**Universidad de los Andes**

**Caso 3**

**Infraestructura Computacional**

**ISIS-2203**

**Kevin Arenas Ospina - 202110673**

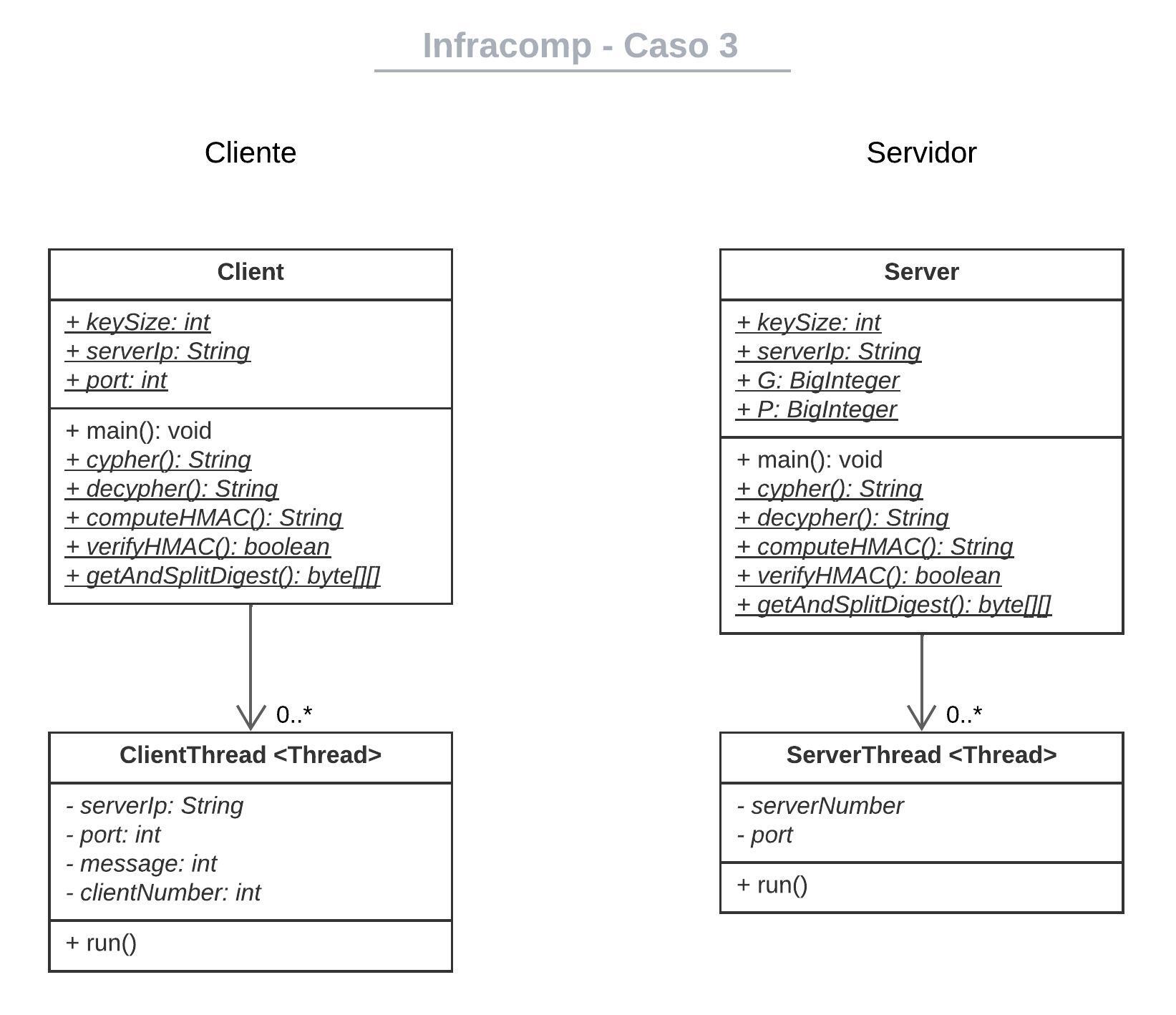
**Juan David Castillo Quiroga - 202210669**

**Camilo Andres Morillo Cervantes - 202015224**

**Mayo 5 de**

**2024-1**

**UML:**



**Descripción de organización de archivos en el zip:**

El archivo extraíble contiene el proyecto con la carpeta docs donde se encuentra la documentación del proyecto y la carpeta src donde se encuentra el código fuente. Dicho código fuente cuenta de 4 archivos correspondientes a las 4 clases implementadas: Client, ClientThread, Server y ServerThread. Los archivos Client y Server contienen el metodo main para realizar la ejecución correspondiente del cliente y del servidor.

