Seguridad

Caso de Estudio 3 - Infraestructura Computacional

Juan Andrés Romero - 202013449

Juan Alegría - 202011282

Organización de los archivos

En un panorama general, esta es la distribución de los archivos en el proyecto:

```
docs/
src/app/
	client/
		Client.java
	repeater/
		Repeater.java
	security/
		keys/
			asymmetric/
				clients/
				repeater/
				server/
			symmetric/
				clients/
				repeater/
				server/
		Asymmetric.java
		Keys.java
		Symmetric.java
	server/
		Server.java
		mensajes.txt
	utils/
		MultipleClients.java
		Termination.java
	App.java
```

Para el desarrollo de este proyecto, se realizaron 6 programas de Java dentro del mismo paquete ‘app’. En la raíz del proyecto podrá encontrar un README.md el cual indica cómo se puede ejecutar cada uno de los programas a través de un comando de terminal. Además, se

cuenta con una carpeta ‘docs’ en la que está este mismo informe que se está leyendo.

Luego, en la carpeta ‘src’ podrá encontrar el paquete ‘app’ e inmediatamente el programa ‘App.java’, el cual tiene el objetivo de facilitar la ejecución de todos los demás programas a través de ejecuciones de procesos concurrentes. Además, cuando se ejecuta ‘App.java’ se permite la ejecución de múltiples clientes que iniciarán al mismo tiempo en diferentes threads, en los que los outputs de cada uno serán indicados con el número del thread correspondiente.

Ahora bien, para la ejecución individual de cada uno de los componentes restantes, se crearon los siguientes programas, que si desea se pueden ejecutar por separado en diferentes terminales, e incluso se recomienda para que se visualice claramente la interacción a través de sockets que se tiene entre los componentes. Todos los componentes antes de iniciar cargan las llaves necesarias de acuerdo al tipo de ejecución indicada por el usuario, asimétricas o simétricas.

- ‘client/Client.java’ representando a un cliente que intentará conectarse a un repetidor y enviarle un ID de mensaje para finalmente recibir el contenido de ese mensaje.
- ‘repeater/Repeater.java’ representando al repetidor que siempre está escuchando en el socket del puerto 27700, si se tiene problemas al ejecutar porque el puerto está ocupado, se recomienda cambiar la variable port tanto en la clase Repeater como en Server. Cuando se inicia una solicitud de conexión en este socket, se creará un thread con un repetidor delegado el cual recibe el identificador del cliente, lo valida, y luego enviará un “OK” al cliente confirmando la conexión. Luego, recibirá el ID de mensaje requerido por parte del cliente, y con esto podrá solicitar el contenido del mensaje al servidor. Una vez se reciba el mensaje del servidor, el repetidor delegado lo comunicará al cliente, cerrará su conexión y el socket del repetidor continuará a la espera de nuevas conexiones.

- ‘server/Server.java’ representando al servidor que siempre está escuchando en el socket del puerto 42600, si se tiene problemas al ejecutar porque el puerto está ocupado, se recomienda cambiar la variable port tanto en la clase Server como en Repeater. Al iniciar este componente, se cargarán los mensajes del archivo ‘server/mensajes.txt’ el cual se podría llegar a interpretar como una base de datos simplificada de un servidor. Cuando se inicia una solicitud de conexión en este socket, se creará un thread con un servidor delegado, se leerá el ID de mensaje recibido y en base a eso se enviará el mensaje correspondiente a la solicitud realizada por algún repetidor.
- ‘security/Asymmetric.java’ programa que genera una llave asimétrica de acuerdo al tipo de componente. Si es un cliente la generará acompañada de un identificador.
- ‘security/Symmetric.java’ programa que genera una llave simétrica de acuerdo al tipo de componente. Si es un cliente la generará acompañada de un identificador.

Cuando se ejecutan por separado estos programas, podrá encontrar más logs en cada uno cuando ocurre un evento (como que el repetidor recibió un mensaje exitosamente del servidor o la información del proceso de generación de una llave).

Aparte de los 6 programas explicados anteriormente, se crearon librerías y directorios extra para el correcto funcionamiento del prototipo:

- ‘security/Keys.java’ es una librería interna para ejecutar operaciones de seguridad, como por ejemplo cifrar o descifrar un mensaje.
- ‘security/keys’ es un directorio encargado de almacenar las llaves generadas para que así sean utilizadas por cada uno de los componentes.
- ‘utils/Termination.java’ se encarga de cerrar e informar la finalización de un socket exitosamente.

