FIRMAS Y CERTIFICADOS DIGITALES: PROYECTO FINAL

Juan Camilo Tamayo Molina

Juan David García Reyes

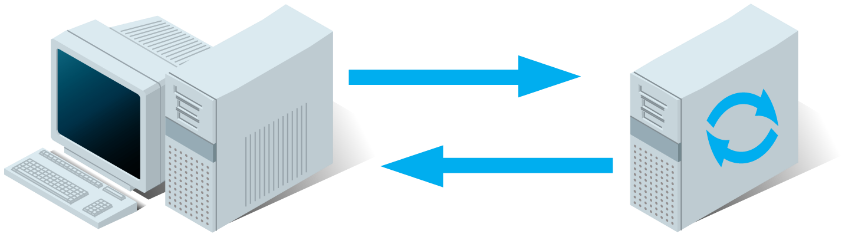
# resumen

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 Algoritm 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.

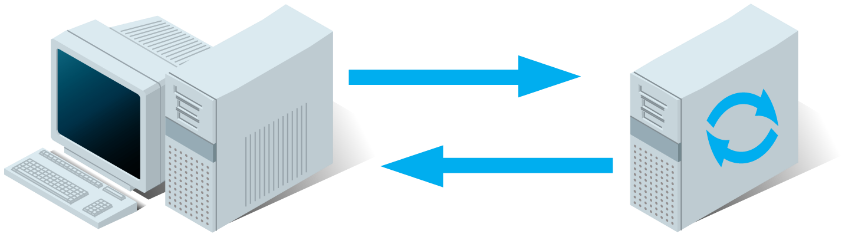
El servidor realizará el proceso de elegir un número privado "b", ya que los otros parámetros de carácter público ya fueron establecidos por el cliente.

|  |  |  |
| --- | --- | --- |
| 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 mod p | p = 17  g = 3 | b=772238  B = 2  B = g^b 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.



(FirmaB,2 )

(FirmaA,9 )

|  |  |  |
| --- | --- | --- |
| 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 fín de proceder a calcular la clave acordada cuyo cálculo se obtiene con la formula B^a mod p, en el caso del cliente, y A^b 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 mod p | p = 17  g = 3 | Cb = C(“92”)  b=772238  B = 2  Kb = 4  A^b 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) |

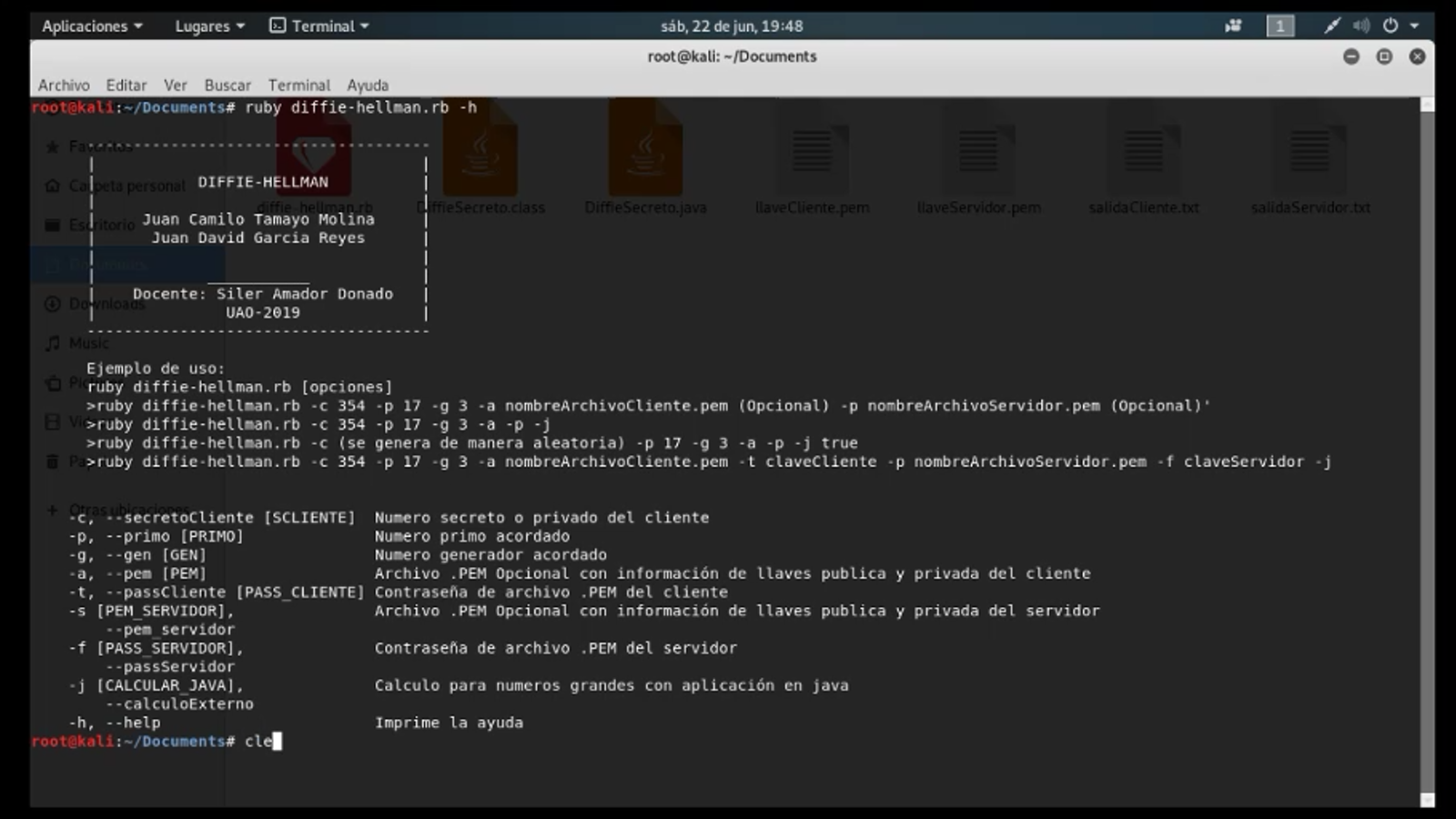
\*Va y Vb corresponden a la validación de las firmas por medio de una función valida a través de la llave pública del servidor si su digest generado de la cadena concatenada, concuerda con esta y con la firma recibida, El servidor realiza la misma tarea, tomando la llave pública del cliente y validando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