**Funcionamiento General del Sistema:**

Tanto el funcionamiento del cliente como del servidor se basa en una clase principal que despliega Threads encargados de la conexión. Para ello dicha clase principal por medio de su método main crea los Threads y los empieza a ejecutar pasándoles el puerto de conexión y en caso del cliente también la IP del servidor. Además, en las clases principales se tienen múltiples atributos estáticos a los cuales pueden acceder los Threads. También se tienen los métodos de cifrado, descifrado, HMAC y Digest de forma estática en las clases principales de forma que los hilos puedan ejecutar estos accediendo a ellos de forma estática. Una vez se crea un hilo de cliente y de servidor estos dos proceden a ejecutarse siguiendo el protocolo de comunicación segura establecido.

**Instrucciones para correr el servidor – cliente:**

Con tal de ejecutar la aplicación en primer lugar es necesario que el servidor se esté corriendo para esto se ejecuta el archivo Server.java el cual contiene el método main que ejecuta el servidor. Posteriormente se procede a ejecutar el cliente, al correr el archivo Client.java que contiene el correspondiente método main del cliente. Una vez se ejecuta el cliente a través de la terminal se selecciona la opción de establecer conexión con el servidor al ingresar la opción ‘1’, posteriormente se solicita el número de clientes concurrentes a desplegar, una vez se ingresa este número se solicita el mensaje a enviar al servidor el cual debe ser el numero al cual se le restara 1. De esta forma se procede a realizar la comunicación cliente-servidor donde tanto en la terminal del cliente como en la del servidor se mostrarán los pasos realizados, así como los tiempos de ejecución de los pasos importantes como son el cifrado o cálculos realizados.

**Respuestas a las preguntas:**

**i)** **En el protocolo descrito el cliente conoce la llave pública del servidor (K\_w+). ¿Cuál es el método comúnmente usado para obtener estas llaves públicas para comunicarse con servidores web?**

El método comúnmente utilizado para obtener la llave pública es el intercambio de certificados SSL/TLS. Básicamente, cuando el cliente accede al servidor HTTPS, el servidor le envía un certificado digital, incluyendo su llave pública. Esto facilita que el cliente verifique la veracidad de la fuente, ya que se verifica la firma digital a través de una entidad certificadora. Después de la verificación, el cliente ya es libre de utilizar la clave pública para establecer una conexión segura.

**ii) ¿Por qué es necesario cifrar G y P con la llave privada?**

Al cifrar G y P con la llave privada, se firma digitalmente esta información. Esto significa que solo el servidor que posee la llave privada correspondiente puede cifrar estos valores. Cuando el cliente recibe los valores cifrados, se puede verificar de forma asertiva la autenticidad del servidor al descifrarlos con la llave pública del servidor y compararlos con los valores originales de G y P. Al momento que se identifique que los valores descifrados coinciden con los originales, el cliente asegura que la información es enviada por el servidor. De esta forma, se protege contra la manipulación maliciosa de estos valores durante la transmisión del mensaje.

**iii) El protocolo Diffie-Hellman garantiza “Forward Secrecy”, presente un caso en el contexto del sistema Banner de la Universidad donde sería útil tener esta garantía, justifique su respuesta (por qué es útil en ese caso).**

- Supongamos que un estudiante accede al sistema Banner para realizar cambios en su inscripción de materias y acceder a su información personal, como calificaciones y horarios. Si el sistema Banner utiliza el protocolo Diffie-Hellman para establecer la comunicación segura entre el navegador del estudiante y los servidores de la universidad, entonces se garantiza que incluso si un atacante intercepta y registra toda la comunicación entre el navegador del estudiante y los servidores en un momento dado, no podrá descifrar la información intercambiada, incluso si en el futuro obtiene la clave privada del servidor.

Esto es crucial porque, en el entorno presentado, la información personal y el acceso a la misma del estudiante, permite afectar información como calificaciones, datos de identificación y registros financieros, esto debe ser protegido de forma sólida y confiable para cada persona en la universidad.