- ‘utils/MultipleClients.java’ es utilizado para ejecutara varios clientes al mismo tiempo y generar un promedio acumulado de los tiempos de ejecución para así tener un análisis de los resultados más exacto.

Instrucciones de ejecución del prototipo

Prototipos de comunicación ejecutándose:

1. Abra una terminal en la raíz de este proyecto `Caso3-InfraComp/`.
2. Ejecute ‘App.java’ o use `java -cp ./bin app.App` para ejecutar todos los componentes. Esto le preguntará el tipo de encriptación para todas las siguientes comunicaciones e iniciará el número deseado de clientes, repetidor y servidor. Para cada cliente, se le pedirá que ingrese su ID de cliente y el ID de mensaje que están solicitando. Luego, el programa ejecutará diferentes procesos para generar automáticamente todas las claves necesarias para la comunicación. Finalmente, podrá ver el mensaje recibido por cada cliente seguido de un indicador de hilo.

Si desea visualizar individualmente los componentes, ejecute los siguientes comandos en diferentes terminales:

1. Asegúrese de que se generen todas las claves de cliente, servidor y repetidor que se utilizarán. Si necesita crear nuevas claves, siga las instrucciones en la sección de generación de claves.
2. Para ejecutar el servidor, utilice el tipo `java -cp ./bin app.server.Server`.
3. Para ejecutar el repetidor, utilice `java -cp ./bin app.repeater.Repeater type`.
4. Para ejecutar un solo cliente, utilice `java -cp ./bin app.client.Client tipo clientID messageID`.

- Tipos de cifrado válidos: SIMÉTRICO o ASIMÉTRICO. El tipo elegido debe ser el mismo para todos los componentes de una ejecución.
- El clientID es un número entero.
- El messageID es un número entero entre 00 y 09.

Generación de claves:

Si necesita generar una nueva clave, ejecute los siguientes comandos:

1. Para cifrado simétrico: `java -cp ./bin app.security.Symmetric type [id]`.
2. Para cifrado asimétrico: `java -cp ./bin app.security.Asymmetric type [id]`.

- Los tipos válidos son client, repeater o server.
- La identificación es un número entero y solo se aplica a las claves del cliente.

Esquema de generación de las llaves y nombres de los archivos que las almacenan

Las llaves se organizaron en el subdirectorio 'security/keys' de la siguiente manera:

```
keys/
	asymmetric/
		clients/
			Client{{ID}}Key.key
			Client{{ID}}Key.pub
		repeater/
			RepeaterKey.key
			RepeaterKey.pub
		server/
			ServerKey.key
			ServerKey.pub
	symmetric/
		clients/
			Client{{ID}}Key.key
		repeater/
			RepeaterKey.key
		server/
			ServerKey.key
```

Las llaves se generan automáticamente de acuerdo a su necesidad en todo el prototipo de comunicación y los clientes usados, si y sólo si se ejecuta el prototipo desde App.java. De lo

contrario, si se va a ejecutar cada programa de los componentes por separado, se debe ejecutar igualmente los programas de generación de llaves y asegurarse de que estén todas las llaves que van a ser utilizadas durante el flujo completo de comunicación.

Llaves simétricas:

Para generar las llaves simétricas, el programa ‘security/Symmetric.java’ le preguntará el tipo de llave que desea, para un servidor, repetidor o cliente. La llave que genera este prototipo se genera usando el algoritmo AES, modo ECB, y un esquema de relleno PKCS5 con una llave de 128 bits.

Si desea una llave de servidor o repetidor se generará una única llave para cada uno de estos componentes en el directorio “keys/symmetric/server” o “keys/symmetric/repeater”, según corresponda, bajo el nombre “nombre_del_componente” + ”Key.key”.

En cambio, si se desea generar un llave de un cliente, su llave estará en el directorio “keys/symmetric/clients” y se preguntará un parámetro extra que representa el ID del cliente. Las llaves de los clientes se guardarán bajo la estructura de nombre: “Client” + “id_del_cliente” + “Key.key”.