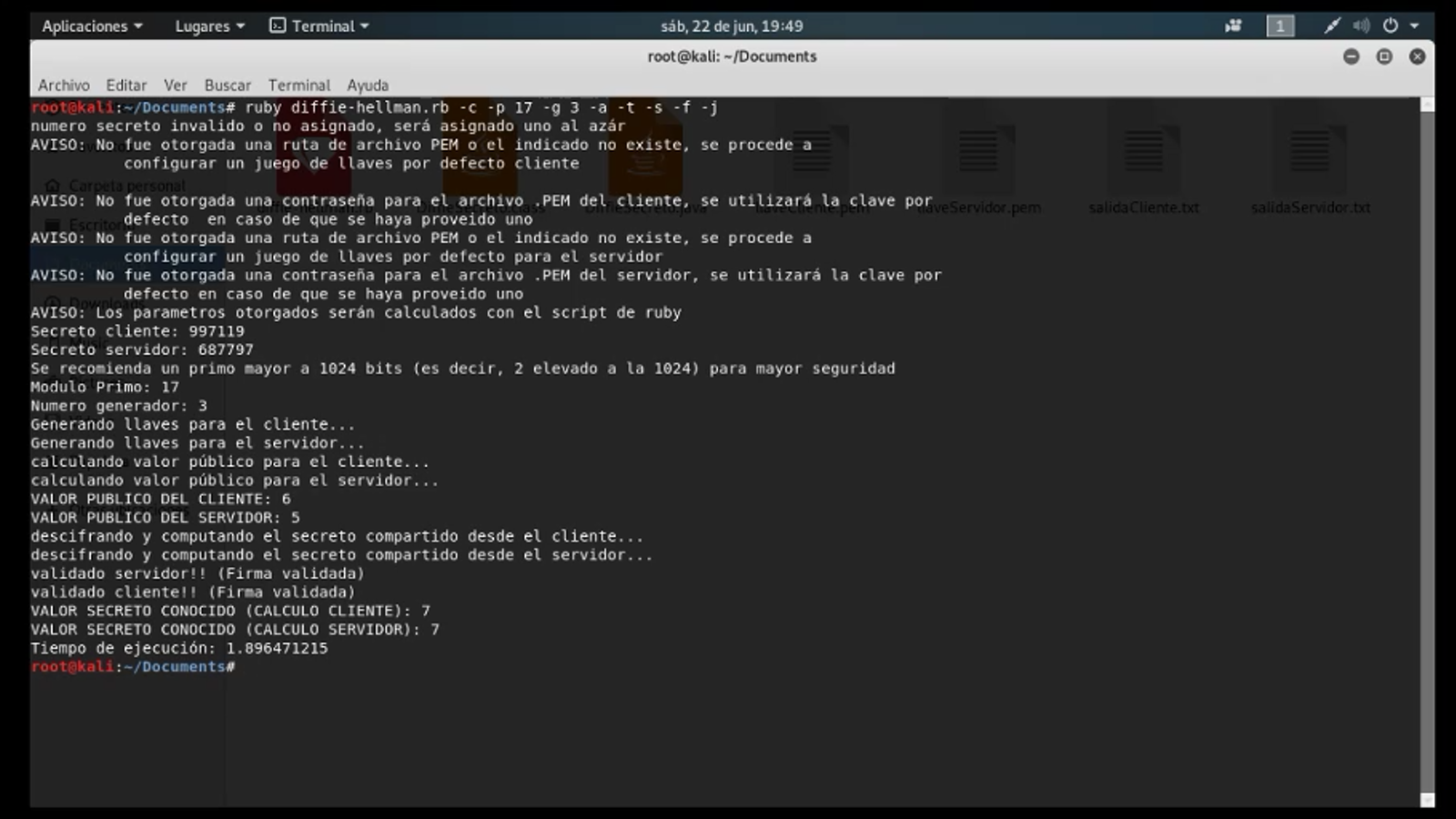
Al escribir el comando “ruby diffie-hellman.rb” junto a la bandera “-h” se nos mostrará la ayuda del script, con ejemplos pertinentes:



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

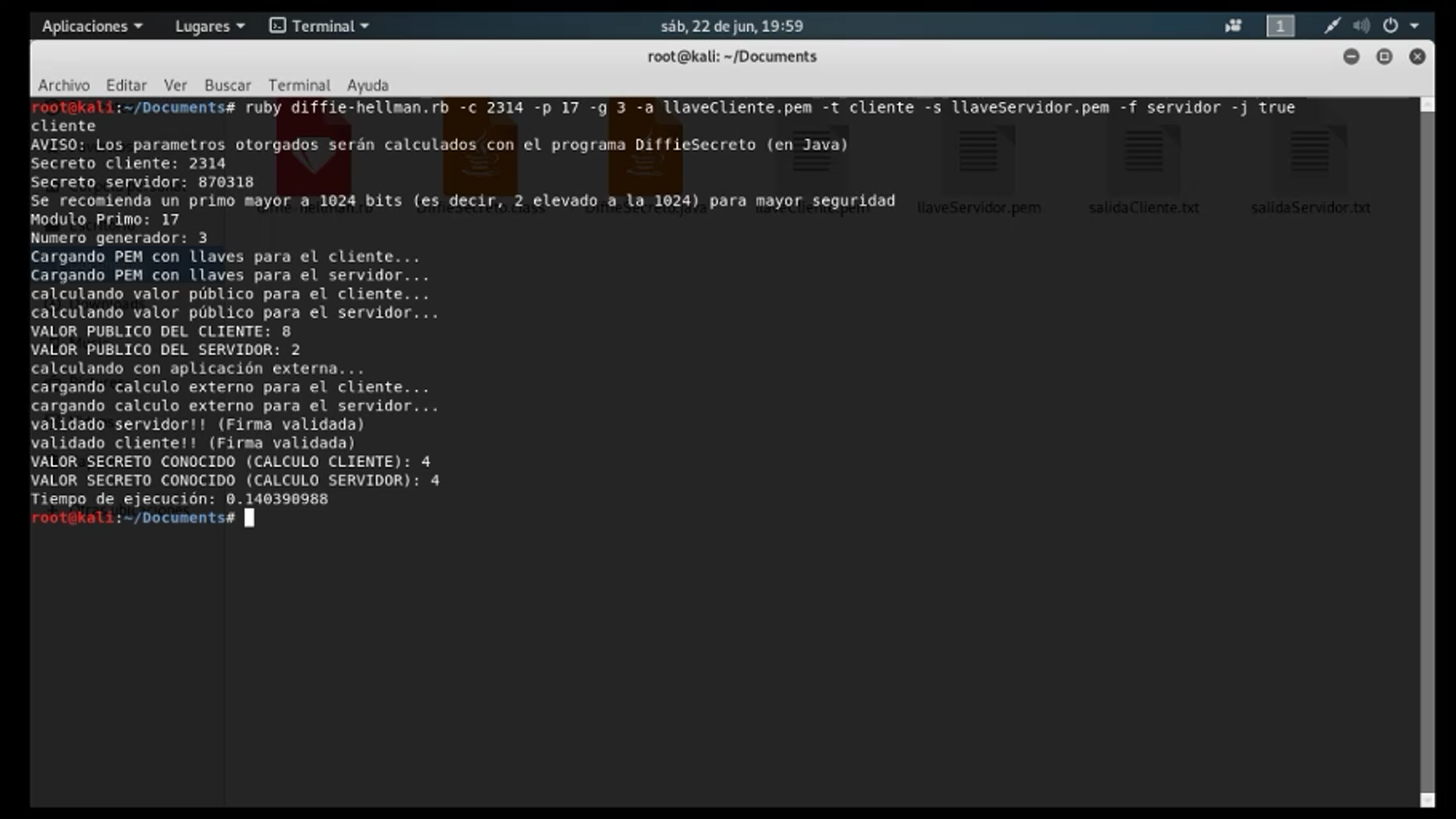
Al ejecutarlo se mostrará la siguiente pantalla:



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.



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

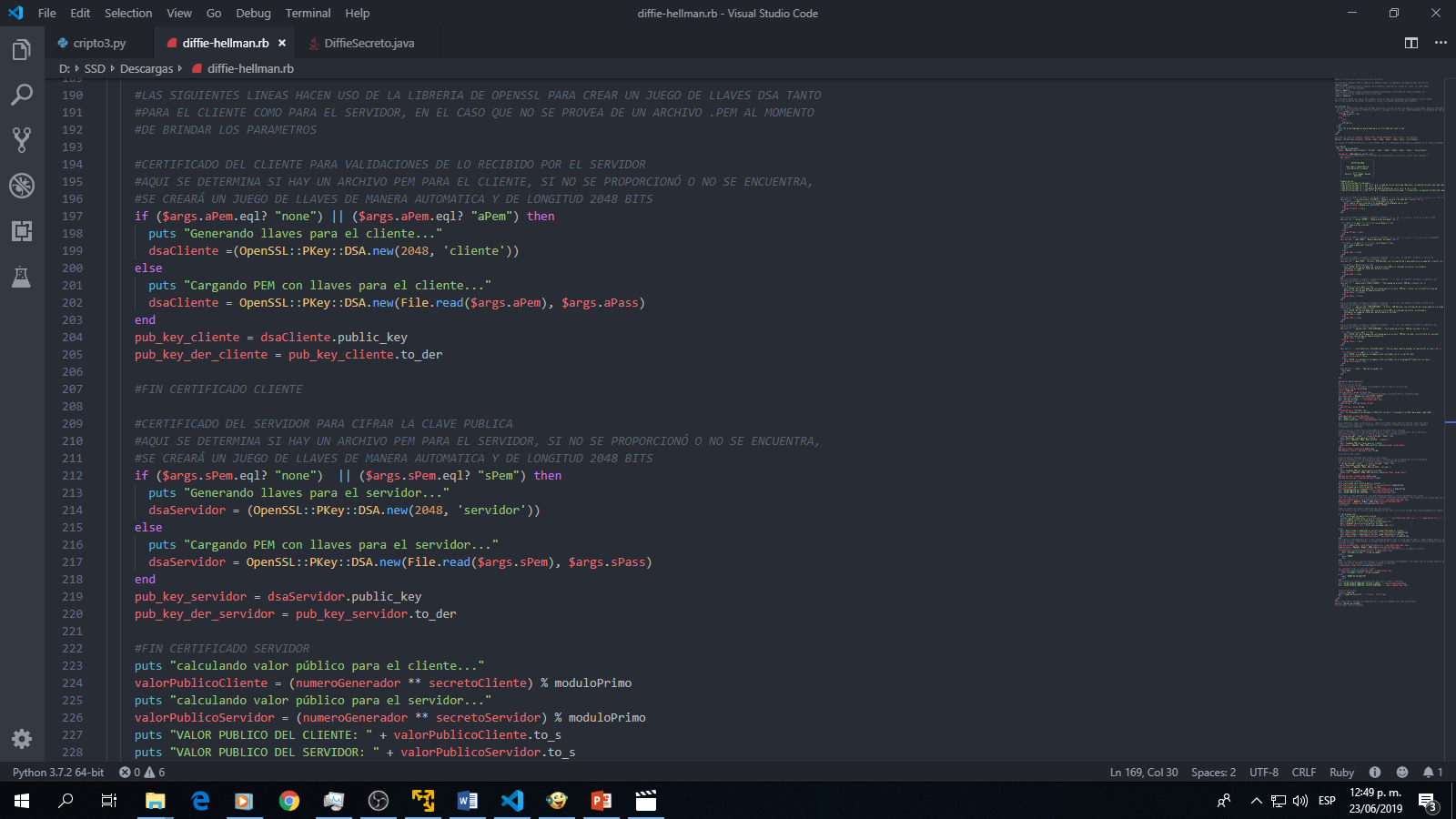
A continuación, se muestran los fragmentos principales del código. En la siguiente imagen se observa cómo se inicia el contador para determinar el tiempo de ejecución, que corresponde a la variable “start”, junto a la variable “calculoJava” que determinará si se calculará con el script de manera nativa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



