invalido o se deje sin parámetros, por ejemplo, si no aplicamos un parámetro a la bandera del número secreto, se nos notificará que será creado uno de manera aleatoria.

En el siguiente ejemplo se enviarán todos los parámetros para las banderas opcionales.

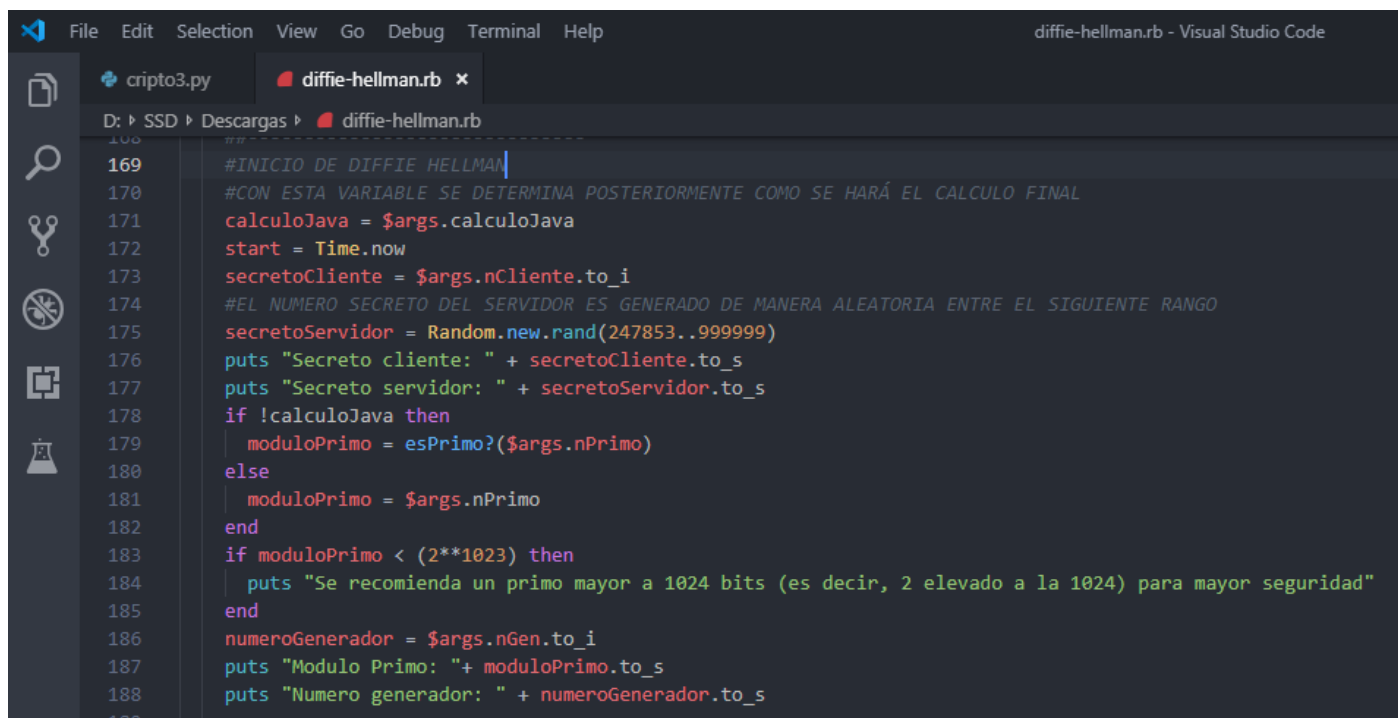
```
Aplicaciones ▾ Lugares ▾ Terminal ▾ sáb, 22 de jun, 19:59 root@kali: ~/Documents
Archivo Editar Ver Buscar Terminal Ayuda
root@kali:~/Documents# ruby diffie-hellman.rb -c 2314 -p 17 -g 3 -a llaveCliente.pem -t cliente -s llaveServidor.pem -f servidor -j true
cliente
AVISO: Los parametros otorgados serán calculados con el programa DiffieSecreto (en Java)
Secreto cliente: 2314
Secreto servidor: 870318
Se recomienda un primo mayor a 1024 bits (es decir, 2 elevado a la 1024) para mayor seguridad
Modulo Primo: 17
Numero generador: 3
Cargando PEM con llaves para el cliente...
Cargando PEM con llaves para el servidor...
calculando valor público para el cliente...
calculando valor público para el servidor...
VALOR PUBLICO DEL CLIENTE: 8
VALOR PUBLICO DEL SERVIDOR: 2
calculando con aplicación externa...
cargando calculo externo para el cliente...
cargando calculo externo para el servidor...
validado servidor!! (Firma validada)
validado cliente!! (Firma validada)
VALOR SECRETO CONOCIDO (CALCULO CLIENTE): 4
VALOR SECRETO CONOCIDO (CALCULO SERVIDOR): 4
Tiempo de ejecución: 0.140390988
root@kali:~/Documents#
```

Se puede observar cómo se omiten los mensajes de aviso de las banderas opcionales pues se enviaron los argumentos respectivos.

A continuación, se muestran los fragmentos principales del código. En la siguiente imagen se observa cómo se inicia el contador para determinar el tiempo de ejecución, que corresponde a la variable “start”, junto a la variable “calculoJava” que determinará si se calculará con el script de manera nativa o con la aplicación en java.