**Tabla de resultados:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Tiempo en nanosegundos | Calcular G^y%P | Cifrar la consulta | Generar el código de autenticación | Descifrar la consulta | Verificar el código de autenticación |
| Escenario 1 (4 clientes | 1005900 | 501400 | 498700 | 498700 | 496500 |
| Escenario 2 (16 clientes | 1500300 | 501100 | 498900 | 503400 | 501600 |
| Escenario 3 (32 clientes | 2000100 | 502400 | 496500 | 2995700 | 502000 |

**Comentarios sobre resultados:**

En general, el aumento de clientes parece no afectar significativamente el tiempo de ejecución, incluso en casos como el de generar el código, disminuyó en la prueba de 32 clientes. Por otro lado, al momento de descifrar la consulta, si existe un cambio drástico en la ejecución con 32 clientes, casi 6 veces más tiempo de ejecución. Nuestra hipótesis en ese caso es que hay un punto de inflexión o un límite en el que el número de clientes afecta masivamente el tiempo de descifrar la consulta, por lo que es probable que a partir de ahí el tiempo de ejecución aumente de forma exponencial. Otra cosa que notamos fue que en el cálculo de G el aumento de tiempo de ejecución fue proporcional, a pesar de que el aumento de clientes no lo fue, por lo que pensamos que entre más clientes hay aumenta el tiempo, pero aumenta cada vez más lento en proporción al aumento inicial. Con las otras secciones no notamos cambios prácticamente, por lo tanto, podemos determinar que tiene un funcionamiento estable y regular incluso en situaciones con un mayor estrés de clientes.

**Cálculos finales:**

Para finalizar, tomamos los datos entregados, para poder realizar el cálculo de la estimación de velocidad del procesador. De esta forma, tomamos como caso base, 50 clientes.

**Cifrar:**

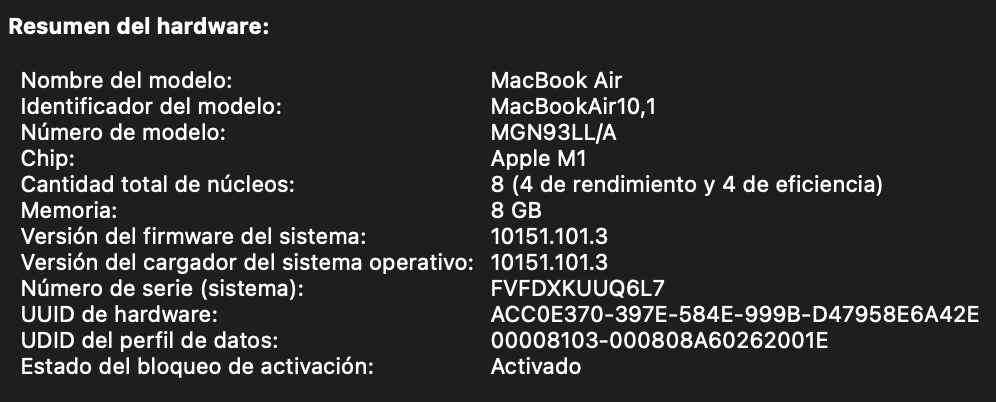
* **Tiempo de respuesta:** 170000 ns (nanosegundos)
* Operaciones por segundo: 1000000/170000 5,88 operaciones por segundo

**Generar código de autenticación:**

* **Tiempo de respuesta: 42000 ns (nanosegundos)**
* **Operaciones por segundo: 1000000/42000 23,81 operaciones por segundo**

**Especificaciones del procesador:**

Para nuestro caso, al ser MacBook, es complejo encontrar las especificaciones del procesador donde nos muestren el número de procesadores lógicos, por tal motivo tomamos este valor como un valor estimado y promedio, el cual suponemos como 12.



**Tareas por número de procesadores en el operador:**

**Cifrar:**

* **Operaciones totales = 5,88\*12 = 70,56 operaciones por segundo**

**Generar código de autenticación:**

* **Operaciones totales = 23,81\*12 = 285,72 operaciones por segundo**

**Bibliografía:**

Cryptography and network security, W. Stallings, Ed. Prentice Hall, 2003.

Computer Networks. Andrew S. Tanenbaum. Cuarta edición. Prentice Hall 2003, Caps 7, 8

Java Security Standard Algorithm Names, Oracle Help Center