Llaves asimétricas:

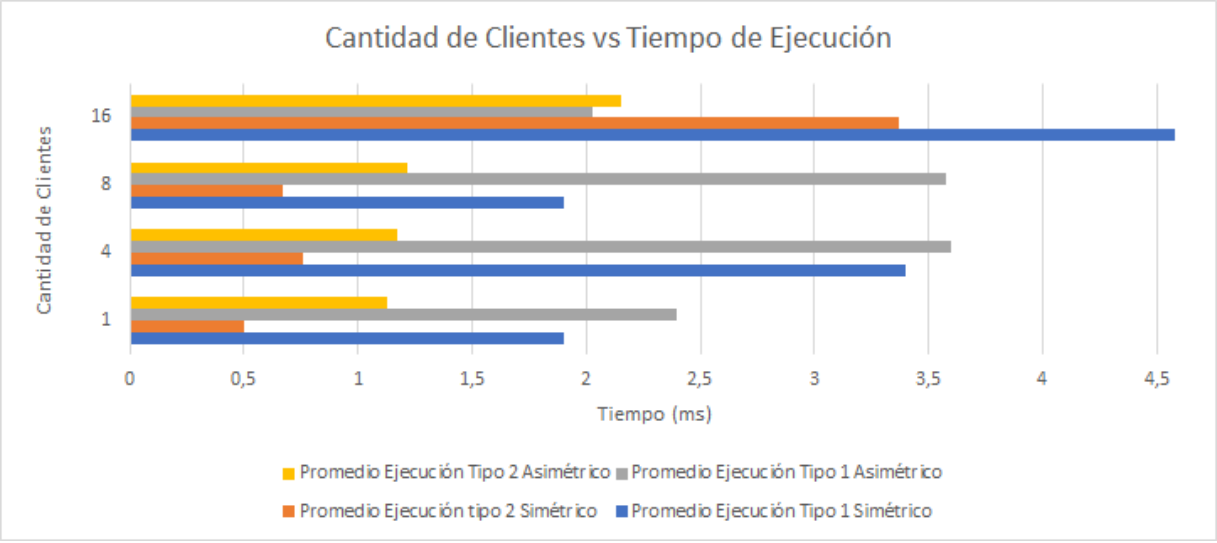
Para generar las llaves asimétricas, el programa ‘security/Asymmetric.java’ le preguntará el tipo de llaves que desea, para un servidor, repetidor o cliente. El par de llaves que genera este prototipo se genera usando el algoritmo de RSA con una llave de 1024 bits.

Si desea llaves de servidor o repetidor se generará un único par de llaves para cada uno de estos componentes en el directorio “keys/asymmetric/server” o “keys/asymmetric/repeater”, según corresponda, bajo el nombre “nombre_del_componente” + ”Key” y la terminación .pub o .key de acuerdo a si es llave pública o privada.

En cambio, si se desea generar un par de llaves de un cliente, su llave estará en el directorio “keys/asymmetric/clients” y se preguntará un parámetro extra que representa el ID del cliente. Las llaves de los clientes se guardarán bajo la estructura de nombre: “Client” + “id_del_cliente” + “Key” y la terminación .pub o .key de acuerdo a si es llave pública o privada.

Pruebas y análisis

Cantidad de Clientes	Promedio Ejecución Tipo 1 Simétrico	Promedio Ejecución tipo 2 Simétrico	Promedio Ejecución Tipo 1 Asimétrico	Promedio Ejecución Tipo 2 Asimétrico
1	1927526ns ≈1.9ms	505106ns ≈0.5ms	2421723ns≈ 2.4ms	1133385ns≈ 1.13ms
4	3464896ns ≈3.4ms	763093ns ≈ 0.76ms	3603677ns≈ 3.6ms	1175790ns≈ 1.17ms
8	1945420ns≈ 1.9ms	679601ns≈ 0.67ms	3582020ns≈ 3.58ms	1220459ns≈ 1.22ms
16	4580881ns≈ 4.58ms	3379532ns ≈ 3.37ms	2066636ns≈ 2.03ms	2156083ns≈ 2.15ms



Ejecución Tipo 1: Desde la recepción de la solicitud, incluyendo las operaciones para descifrar y luego volver a cifrar para el servidor.

Ejecución Tipo 2: Desde la recepción de la respuesta, incluyendo las operaciones para descifrar y luego volver a cifrar para el cliente.

Comentarios respecto a los resultados:

Los resultados fueron fruto de la clase MultipleClients donde se ejecutó el comando de cliente múltiples veces, en el caso de 16, se logró una cantidad de alrededor de 300 peticiones, ejecutando la clase aproximadamente 20 veces consecutivas.