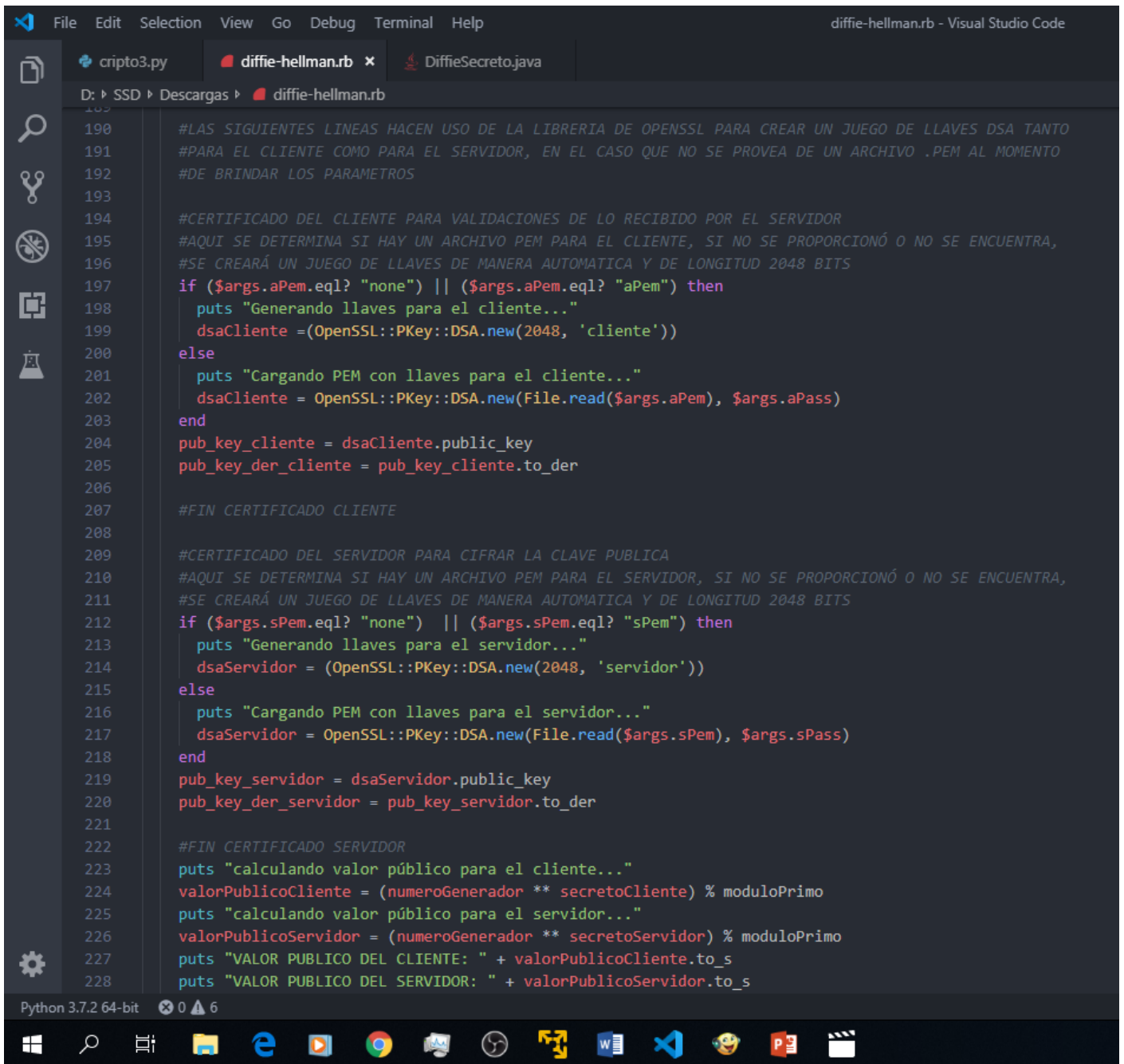
Posteriormente se determina si validar el numero primo o no, ya que dependiendo de por medio de que opción se haga el cálculo, esto podrá hacerse, o no, ya que, si el número es muy grande, ruby no podrá validarlo, o le tomaría eternidades.

También se valida si el numero primo es menor a 2 elevado a la 1023 ya que lo recomendado es que esté por encima de dicho valor. Sin embargo, este solo arroja un mensaje de sugerencia o recomendación y no fuerza a que tenga que manejarse por encima de ese rango.



```
169  #INICIO DE DIFFIE HELLMAN
170  #CON ESTA VARIABLE SE DETERMINA POSTERIORMENTE COMO SE HARÁ EL CALCULO FINAL
171  calculoJava = $args.calculoJava
172  start = Time.now
173  secretoCliente = $args.nCliente.to_i
174  #EL NUMERO SECRETO DEL SERVIDOR ES GENERADO DE MANERA ALEATORIA ENTRE EL SIGUIENTE RANGO
175  secretoServidor = Random.new.rand(247853..999999)
176  puts "Secreto cliente: " + secretoCliente.to_s
177  puts "Secreto servidor: " + secretoServidor.to_s
178  if !calculoJava then
179    moduloPrimo = esPrimo?($args.nPrimo)
180  else
181    moduloPrimo = $args.nPrimo
182  end
183  if moduloPrimo < (2**1023) then
184    puts "Se recomienda un primo mayor a 1024 bits (es decir, 2 elevado a la 1024) para mayor seguridad"
185  end
186  numeroGenerador = $args.nGen.to_i
187  puts "Modulo Primo: " + moduloPrimo.to_s
188  puts "Numero generador: " + numeroGenerador.to_s
```