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.



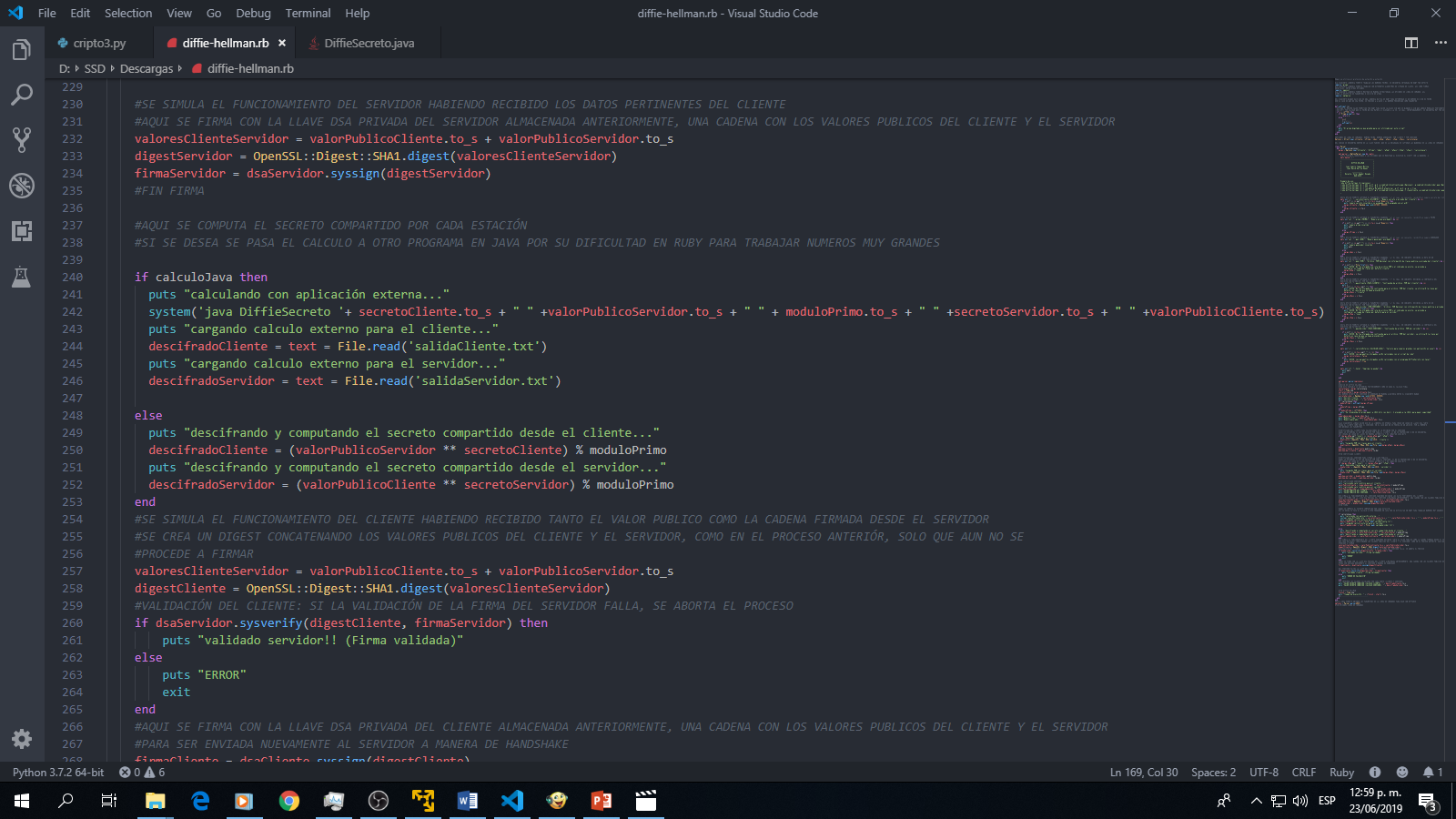
Como se menciona anteriormente, el script realiza una simulación entre la comunicación entre un cliente y un servidor, lo cual queda plasmado en la siguiente imagen, con sus comentarios respectivos.

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.

