Mateo López - 202220119

Maria Alejandra Londoño - 202220983

Infraestructura Computacional

Caso 3

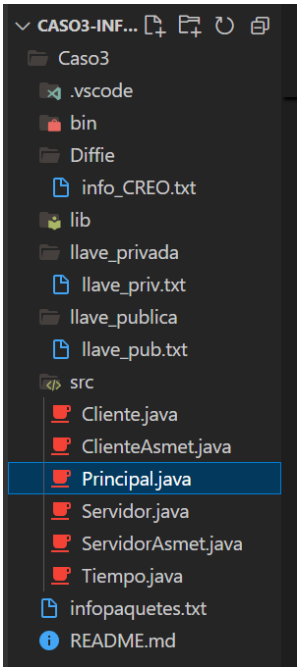
Organización de los archivos:

El proyecto tiene una organizacón como se ve en el primer punto de las reglas para ejecutar el programa, y los archivos relevantes para el caso son los siguientes:

- → Llave_privada: carpeta que contiene un archivo de texto con la llave privada generada al ejecutar la opción 1.
- → Llave_publica: carpeta que contiene un archivo de texto con la llave privada generada al ejecutar la opción 1.
- → Src: Carpeta que contiene la lógica del programa:
  - Cliente.java: Clase que simula el comportamiento completo de un cliente en la simulación propuesta, desde la conexión al socket del servidor hasta la recepción de la solicitud del estado de un paquete.
  - ClienteAsmet.java: Idéntico a la clase Cliente, creado para la ejecución de la opción "3" que busca medir el tiempo del cifrado del estado del paquete.
  - Principal.java: Clase principal que contiene el método main(), y crea los delegados de los clientes y el objeto del Servidor.
  - Servidor.java: Encargado de habilitar el socket de comunicación con el cliente. Cuando recibe uno, delega todo el proceso solicitado para la simulación a una clase interna llamada ManejadorCliente.
  - ServidorAsmet.java: Igual que ClienteAsmet, realiza las mismas funciones que la clase original del servidor, pero también intenta cifrar el estado del paquete usando la llave privada propia y toma el tiempo que se demora haciendolo.
  - Tiempo.java: Clase utilizada para medir el Delta de tiempo entre los procesos que necesitan una medición de tiempo.
- → Infopaquetes.txt: Archivo de texto que contiene las 32 entradas de información sobre los paquetes, genera la tabla predefinida de información de paquetes que el Servidor contiene según el enunciado.

Reglas para ejecutar el programa:

- Posterior a descomprimir la carpeta del envío, abra en (preferiblemente) Visual Studio Code la carpeta Caso3-Infracomp, al hacerlo debería verse de esta manera:

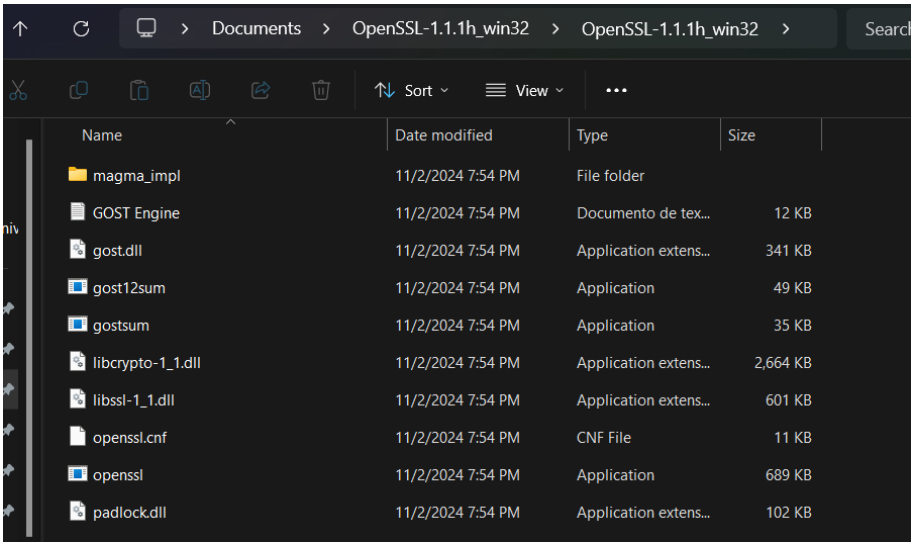


Es extremadamente importante que aparezca la carpeta Caso3 y dentro de ella esté contenido todo el proyecto, para que las rutas relativas manejadas por las clases funcionen correctamente.