En el siguiente recuadro se puede observar el proceso de generación del juego de llaves tanto para el cliente como para el servidor, por medio del cual, si no se provee al script de los archivos PEM correspondientes, se generarán de manera automática, con una contraseña por defecto.



```
189
190 #LAS SIGUIENTES LINEAS HACEN USO DE LA LIBRERIA DE OPENSLL PARA CREAR UN JUEGO DE LLAVES DSA TANTO
191 #PARA EL CLIENTE COMO PARA EL SERVIDOR, EN EL CASO QUE NO SE PROVEA DE UN ARCHIVO .PEM AL MOMENTO
192 #DE BRINDAR LOS PARAMETROS
193
194 #CERTIFICADO DEL CLIENTE PARA VALIDACIONES DE LO RECIBIDO POR EL SERVIDOR
195 #AQUI SE DETERMINA SI HAY UN ARCHIVO PEM PARA EL CLIENTE, SI NO SE PROPORCIONÓ O NO SE ENCUENTRA,
196 #SE CREARÁ UN JUEGO DE LLAVES DE MANERA AUTOMATICA Y DE LONGITUD 2048 BITS
197 if ($args.aPem.eql? "none") || ($args.aPem.eql? "aPem") then
198   puts "Generando llaves para el cliente..."
199   dsaCliente =(OpenSSL::PKey::DSA.new(2048, 'cliente'))
200 else
201   puts "Cargando PEM con llaves para el cliente..."
202   dsaCliente = OpenSSL::PKey::DSA.new(File.read($args.aPem), $args.aPass)
203 end
204 pub_key_cliente = dsaCliente.public_key
205 pub_key_der_cliente = pub_key_cliente.to_der
206
207 #FIN CERTIFICADO CLIENTE
208
209 #CERTIFICADO DEL SERVIDOR PARA CIFRAR LA CLAVE PUBLICA
210 #AQUI SE DETERMINA SI HAY UN ARCHIVO PEM PARA EL SERVIDOR, SI NO SE PROPORCIONÓ O NO SE ENCUENTRA,
211 #SE CREARÁ UN JUEGO DE LLAVES DE MANERA AUTOMATICA Y DE LONGITUD 2048 BITS
212 if ($args.sPem.eql? "none") || ($args.sPem.eql? "sPem") then
213   puts "Generando llaves para el servidor..."
214   dsaServidor = (OpenSSL::PKey::DSA.new(2048, 'servidor'))
215 else
216   puts "Cargando PEM con llaves para el servidor..."
217   dsaServidor = OpenSSL::PKey::DSA.new(File.read($args.sPem), $args.sPass)
218 end
219 pub_key_servidor = dsaServidor.public_key
220 pub_key_der_servidor = pub_key_servidor.to_der
221
222 #FIN CERTIFICADO SERVIDOR
223 puts "calculando valor público para el cliente..."
224 valorPublicoCliente = (numeroGenerador ** secretoCliente) % moduloPrimo
225 puts "calculando valor público para el servidor..."
226 valorPublicoServidor = (numeroGenerador ** secretoServidor) % moduloPrimo
227 puts "VALOR PUBLICO DEL CLIENTE: " + valorPublicoCliente.to_s
228 puts "VALOR PUBLICO DEL SERVIDOR: " + valorPublicoServidor.to_s
```

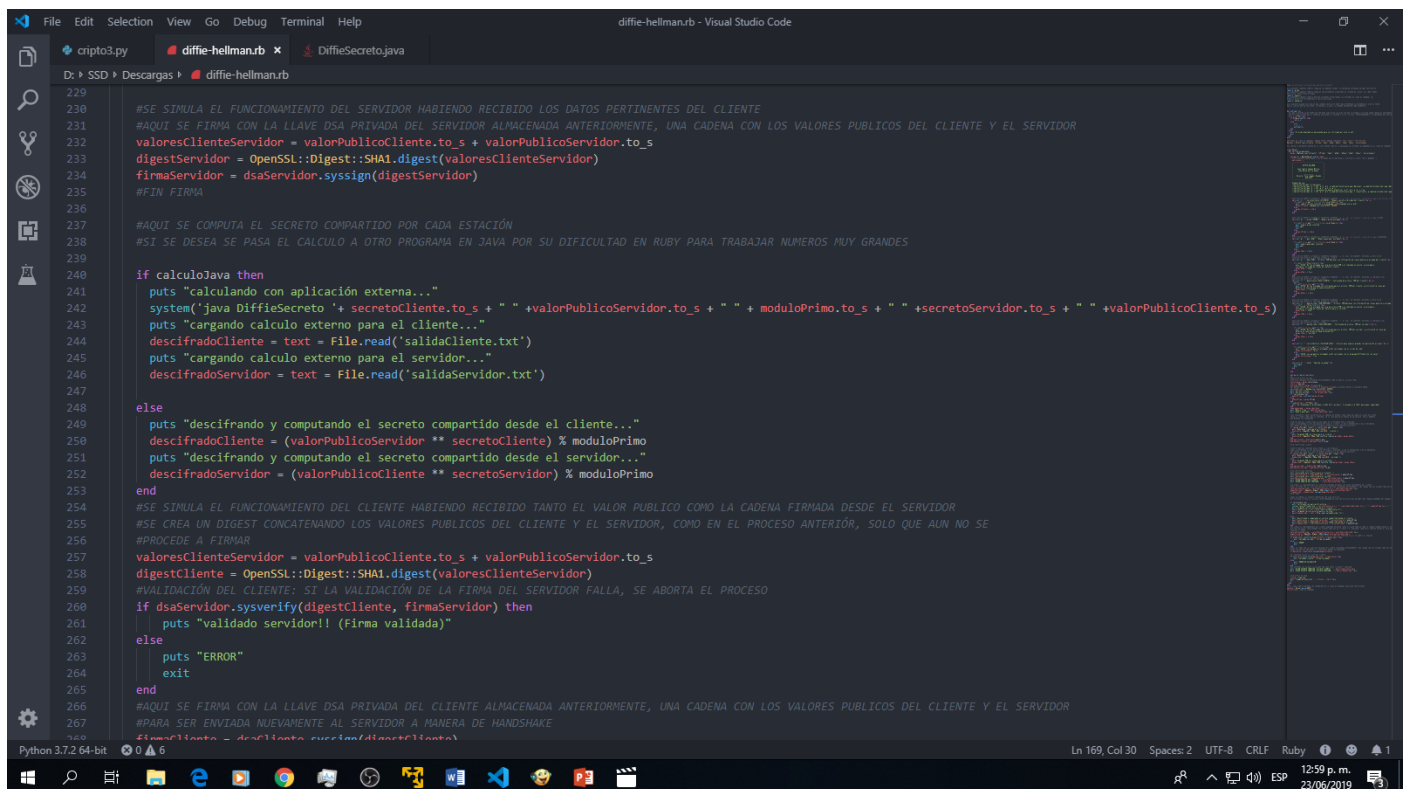
Como se menciona anteriormente, el script realiza una simulación entre la comunicación entre un cliente y un servidor, lo cual queda plasmado en la siguiente imagen, con sus comentarios respectivos.

El cliente y el servidor crean un digest con su llave privada, es decir, firman la cadena correspondiente a la concatenación de el numero público de cada uno. Para poder compararlo y validarlo con la firma que es enviada en conjunto al número público.

También aquí es donde se valida si enviar los parámetros al programa externo en java o simplemente dejar que el script en ruby los procese.