Se nos hizo curioso que con pocos clientes, los tiempos se comportaron como se esperaba, que el cifrado Asimétrico tome más tiempo que el simétrico, sin embargo, en el caso de los 16 clientes concurrentes, ocurrió lo contrario. Es posible que este resultado, se deba a la ejecución de otro proceso en el computador que ralentizará la ejecución del cifrado simétrico, sin embargo, todos los tiempos de ejecución son excelentes y bastante rápidos.

Procesador Intel i5-9400F

@2.90 GHz base

@3.80 GHz Idle-Work

- **Ciclos Simétrico - Vía Repetidor-Servidor**

En promedio una solicitud toma 192752ns en modo simétrico. Se sabe que Tiempo de ejecución = Ciclos/Velocidad del Reloj, entonces al reemplazar:

Sabiendo que 1Hz = 1 ciclo/segundo

$$1927526ns = \frac{Ciclos}{3.8GHz}$$

$$0.00019s * \frac{3.8 * 10^9 ciclos}{segundos} = 732457 ciclos$$

La operación toma 732457 ciclos en completarse.

- **Ciclos Simétrico - Vía Repetidor-Cliente**

Tiempo: 505106ns la operación

$$\frac{505106ns}{10^9 ns/s} = \frac{Ciclos}{3.8GHz}$$

$$\frac{505106ns}{10^9 ns/s} * \frac{3.8 * 10^9 ciclos}{segundos} = 1919402{,}8\ ciclos$$

Ciclos: 1919402,8 ciclos toma la operación

- **Ciclos Asimétrico - Vía Repetidor-Servidor**

Tiempo: 2421723ns

$$\frac{2421723ns}{10^9 ns/s} = \frac{Ciclos}{3.8GHz}$$

$$\frac{2421723ns}{10^9 ns/s} * \frac{3.8 * 10^9 ciclos}{segundos} = 9202547{,}4\ ciclos$$

Ciclos: 9202547,4 ciclos toma la operación.

- **Ciclos Asimétrico - Vía Repetidor-Cliente**

Tiempo: 1133385ns la operación

$$\frac{1133385ns}{10^9 ns/s} = \frac{Ciclos}{3.8GHz}$$

$$\frac{1133385ns}{10^9 ns/s} * \frac{3.8 * 10^9 ciclos}{segundos} = 4306863\ ciclos$$

Ciclos: 4306863 ciclos toma la operación.

Referencias

- Pawamoy.github.io. 2021. *Passing Makefile arguments to a command, as they are typed in the command line. - pawamoy's website*. [online] Available at: https://pawamoy.github.io/posts/pass-makefile-args-as-typed-in-command-line [Accessed 4 November 2021].
- Java - Problem with multiple, c., Sengupta, M. and Neyland, T., 2021. *Java - Problem with multiple, concurrent runtime.exec() InputStreams*. [online] Stack Overflow. Available at: https://stackoverflow.com/questions/4753901/java-problem-with-multiple-concurrent-runtime-exec-inputstreams [Accessed 27 October 2021].
- GeeksforGeeks. 2021. *Symmetric Encryption Cryptography in Java - GeeksforGeeks*. [online] Available at: https://www.geeksforgeeks.org/symmetric-encryption-cryptography-in-java [Accessed 29 October 2021].
- Baeldung. 2021. *Java AES Encryption and Decryption*. [online] Available at: https://www.baeldung.com/java-aes-encryption-decryption [Accessed 29 October 2021].
- dzone.com. 2021. *Encryption, Part 1: Symmetric Encryption - DZone Security*. [online] Available at: https://dzone.com/articles/encryption-part-1-symmetric-encryption [Accessed 1 November 2021].
- R., Bender, T. and Pathai, N., 2021. *Running java runtime.exec() for multiple process*. [online] Stack Overflow. Available at: https://stackoverflow.com/questions/18010604/running-java-runtime-exec-for-multiple-process [Accessed 5 November 2021].

- Herongyang.com. 2021. *JceSecretKeyTest.java - Secret Key Test Program*. [online] Available at: https://www.herongyang.com/JDK/Secret-Key-Test-Program-JceSecretKeyTest.html [Accessed 27 October 2021].
- 2021. *Load RSA public key from file*. [online] Stack Overflow. Available at: https://stackoverflow.com/questions/11410770/load-rsa-public-key-from-file [Accessed 27 October 2021].