- Una vez se tenga lista la carpeta del proyecto, se ejecutará desde la clase Principal.java. Lo primero que se pedirá es la ruta al ejecutable de openssl, la cual debería verse similar a esta ruta:

  `C:\Users\57304\Documents\OpenSSL-1.1.1h_win32\OpenSSL-1.1.1h_win32`

  Esta ruta deberá llevar a una carpeta similar a la que se verá a continuación, lo más importante siendo que exista el archivo openssl.exe:

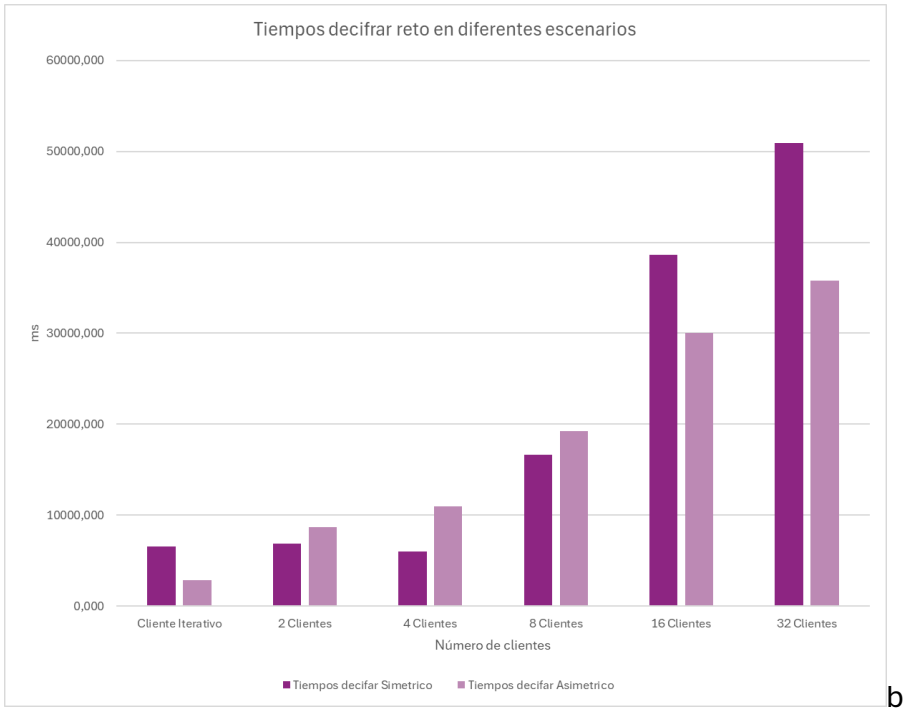


- Una vez se ingresa la ruta, se presentarán 3 opciones: realizar la opción 1, realizar la opción 2 y ejecutar la opción 2 de nuevo pero con el escenario de cifrar con la llave privada el estado del paquete:
  - Si se ejecuta la opción 1, se guardarán la llaves pública y la privada en las carpetas `llave_publica` y `llave_privada`, respectivamente

- Si se ejecuta la opción 2, se pedirá ingresar el número de clientes concurrentes que se desean crear. En caso de que se cree solo un cliente, este va a realizar 32 consultas, si se escribe cualquier otro número x>1, se crearán x clientes y cada uno realizará una sola consulta. Hay unos prints que se realizan por cada conexión cliente servidor para notificar que el procedimiento se está realizando correctamente, además de algunos temporizadores que marcan cuanto se demoró el proceso de generación de G, P y G^x, verificación de la consulta del cliente, el tiempo que tomó cifrar el estado del paquete con la llave simétrica de la sesión y el tiempo total que tomó ejecutar la opción 2 completa.
- Si se ejecuta la opción 3, se hará exactamente el mismo procedimiento que la opción pasada, lo único que cambia es que toma el tiempo que se demoró el servidor en cifrar el estado del paquete con su llave privada, para realizar la comparación posterior.