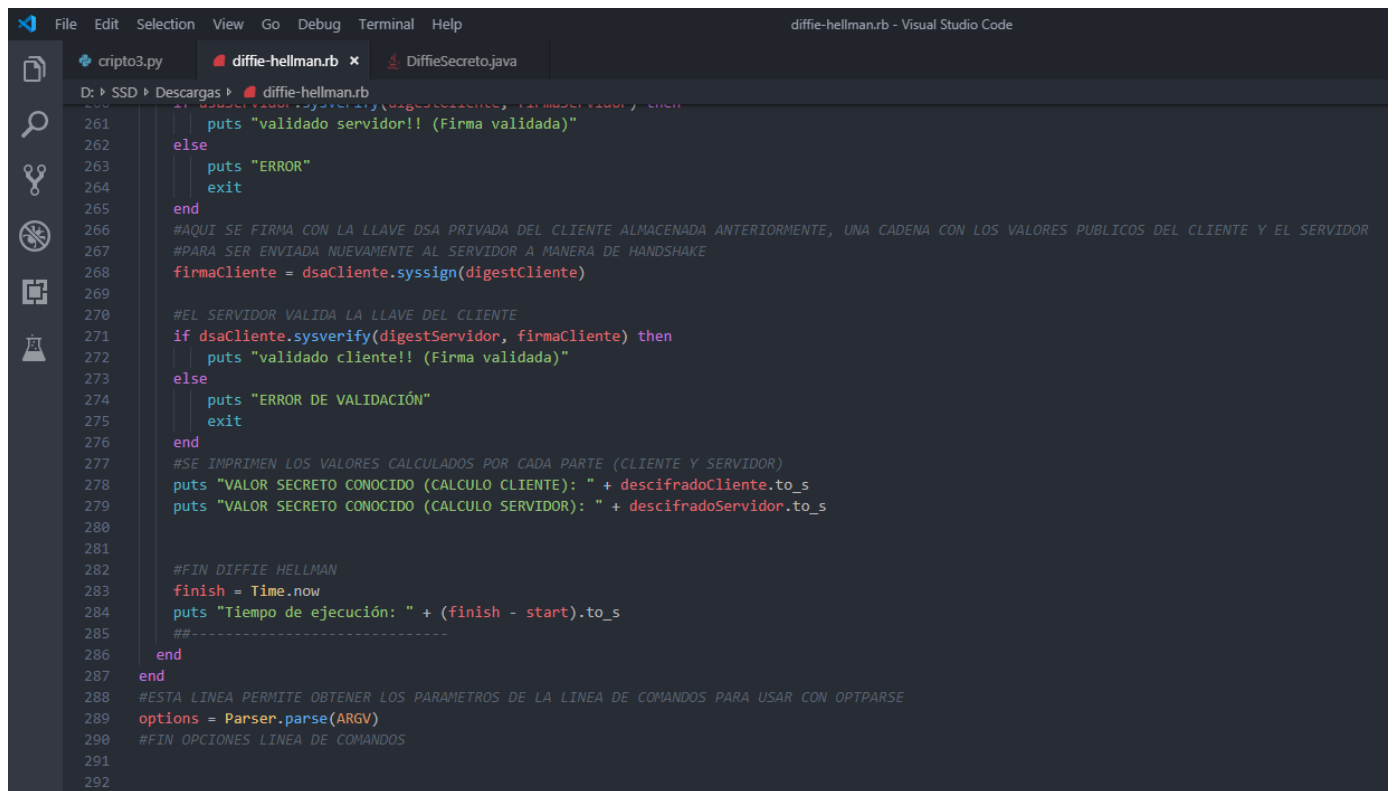
En caso de presentarse un error se procede a abortar la conexión, esto se debe a que, en la hipotética situación en que un atacante tipo hombre en el medio intentase hacerse pasar por un servidor, este no podría mimetizar las llaves correspondientes de cada uno.

Este script se basa en la premisa de que cada terminal posee la llave publica del otro, con el fin de que se mantenga segura la comunicación, y de que estas no tengan que ser enviadas también por un canal inseguro.



```
229
230
231 #SE SIMULA EL FUNCIONAMIENTO DEL SERVIDOR HABIENDO RECIBIDO LOS DATOS PERTINENTES DEL CLIENTE
232 #AQUI SE FIRMA CON LA LLAVE DSA PRIVADA DEL SERVIDOR ALMACENADA ANTERIORMENTE, UNA CADENA CON LOS VALORES PUBLICOS DEL CLIENTE Y EL SERVIDOR
233 valoresClienteServidor = valorPublicoCliente.to_s + valorPublicoServidor.to_s
234 digestServidor = OpenSSL::Digest::SHA1.digest(valoresClienteServidor)
235 firmaServidor = dsaServidor.syssign(digestServidor)
236 #FIN FIRMA
237
238 #AQUI SE COMPUTA EL SECRETO COMPARTIDO POR CADA ESTACIÓN
239 #SI SE DESEA SE PASA EL CALCULO A OTRO PROGRAMA EN JAVA POR SU DIFICULTAD EN RUBY PARA TRABAJAR NUMEROS MUY GRANDES
240
241 if calculoJava then
242   puts "calculando con aplicación externa..."
243   system('java DiffieSecreto '+ secretoCliente.to_s + " " + valorPublicoServidor.to_s + " " + moduloPrimo.to_s + " " + secretoServidor.to_s + " " + valorPublicoCliente.to_s)
244   puts "cargando calculo externo para el cliente..."
245   descifradoCliente = text = File.read('salidaCliente.txt')
246   puts "cargando calculo externo para el servidor..."
247   descifradoServidor = text = File.read('salidaServidor.txt')
248 else
249   puts "descifrando y computando el secreto compartido desde el cliente..."
250   descifradoCliente = (valorPublicoServidor ** secretoCliente) % moduloPrimo
251   puts "descifrando y computando el secreto compartido desde el servidor..."
252   descifradoServidor = (valorPublicoCliente ** secretoServidor) % moduloPrimo
253 end
254
255 #SE SIMULA EL FUNCIONAMIENTO DEL CLIENTE HABIENDO RECIBIDO TANTO EL VALOR PUBLICO COMO LA CADENA FIRMADA DESDE EL SERVIDOR
256 #SE CREA UN DIGEST CONCATENANDO LOS VALORES PUBLICOS DEL CLIENTE Y EL SERVIDOR, COMO EN EL PROCESO ANTERIOR, SOLO QUE AUN NO SE
257 #PROCEDE A FIRMAR
258 valoresClienteServidor = valorPublicoCliente.to_s + valorPublicoServidor.to_s
259 digestCliente = OpenSSL::Digest::SHA1.digest(valoresClienteServidor)
260 #VALIDACIÓN DEL CLIENTE: SI LA VALIDACIÓN DE LA FIRMA DEL SERVIDOR FALLA, SE ABORTA EL PROCESO
261 if dsaServidor.sysverify(digestCliente, firmaServidor) then
262   puts "validado servidor!! (Firma validada)"
263 else
264   puts "ERROR"
265   exit
266 end
267 #AQUI SE FIRMA CON LA LLAVE DSA PRIVADA DEL CLIENTE ALMACENADA ANTERIORMENTE, UNA CADENA CON LOS VALORES PUBLICOS DEL CLIENTE Y EL SERVIDOR
268 #PARA SER ENVIADA NUEVAMENTE AL SERVIDOR A MANERA DE HANDSHAKE
269 firmaCliente = dsaCliente.syssign(digestCliente)
```