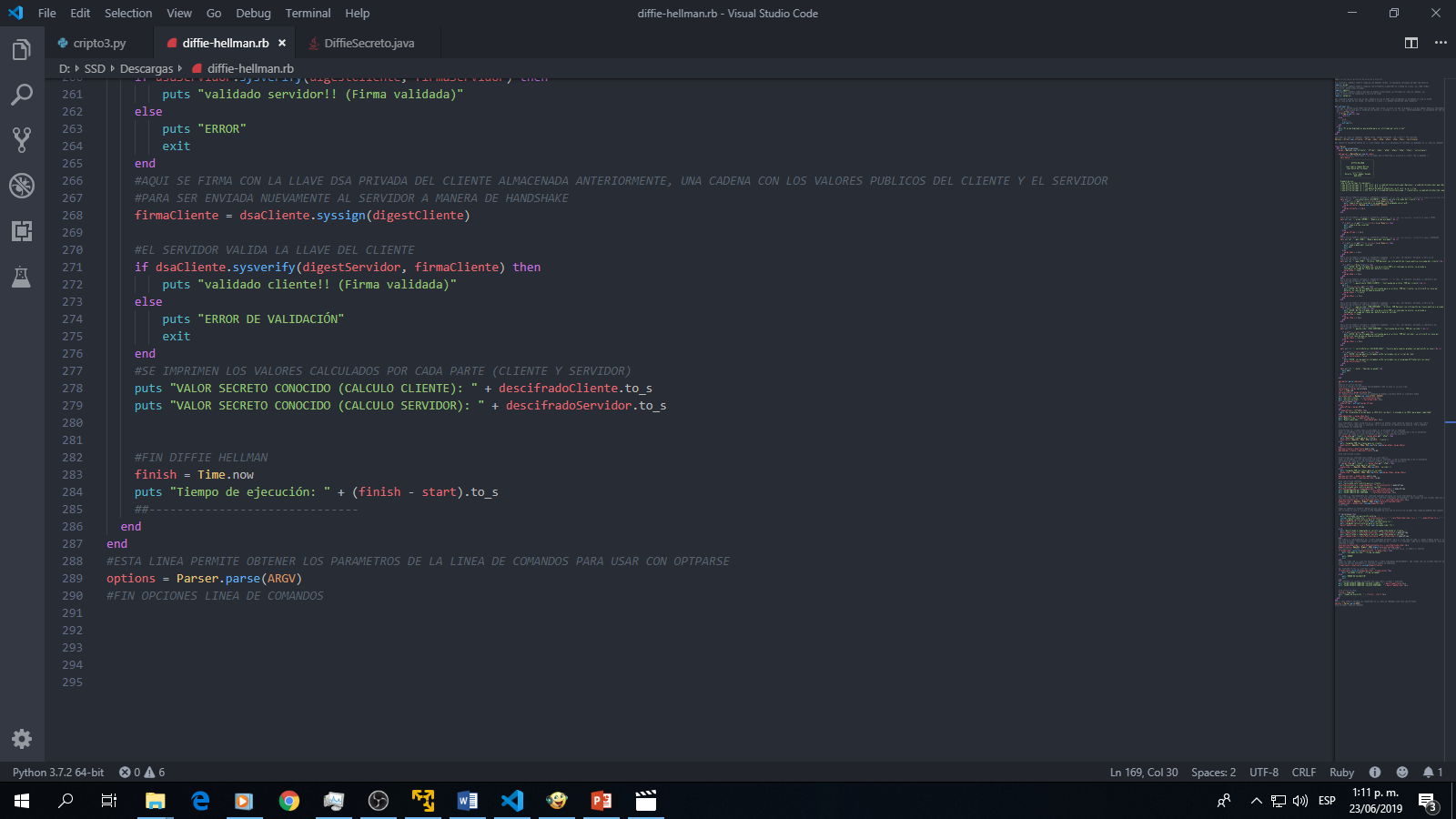
También aquí es donde se valida si enviar los parámetros al programa externo en java o simplemente dejar que el script en ruby los procese.

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.

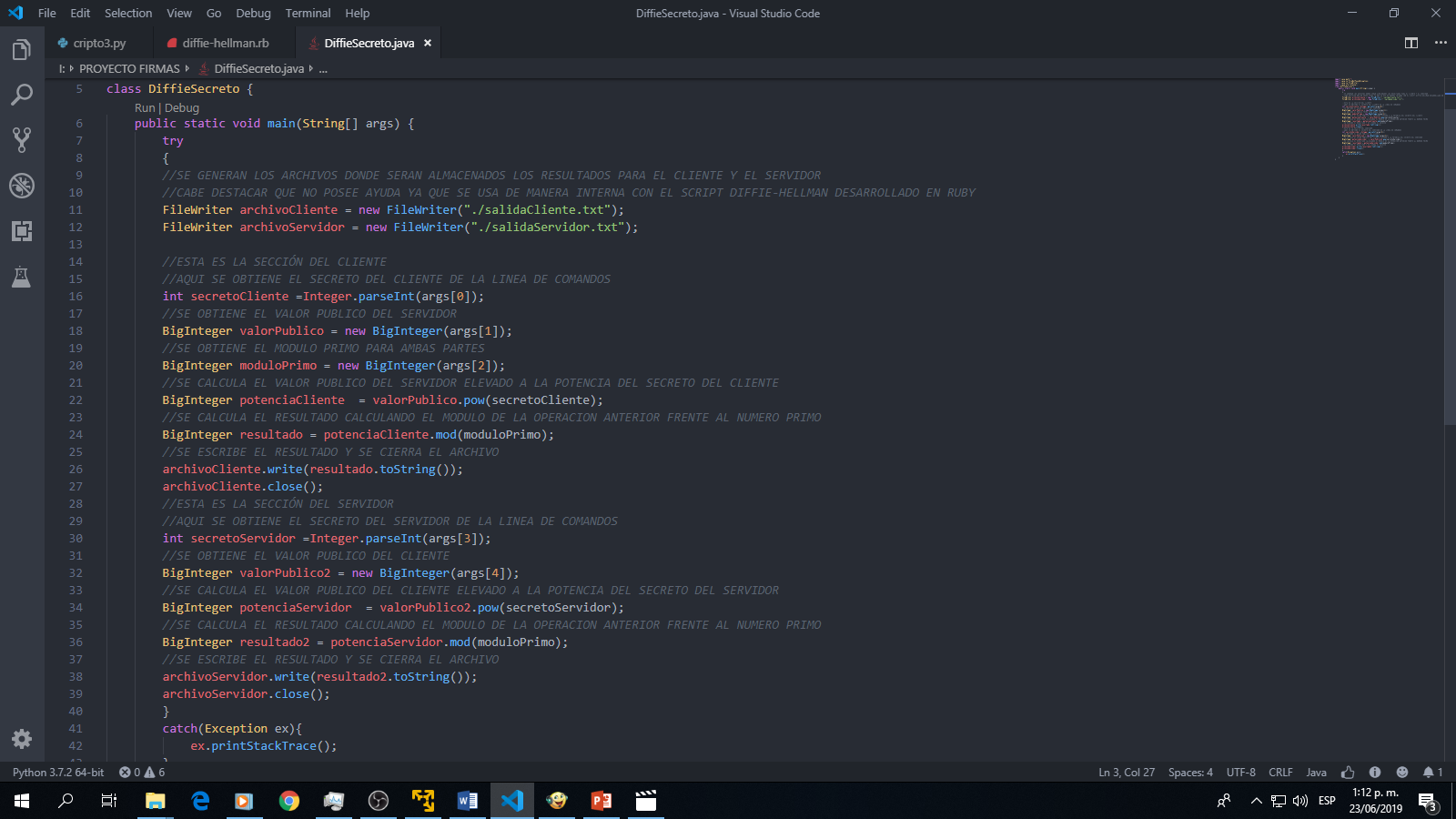
Este script se basa en la premisa de que cada terminal posee la llave publica del otro, con el fin de que se mantenga segura la comunicación, y de que estas no tengan que ser enviadas también por un canal inseguro.