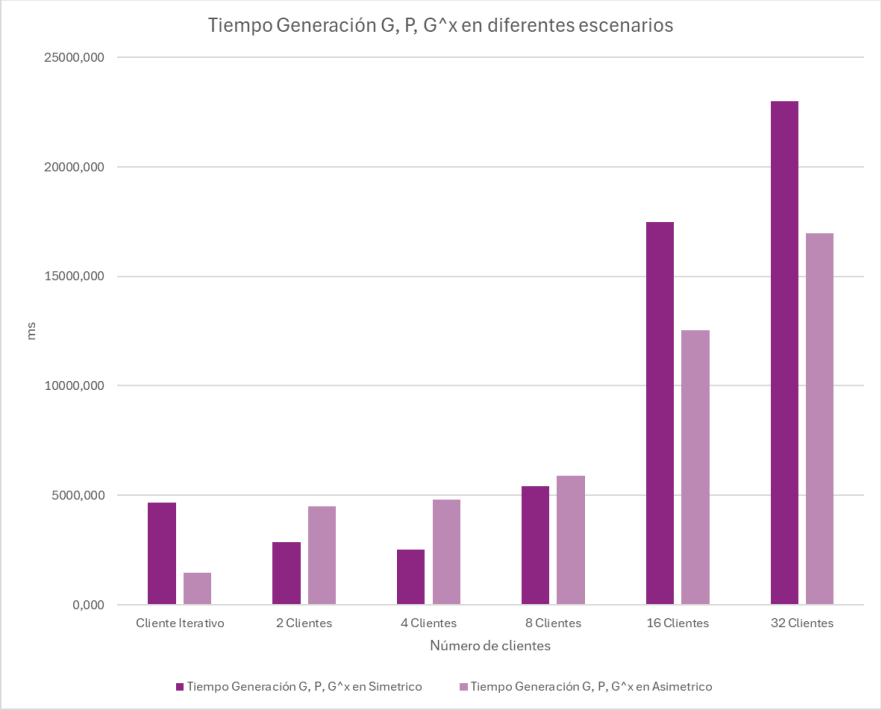
2. Las tablas con los datos tomados para las gráficas se encuentran en el siguiente excel

Graficas:

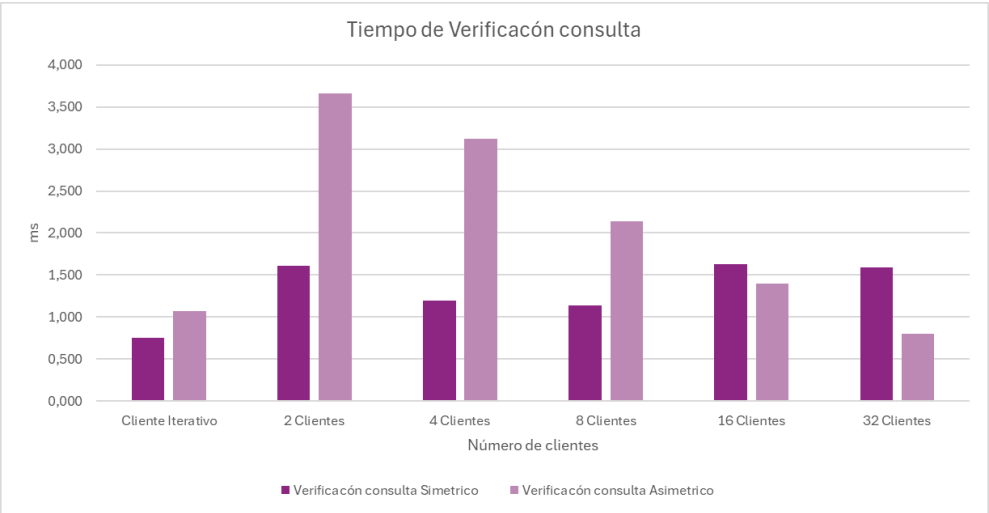


b

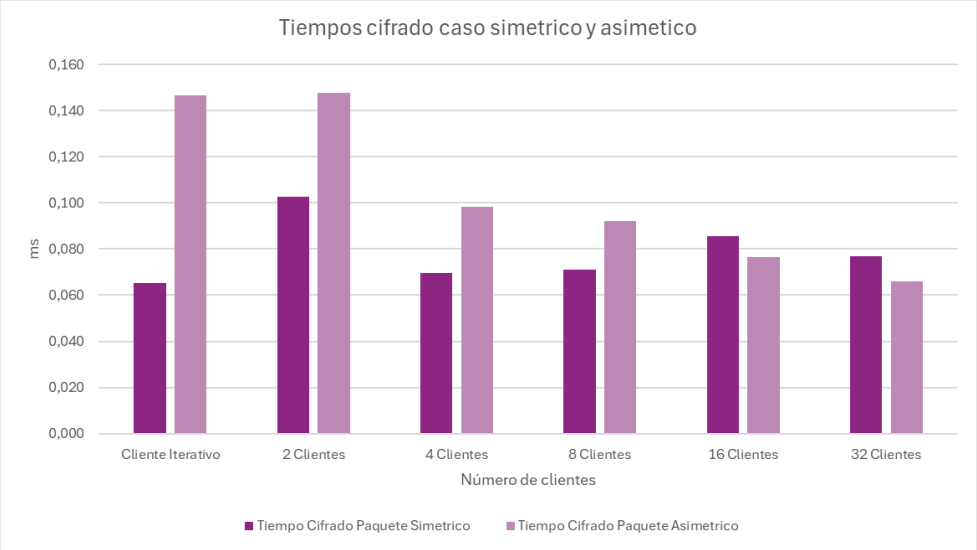
Al aumentar el número de clientes concurrentes, los tiempos de descifrado incrementan notablemente, especialmente para el cifrado simétrico, a pesar de que el caso asimétrico requiera de más recursos computacionales es posible que, al alcanzar un cierto umbral de clientes, el sistema priorice o administre de forma más eficiente las solicitudes, mejorando la eficiencia en cargas altas para el cifrado asimétrico. En el caso de un cliente iterativo, los tiempos para ambos cifrados son relativamente bajos en comparación con los escenarios de múltiples clientes y se puede ver que en el caso asimétrico se tarda más debido a más operaciones que se deben lograr para este caso.



La gráfica muestra que los tiempos de generación de G, P, y G^x aumentan con el número de clientes en ambos tipos de cifrado, siendo más altos en el cifrado simétrico que en el asimétrico. Aunque el cifrado simétrico suele ser más rápido, como ya fue mencionado anteriormente, en este caso enfrenta tiempos de generación mayores, especialmente en escenarios de alta concurrencia. En contraste, el cifrado asimétrico parece manejar mejor la carga concurrente, posiblemente gracias a optimizaciones en su implementación que permiten distribuir la generación de forma más eficiente.



La gráfica muestra que el cifrado simétrico tiene un mejor desempeño en términos de tiempo de verificación de consulta al incrementarse el número de clientes concurrentes, casi se mantiene constante con todos los escenarios de prueba. Con el caso asimétrico se puede ver cómo se disminuye considerablemente el tiempo de verificación, esto puede ser debido al uso de cache ya que en todos los casos se observa como la primera consulta siempre tarda un tiempo notablemente más grande mientras que el resto toman muchísimo menos, lo anterior explica esa disminución puesto que se promediaron los tiempos.