Por último, el script realiza la impresión de los resultados, y del tiempo de ejecución del mismo



```
261     puts "validado servidor!! (Firma validada)"
262   else
263     puts "ERROR"
264     exit
265   end
266   #AQUI SE FIRMA CON LA LLAVE DSA PRIVADA DEL CLIENTE ALMACENADA ANTERIORMENTE, UNA CADENA CON LOS VALORES PUBLICOS DEL CLIENTE Y EL SERVIDOR
267   #PARA SER ENVIADA NUEVAMENTE AL SERVIDOR A MANERA DE HANDSHAKE
268   firmaCliente = dsaCliente.syssign(digestCliente)
269
270   #EL SERVIDOR VALIDA LA LLAVE DEL CLIENTE
271   if dsaCliente.sysverify(digestServidor, firmaCliente) then
272     puts "validado cliente!! (Firma validada)"
273   else
274     puts "ERROR DE VALIDACIÓN"
275     exit
276   end
277   #SE IMPRIMEN LOS VALORES CALCULADOS POR CADA PARTE (CLIENTE Y SERVIDOR)
278   puts "VALOR SECRETO CONOCIDO (CALCULO CLIENTE): " + descifradoCliente.to_s
279   puts "VALOR SECRETO CONOCIDO (CALCULO SERVIDOR): " + descifradoServidor.to_s
280
281
282   #FIN DIFFIE HELLMAN
283   finish = Time.now
284   puts "Tiempo de ejecución: " + (finish - start).to_s
285   ##-----
286   end
287 end
288 #ESTA LINEA PERMITE OBTENER LOS PARAMETROS DE LA LINEA DE COMANDOS PARA USAR CON OPTPARSE
289 options = Parser.parse(ARGV)
290 #FIN OPCIONES LINEA DE COMANDOS
291
292
```

Como un plus, o añadido, se muestra una captura del programa en java; cabe resaltar que, como fue desarrollado para servir de apoyo al script de ruby, este no posee ningún menú o ayuda, ya que solo está hecho para recibir parámetros sin banderas.


```
5 class DiffieSecreto {
6     public static void main(String[] args) {
7         try
8         {
9             //SE GENERAN LOS ARCHIVOS DONDE SERAN ALMACENADOS LOS RESULTADOS PARA EL CLIENTE Y EL SERVIDOR
10            //CABE DESTACAR QUE NO POSEE AYUDA YA QUE SE USA DE MANERA INTERNA CON EL SCRIPT DIFFIE-HELLMAN DESARROLLADO EN RUBY
11            FileWriter archivoCliente = new FileWriter("./salidaCliente.txt");
12            FileWriter archivoServidor = new FileWriter("./salidaServidor.txt");
13
14            //ESTA ES LA SECCIÓN DEL CLIENTE
15            //AQUI SE OBTIENE EL SECRETO DEL CLIENTE DE LA LINEA DE COMANDOS
16            int secretoCliente = Integer.parseInt(args[0]);
17            //SE OBTIENE EL VALOR PUBLICO DEL SERVIDOR
18            BigInteger valorPublico = new BigInteger(args[1]);
19            //SE OBTIENE EL MODULO PRIMO PARA AMBAS PARTES
20            BigInteger moduloPrimo = new BigInteger(args[2]);
21            //SE CALCULA EL VALOR PUBLICO DEL SERVIDOR ELEVADO A LA POTENCIA DEL SECRETO DEL CLIENTE
22            BigInteger potenciaCliente = valorPublico.pow(secretoCliente);
23            //SE CALCULA EL RESULTADO CALCULANDO EL MODULO DE LA OPERACION ANTERIOR FRENTE AL NUMERO PRIMO
24            BigInteger resultado = potenciaCliente.mod(moduloPrimo);
25            //SE ESCRIBE EL RESULTADO Y SE CIERRA EL ARCHIVO
26            archivoCliente.write(resultado.toString());
27            archivoCliente.close();
28            //ESTA ES LA SECCIÓN DEL SERVIDOR
29            //AQUI SE OBTIENE EL SECRETO DEL SERVIDOR DE LA LINEA DE COMANDOS
30            int secretoServidor = Integer.parseInt(args[3]);
31            //SE OBTIENE EL VALOR PUBLICO DEL CLIENTE
32            BigInteger valorPublico2 = new BigInteger(args[4]);
33            //SE CALCULA EL VALOR PUBLICO DEL CLIENTE ELEVADO A LA POTENCIA DEL SECRETO DEL SERVIDOR
34            BigInteger potenciaServidor = valorPublico2.pow(secretoServidor);
35            //SE CALCULA EL RESULTADO CALCULANDO EL MODULO DE LA OPERACION ANTERIOR FRENTE AL NUMERO PRIMO
36            BigInteger resultado2 = potenciaServidor.mod(moduloPrimo);
37            //SE ESCRIBE EL RESULTADO Y SE CIERRA EL ARCHIVO
38            archivoServidor.write(resultado2.toString());
39            archivoServidor.close();
40        }
41        catch(Exception ex){
42            ex.printStackTrace();
43        }
44    }
45 }
```