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



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.



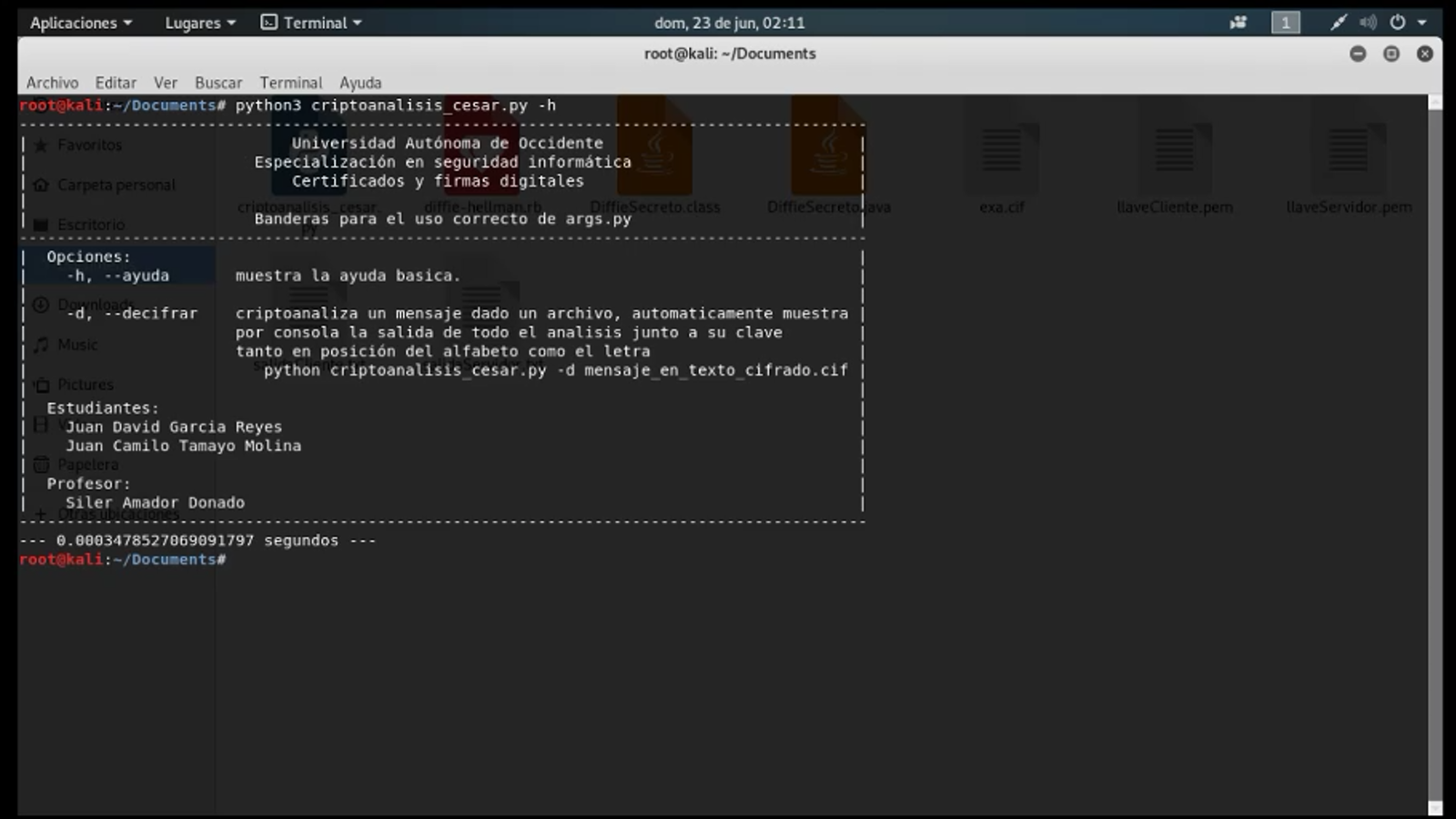
# Criptoanálisis del cifrado tipo cesar

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



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.