Para el tiempo de cifrado se puede ver que, en comparación con el resto de las medidas tomadas, los tiempos en ms son prácticamente constantes lo que indica que el proceso de cifrado en sí no se ve significativamente afectado por el incremento en el número de clientes. Se puede ver, igual que en la gráfica anterior, que los tiempos del caso asimétrico tienden a disminuir lo que igualmente puede ser explicado con el uso de chaché.

Cálculos de operaciones por segundo para cifrado y descifrado simetrico:

Para la toma de muestras se utilizó un computador cuyo procesador es un Intel i7 11370H de 11va generación. Este procesador tiene una velocidad de 3.3Ghz, que es lo mismo a $3.3 * 2^{30} Hz = 3.543 * 10^9$ ciclos por segundo.

Según la página de cifrado por AES de Wikipedia, este algoritmo toma 18 ciclos por byte.

Dado que el algoritmo de AES usado utiliza PKCS5Padding, el cual separa el mensaje a cifrar en bloques de 8 bytes cada uno, se puede concluir que se necesitan 18 * 8 = 144 ciclos por bloque a cifrar. En un segundo entonces se podrían cifrar alrededor de $3.543 * 10^9$ / 144 = 24,604,000 bloques en un segundo.

Si se asume que se necesitan 4 ciclos por instrucción, esto lleva a 144/4 = 36 instrucciones por bloque, entonces al multiplicar el número de bloques por segundo * instrucciones por bloque, se puede llegar a la conclusión de que en un segundo se pueden ejecutar alrededor de 885,750,000 instrucciones de cifrado simétrico por segundo

Cálculos de operaciones por segundo para cifrado y descifrado asimétrico:

Para la toma de muestras se utilizó el mismo computador y procesador Intel i7 11370H de 11va generación. Se utilizará la misma velocidad de $3.543 * 10^9$ ciclos por segundo.

Usando el comando speed de openssl, podemos ver una prueba de benchmark realizada en este computador para saber cuántas operaciones de cifrado con la llave privada (firma) y verificación con la llave pública se pueden hacer en un segundo:

```
C:\Users\57304>openssl speed rsa1024
Doing 1024 bits private rsa's for 10s: 123413 1024 bits private RSA's in 8.98s
Doing 1024 bits public rsa's for 10s: 1694996 1024 bits public RSA's in 8.89s
OpenSSL 1.1.1t  7 Feb 2023
built on: Wed Feb  8 13:37:49 2023 UTC
options:bn(64,64) rc4(8x,int) des(int) aes(partial) blowfish(ptr)
compiler: gcc  -march=nocona -msahf -mtune=generic -O2 -pipe -DTERMIOS -DL_ENDIAN -DOPENSSL_PIC -DOPENSSL_CPUID_OBJ -DOP
ENSSL_IA32_SSE2 -DOPENSSL_BN_ASM_MONT -DOPENSSL_BN_ASM_MONT5 -DOPENSSL_BN_ASM_GF2m -DSHA1_ASM -DSHA256_ASM -DSHA512_ASM
-DKECCAK1600_ASM -DRC4_ASM -DMD5_ASM -DAESNI_ASM -DVPAES_ASM -DGHASH_ASM -DECP_NISTZ256_ASM -DX25519_ASM -DPOLY1305_ASM
-DZLIB -DNDEBUG
                  sign    verify    sign/s verify/s
rsa 1024 bits 0.000073s 0.000005s  13737.0 190641.8
```

Dadas estas pruebas, se puede identificar que se pueden hacer 13,737 operaciones con la llave privada por segundo y 190,641 operaciones con la llave pública por segundo.

Recursos utilizados:

- https://sourceforge.net/projects/openssl-for-windows/
- https://www.baeldung.com/java-rsa
- https://www.ibm.com/docs/en/sdk-java-technology/8?topic=interfaces-signature-class
- https://www.baeldung.com/java-hmac