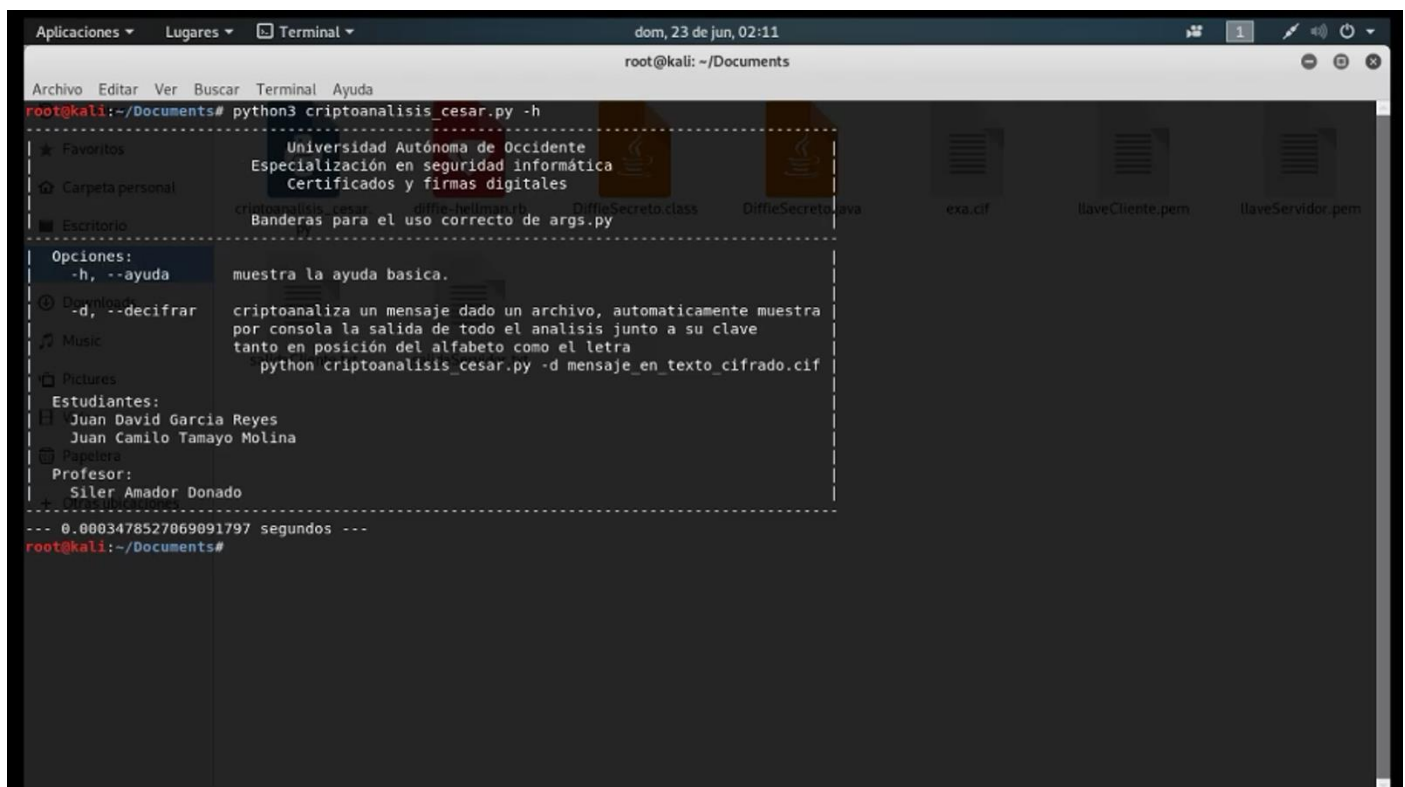
CRIPTOANÁLISIS DEL CIFRADO TIPO CESAR

para poder determinar la llave de un texto el cual asumimos que fue cifrado por medio del algoritmo de julio cesar, se hace uso del análisis de frecuencia estadístico, con el fin de encontrar la clave de transposición de este, y proceder a descifrarlo.

Para la ejecución del script se provee los siguientes parámetros o banderas:

- Un parámetro o bandera que desplegará la ayuda o determinará la clave con la que fue cifrado el texto que se le indique
 - -h para desplegar la ayuda.
 - -d para encontrar la clave o llave del texto a seleccionar
- La ruta o nombre de un archivo de texto cifrado.

A continuación, se realiza la ejecución del archivo con la bandera -h para así visualizar la ayuda que provee el script `criptoanalisis_cesar.py`



```
root@kali:~/Documents# python3 criptoanalisis_cesar.py -h
criptoanalisis_cesar
Banderas para el uso correcto de args.py

Opciones:
-h, --ayuda      muestra la ayuda basica.
-d, --decifrar   criptoanaliza un mensaje dado un archivo, automaticamente muestra
                  por consola la salida de todo el analisis junto a su clave
                  tanto en posición del alfabeto como el letra
                  python criptoanalisis_cesar.py -d mensaje_en_texto_cifrado.cif

--- 0.0003478527069091797 segundos ---
root@kali:~/Documents#
```

El programa comenzará el proceso pertinente, empieza a recorrer todas las letras del texto, extrae la cantidad de veces que se repiten, posteriormente ordena de mayor a menor las ocurrencias de letras.

```
def countAndSortLetters(encryptedMessage):
    lettersInMessage = []

    for indexLetterMessage, letterMessage in enumerate(encryptedMessage):
        isLetterInMessage = 0
        indexCounterLetterInMessage = 0
        for indexCounterLetter, counterLetter in enumerate(lettersInMessage):
            if letterMessage == counterLetter['letter']:
                isLetterInMessage = 1
                indexCounterLetterInMessage = indexCounterLetter
        if isLetterInMessage:
            lettersInMessage[indexCounterLetterInMessage]['quantity'] += 1
        else:
            lettersInMessage.append({'letter': letterMessage, 'quantity': 1})

    lettersInMessage = sorted(lettersInMessage, key = lambda i: i['quantity'], reverse=True)

    return lettersInMessage
```

luego procede a recorrer una lista o arreglo del alfabeto de frecuencias, es decir, de las letras que más se repiten en el idioma español e inglés, con el fin de obtener las hipótesis y realizar las operaciones pertinentes para poder verificar si el par de letras escogidas para las hipótesis son las correctas y así realizar el criptoanálisis. La explicación de lo que realiza el algoritmo automáticamente es la siguiente:

1.Hipótesis: Frecuencia de U: 114, luego probablemente $U \rightarrow e$, pero al hacer las pruebas no corresponde.

