**Universidad de Los Andes**

**Infraestructura Computacional**

**Caso 3**

**Alejandro Ahogado Prieto – 201920701**

**Juan Alejandro Charry Gavilan – 201923748**

* **Cada grupo debe entregar un zip con:**
  + **archivos con la implementación del prototipo y los archivos de las llaves del servidor.**
  + **un subdirectorio docs con un informe que incluya: (i) la descripción de la organización de los archivos en el zip, (ii) las instrucciones para correr el prototipo incluyendo cómo correr el servidor y cómo correr los clientes de forma concurrente, (iii) descripción del esquema que siguieron para generación de las llaves y nombres de los archivos que las almacenan, (iv) las respuestas a las tareas 1 a 5.**
* **Recuerde incluir en su informe todas las referencias que use para resolver este proyecto.**

1. **Descripción de la organización de archivos en el zip**
2. **Instrucciones para correr el servidor y los clientes**

Para correr el prototipo realizado es necesario ejecutarlo simultáneamente en 2 terminales, al iniciar el programa le solicitará indicar si quiere iniciar como cliente o como servidor, luego de esto se debe indicar el número de clientes que se espera recibir (en el caso del servidor) y el número de clientes que se van a ejecutar (en el caso del cliente) este número debe ser igual en ambos casos. Por último se debe indicar en ambos casos el escenario que se va a ejecutar (este número debe ser igual en ambos casos). Cabe resaltar que es necesario iniciar primero el servidor que el modo cliente. Después se solicitará por consola el nombre del cliente que quiere consultar el estado del paquete y el id del paquete (para manejar esta información se hace uso de un archivo .csv).

1. **Descripción del esquema que siguieron para la generación de las llaves y nombres de los archivos que las almacenan**

Para la generación de las llaves asimétricas se hizo uso de la clase KeyPairGenerator y se tomó como algoritmo “RSA”, de igual manera se inicializo en 1024, lo que correspondería a un tamaño de 1024 bits para las llaves. Después de esto, a partir del objeto inicializado de tipo KeyPairGenerator se hizo uso del método .generateKeyPair(), obteniendo como resultado un objeto de tipo KeyPair al que se le puede pedir la llave publica o privada. Para cifrar y descifrar haciendo uso de estas llaves se uso la clase Cipher, con la que se creo un objeto con el algoritmo “RSA”. El envió de la llave publica se realizó haciendo uso de un objeto ObjectOutputStream inicializado con el .getOutPutStream del socket y luego en el lado del cliente se recibía y se casteaba a PublicKey.

Para la generación de la llave simétrica se hizo uso de la clase KeyGenerator y se tomo como algoritmo”AES”, luego de esto este objeto se inicializo con el tamaño de la llave en bits (256) y también con un objeto de tipo SecureRandom. Finalmente, a partir del objeto inicializado se hizo uso del método .generateKey() y se guardo en un objeto de tipo SecretKey. El envío de esta llave se realizo al igual que con la llave publica por medio de un objeto de tipo ObjectOutpotStream, solo que en este caso esta iba cifrada con la llave publica del servidor para garantizar que solo este pueda conocerla. Para cifrar y descifrar con esta llave se usó la clase Cipher, con la que se creo un objeto con el algoritmo “AES/CBC/PKCS5Padding” y luego este se inicializo con el respectivo modo encriptar/desencriptar, la llave simétrica y el objeto de tipo IvParameterSpec.

1. **Construya una tabla con los datos recopilados. Tenga en cuenta que necesitará correr cada escenario en más de una ocasión para validar los resultados.**

**Tabla

Descripción generada automáticamente**

**Datos usados:**

**Tabla

Descripción generada automáticamente**

1. **Construya dos gráficas: una que muestre los tiempos para el caso simétrico (en cada uno de los escenarios) y una para el caso asimétrico (en cada uno de los escenarios).**
2. **Escriba sus comentarios sobre las gráficas, explicando los comportamientos observados.**

A partir de las anteriores gráficas se puede evidenciar que el cifrado simétrico tiene tiempos de ejecución menores al cifrado asimétrico, por lo que en este caso al intentarse cifrar un mensaje (el reto) que no es extenso va a ser siempre mucho más rápido.

1. **Identifique la velocidad de su procesador, y estime cuántos retos puede cifrar su máquina por segundo, en el caso evaluado de cifrado simétrico y cuántos en el caso evaluado de cifrado asimétrico. Escriba todos sus cálculos**

Tabla

Descripción generada automáticamenteTras hacer uso de los 32 datos recogidos en las pruebas de cifrado simétrico y asimétrico, obteniendo un promedio de cifrado del reto de 0.1577ms para el simétrico y de 3.3439ms para el asimétrico. Podría afirmar que bajo las mismas condiciones si en simétrico se cifra un reto en 0.1577ms, podría cifrar aproximadamente 6341 retos en 1 segundo y si en asimétrico se cifra un reto en 3.3439ms, podría decir que se puede cifrar aproximadamente 299 retos en 1 segundo por este cifrado. Los cálculos se realizaron por medio de una regla de tres.

**Diagrama de clases:**

**Escala de tiempo

Descripción generada automáticamente**

**Referencias:**

* Jenkov, J. (2022). Java KeyGenerator. Retrieved 12 May 2022, from <https://jenkov.com/tutorials/java-cryptography/keygenerator.html#:~:text=The%20Java%20KeyGenerator%20class%20%28javax.crypto.KeyGenerator%29%20is%20used%20to,show%20you%20how%20to%20generate%20symmetric%20encryption%20keys.?msclkid=31d3f4a0d06e11ec89279e5c66355dbb>
* Symmetric Encryption Cryptography in Java - GeeksforGeeks. (2022). Retrieved 12 May 2022, from <https://www.geeksforgeeks.org/symmetric-encryption-cryptography-in-java/?msclkid=e3891b2bd0be11ec90d31be3be0da9ee>
* Encrypting using AES-256, c., Pornin, T., & Bodewes, M. (2022). Encrypting using AES-256, can I use 256 bits IV?. Retrieved 12 May 2022, from <https://security.stackexchange.com/questions/90848/encrypting-using-aes-256-can-i-use-256-bits-iv#:~:text=ECB%20mode%20of%20course%20does%20not%20require%20an,Rijndael%20is%20not%20included%20in%20the%20standard%20runtime.?msclkid=bbe8998bd0ce11ec8e3ef387dfca8e17>
* Using HMac Sha256 for Message Authentication (MAC) in Java | Novixys Software Dev Blog. (2022). Retrieved 12 May 2022, from <https://www.novixys.com/blog/hmac-sha256-message-authentication-mac-java/#:~:text=Using%20HMac%20Sha256%20for%20Message%20Authentication%20%28MAC%29%20in,in%20this%20world%2C%20and%20it%E2%80%99s%20worth%20fighting%20for.%E2%80%9D?msclkid=9767b920d0e211ec8323fad7ae2fd498>
* Criptografía: Algoritmos de Digestión de Mensajes - Línea de Código. (2022). Retrieved 12 May 2022, from <https://lineadecodigo.com/java/criptografia-algoritmos-de-digestion-de-mensajes/?msclkid=fc451583d0e711ecbe78fb54f28ab0e2>