A screenshot of a cell phone

Description automatically generated

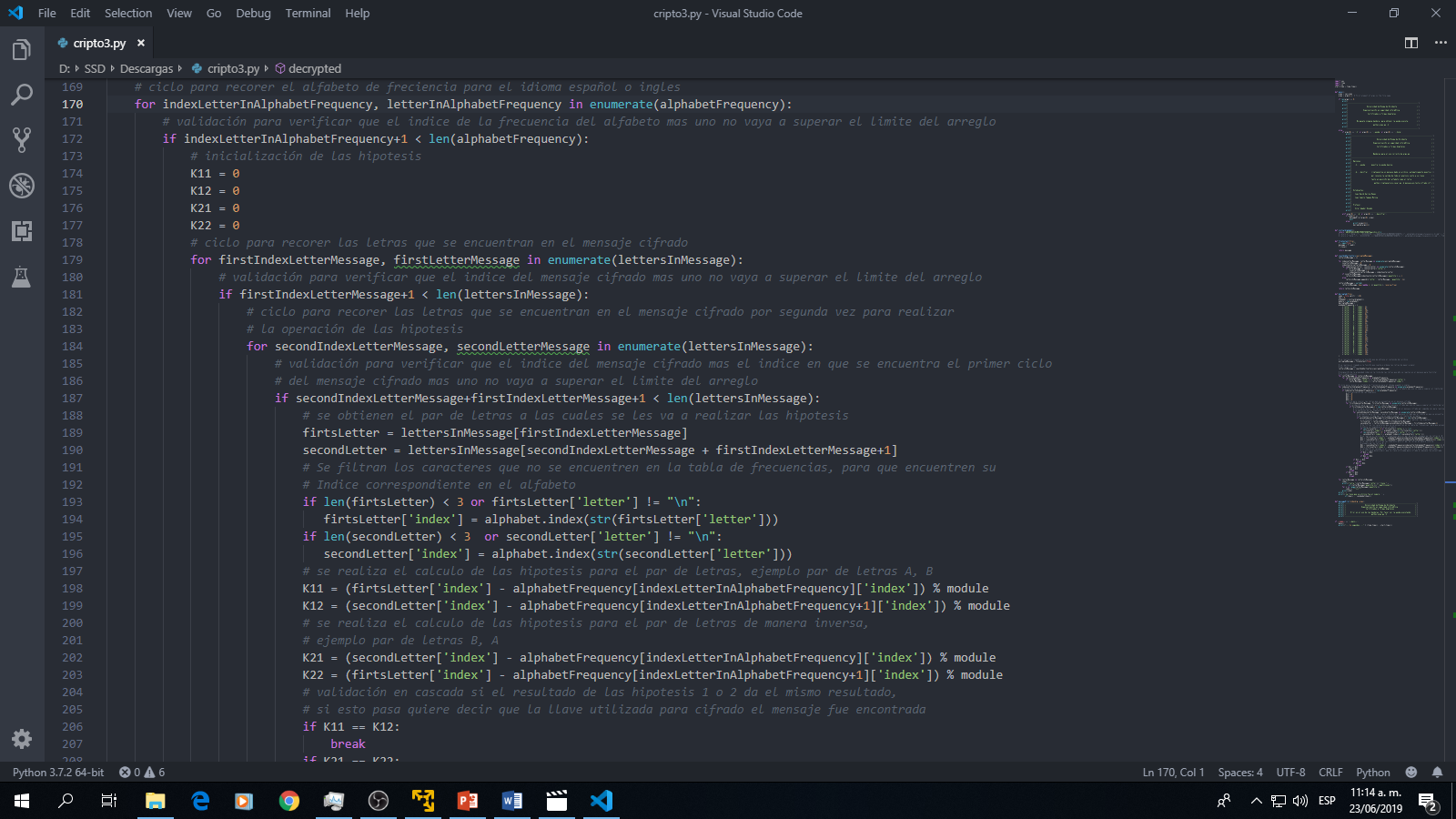
luego procede a recorrer una lista o arreglo del alfabeto de frecuencias, es decir, de las letras que más se repiten en el idioma español e inglés, con el fin de obtener las hipótesis y realizar las operaciones pertinentes para poder verificar si el par de letras escogidas para las hipótesis son las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