2.Hipótesis Frecuencia de Y: 100, luego probablemente $Y \rightarrow e$, $U \rightarrow a$

Recordamos la función de ciframiento: $(M_i + K) \bmod 27 = C_i$

Como la $Y \rightarrow e$ y la Y, E ocupan respectivamente las posiciones 25, 4 en el alfabeto, entonces reemplazamos los valores en la función: $(4 + K) \bmod 27 = 25$, hallamos una K que la suma modulo 27 con 4 me de como resultado 25, luego esa **$K = 21$** .

Verificamos la clave asumiendo que $U \rightarrow a$, luego $(0 + K') \bmod 27 = 21$, **$K' = K = 21$**

Aplicamos la función inversa $(C_i - K) \bmod 27 = M_i$ para descifrar el mensaje:

WJGYHTUMYKJMXYWCMNJVMYFJNXCUNSUIJNXYGCCHZUHWCULOYGCOHCW
JKYMNJHUDYCHJFPCXUVF...

Algoritmo automatizado:

En esta parte del código es donde cada palabra del mensaje, después de haber sido organizada de mayor a menor, es procesada para la realización de la hipótesis, tomando la cantidad de recurrencias y almacenándolas en las variables Kij, con el fin de encontrar un numero tal que K_{11} igual a K_{12} o K_{21} igual a K_{22} .

```

File Edit Selection View Go Debug Terminal Help
cripto3.py - Visual Studio Code

cripto3.py x
D: > SSD > Descargas > cripto3.py > decrypted
169 # ciclo para recorrer el alfabeto de frecuencia para el idioma español o ingles
170 for indexLetterInAlphabetFrequency, letterInAlphabetFrequency in enumerate(alphabetFrequency):
171     # validación para verificar que el índice de la frecuencia del alfabeto mas uno no vaya a superar el limite del arreglo
172     if indexLetterInAlphabetFrequency+1 < len(alphabetFrequency):
173         # inicialización de las hipotesis
174         K11 = 0
175         K12 = 0
176         K21 = 0
177         K22 = 0
178     # ciclo para recorrer las letras que se encuentran en el mensaje cifrado
179     for firstIndexLetterMessage, firstLetterMessage in enumerate(lettersInMessage):
180         # validación para verificar que el índice del mensaje cifrado mas uno no vaya a superar el limite del arreglo
181         if firstIndexLetterMessage+1 < len(lettersInMessage):
182             # ciclo para recorrer las letras que se encuentran en el mensaje cifrado por segunda vez para realizar
183             # la operación de las hipotesis
184             for secondIndexLetterMessage, secondLetterMessage in enumerate(lettersInMessage):
185                 # validación para verificar que el índice del mensaje cifrado mas el índice en que se encuentra el primer ciclo
186                 # del mensaje cifrado mas uno no vaya a superar el limite del arreglo
187                 if secondIndexLetterMessage+firstIndexLetterMessage+1 < len(lettersInMessage):
188                     # se obtienen el par de letras a las cuales se les va a realizar las hipotesis
189                     firtsLetter = lettersInMessage[firstIndexLetterMessage]
190                     secondLetter = lettersInMessage[secondIndexLetterMessage + firstIndexLetterMessage+1]
191                     # Se filtran los caracteres que no se encuentren en la tabla de frecuencias, para que encuentren su
192                     # índice correspondiente en el alfabeto
193                     if len(firtsLetter) < 3 or firtsLetter['letter'] != "\n":
194                         firtsLetter['index'] = alphabet.index(str(firtsLetter['letter']))
195                     if len(secondLetter) < 3 or secondLetter['letter'] != "\n":
196                         secondLetter['index'] = alphabet.index(str(secondLetter['letter']))
197                     # se realiza el calculo de las hipotesis para el par de letras, ejemplo par de letras A, B
198                     K11 = (firtsLetter['index'] - alphabetFrequency[indexLetterInAlphabetFrequency]['index']) % module
199                     K12 = (secondLetter['index'] - alphabetFrequency[indexLetterInAlphabetFrequency+1]['index']) % module
200                     # se realiza el calculo de las hipotesis para el par de letras de manera inversa,
201                     # ejemplo par de letras B, A
202                     K21 = (secondLetter['index'] - alphabetFrequency[indexLetterInAlphabetFrequency]['index']) % module
203                     K22 = (firtsLetter['index'] - alphabetFrequency[indexLetterInAlphabetFrequency+1]['index']) % module
204                     # validación en cascada si el resultado de las hipotesis 1 o 2 da el mismo resultado,
205                     # si esto pasa quiere decir que la llave utilizada para cifrado el mensaje fue encontrada
206                     if K11 == K12:
207                         break

```

```

File Edit Selection View Go Debug Terminal Help
cripto3.py - Visual Studio Code

cripto3.py x
D: > SSD > Descargas > cripto3.py > decrypted
206         if K11 == K12:
207             break
208         if K21 == K22:
209             break
210     if K11 == K12:
211         break
212     if K21 == K22:
213         break
214     if K11 == K12:
215         key = K11
216         break
217     if K21 == K22:
218         key = K21
219         break
220
221 for letterMessage in lettersInMessage:
222     cube=""
223     print(("letra "+letterMessage['letter']+" tiene " +
224           str(letterMessage['quantity'])+" repeticiones"))
225     for x in range(letterMessage['quantity']):
226         cube += ""[key]""
227     print(cube)
228     print("")
229     print(('La llave que se utilizo fue el numero: ' +
230           str(key)+' ', 'alphabet[key]'))
231
232
233 def messageError(indexArg, args):
234     print('-----')
235     print('| Universidad Autónoma de Occidente |')
236     print('| Especialización en seguridad informática |')
237     print('| Certificados y firmas digitales |')
238     print('| |')
239     print('| Error en el uso de las banderas, Por favor ver la ayuda ejecutando: |')
240     print('| python args.py -h |')
241     print('-----')
242
243
244 if __name__ == '__main__':