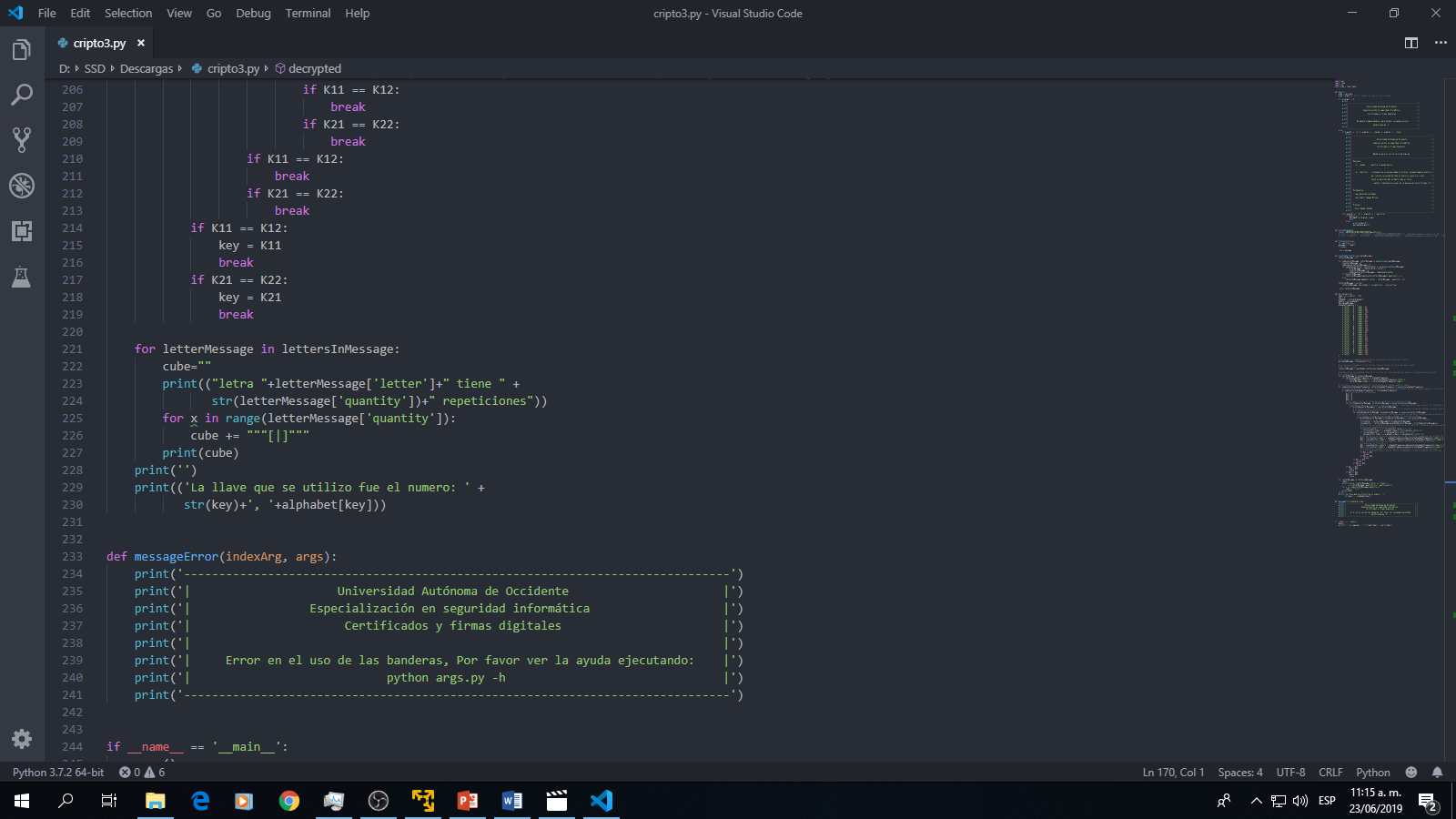
A screenshot of a cell phone

Description automatically generated

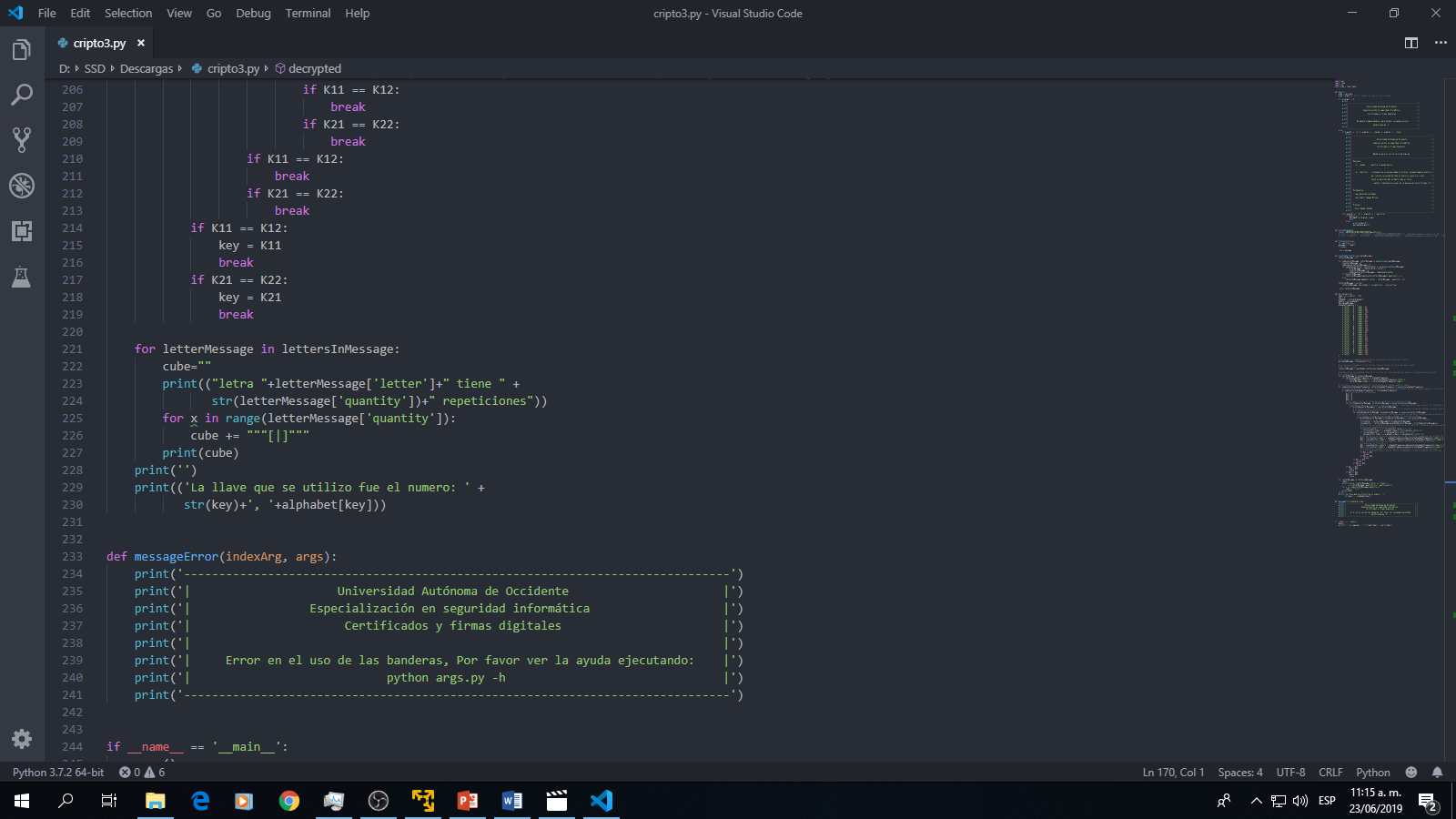
Algoritmo automatizado:

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 K11 igual a K12 o K21 igual a K22.





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.



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.