```

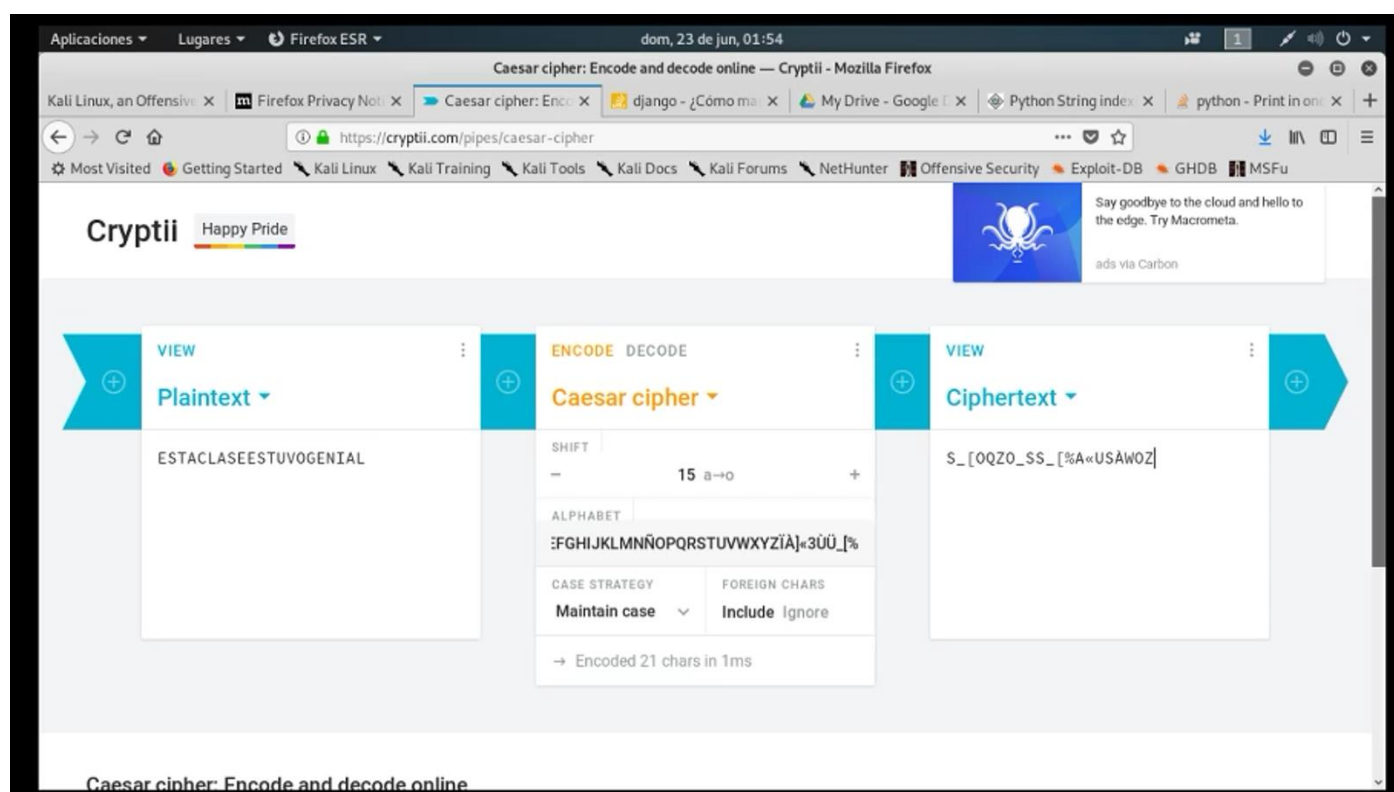
Esta parte del código desencadena a fuerza la finalización de los ciclos si y solo si las hipótesis en orden o inversas son iguales, ya que esto quiere decir que se encontró la clave con la cual fue cifrado en criptograma dado.

```
        if K11 == K12:
            break
        if K21 == K22:
            break

    if K11 == K12:
        break
    if K21 == K22:
        break

if K11 == K12:
    key = K11
    break
if K21 == K22:
    key = K21
    break
```

A continuación, se imprime a la consola el resultado del cálculo realizado, el criptograma utilizado corresponde a la palabra “ESTACLASEESTUVOGENIAL”. en esta imagen se muestra la recurrencia de las letras que conforman el criptograma, la llave o clave utilizada para cifrarlo y por último el tiempo de ejecución del script.



```
Aplicaciones ▾ Lugares ▾ Terminal ▾ dom, 23 de jun, 01:55 root@kali: ~/Documents
Archivo Editar Ver Buscar Terminal Ayuda
root@kali:~/Documents# python3 criptoanalisis_cesar.py -d exa.cif
exa.cif
letra S tiene 4 repeticiones
[ ] [ ] [ ] [ ] [ ]
letra _ tiene 3 repeticiones
[ ] [ ] [ ] [ ]
letra 0 tiene 3 repeticiones
[ ] [ ] [ ] [ ]
letra [ tiene 2 repeticiones
[ ] [ ] [ ]
letra Z tiene 2 repeticiones
[ ] [ ] [ ]
letra Q tiene 1 repeticiones
[ ] [ ]
letra % tiene 1 repeticiones
[ ] [ ]
letra A tiene 1 repeticiones
[ ] [ ]
letra « tiene 1 repeticiones
[ ] [ ]
letra U tiene 1 repeticiones
[ ] [ ]
letra À tiene 1 repeticiones
[ ] [ ]
letra W tiene 1 repeticiones
[ ] [ ]
letra
tiene 1 repeticiones
[ ] [ ]

La llave que se utilizo fue el numero: 15, 0
--- 0.0008778572082519531 segundos ---
root@kali:~/Documents#
```