Caso 3: Canales Seguros

Andrea Lucía Galindo 202122477
Luis Fernando Ruiz Ortega 202211513
Santiago Navarrete Varela 202211202

Organización Archivos

- ClienteGrande: Esta clase es la encargada de delegar usuarios según sean pedidos. Además, dentro de ella se encuentran las variables de tiempo las cuales son actualizadas y mostradas al final para saber cuánto se demoraron los procesos de interés.
- Cliente: Esta clase maneja la lógica de cada cliente. dentro del método run() se encuentran los pasos en órden que tiene que ir realizando para cumplir con el protocolo establecido. Asimismo, es importante resaltar que antes de realizar el ***PASO 1***, cada usuario lee la llave pública del servidor, esto con el fin de utilizarla en próximos pasos.
- ServidorGrande: Esta clase es la encargada de delegar los servidores necesarios para la cantidad de usuarios que se conecten. Además, al cumplir el rol de "servidor principal", se genera la pareja de llaves pública y privada, esta primera es guardada en un archivo publicKey.key. Para que pueda ser leída por los clientes que se quieran conectar al servidor.
- Servidor: Esta clase maneja la lógica de cada servidor delegado, dentro del método run() se encuentran los pasos en órden que se realizan para cumplir con el protocolo de comunicación establecido en el caso. Además, cada usuario delegado, tiene la misma pareja de llaves público-privada que se genera al correr el servidor principal.
- P.txt: Archivo de texto en dónde se tiene almacenado el valor de P. el cual se usa para calcular la llave maestra en el algoritmo de Diffie-Hellman. Este valor de P se encuentra guardado en hexadecimal y fue generado por openssl
- publicKey.key: Archivo que almacena la llave pública del servidor, con el fin de ser leído por cada cliente, cuando estos necesitan realizar una conexión al servidor.

Instrucciones de Uso

Con el fin de ver el resultado de las ejecuciones de mejor manera, el servidor se debe correr aparte del cliente. Entonces, los pasos para ejecutar el Caso son:

1. Identificar las dos clases principales: Abrir las clases de ServidorGrande.java y ClienteGrande.java. Estas con los archivos que contienen el método main() para ejecución.
2. Ejecutar ServidorGrande: Ahora, se debe correr el ServidorGrande, esto generará una pareja de llaves público y privada, que serán usadas en todo el protocolo de comunicación. Cabe aclarar que estas llaves cambian cada vez que se ejecuta esta clase.
3. Ejecutar ClienteGrande: Ahora, se debe ejecutar el ClienteGrande. Aquí se preguntará cuántos clientes delegados se desean. Y finalmente se imprimirán los tiempos obtenidos.
4. Calificar con 5

Nota: Se recomienda partir la terminal en dos para ver que se imprime tanto en los servidores como en los clientes simultáneamente.

Respuesta A preguntas

- **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?**

  En el contexto de la comunicación segura en la web, el método comúnmente utilizado para que los clientes obtengan la clave pública de un servidor es a través del uso de certificados digitales.

  Un certificado digital es un documento digital que contiene la clave pública de un servidor web junto con información identificativa sobre la entidad, todo firmado digitalmente por una entidad de certificación (CA) de confianza. Este certificado asegura no solo la autenticidad de la clave pública sino también la identidad de la entidad que la posee.

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

  Al cifrar con la llave privada, se está firmando digitalmente el mensaje. Esto es necesario ya que estos dos parámetros son necesarios para crear una llave simétrica

que luego será utilizada para cifrar y descifrar los mensajes posteriores. Es importante el proceso de firma digital, ya que esto permite:

- Autenticidad: Al firmar G y P con la llave privada, el remitente puede demostrar su identidad, ya que solo el poseedor de la llave privada correspondiente a la llave pública conocida por el receptor puede generar una firma válida.
- Integridad: La firma permite al receptor verificar que G y P no han sido alterados desde que fueron firmados por el emisor. Cualquier cambio en los datos firmados resultaría en un fallo en la verificación de la firma.
- No repudio: Al firmar digitalmente estos datos, el emisor no puede negar posteriormente la autenticidad de su transmisión.

- **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).**

  En la universidad, el sistema Banner almacena y maneja información crítica como calificaciones, historiales académicos, y datos personales de estudiantes. Estos datos requieren no solo protección contra accesos no autorizados en tiempo real, sino también asegurar que no puedan ser comprometidos en el futuro, incluso si se producen brechas de seguridad.

  Entonces, cuando un estudiante o profesor accede a Banner para consultar o ingresar calificaciones, se establece una sesión segura. Utilizando Diffie-Hellman para intercambiar claves de sesión, se puede garantizar que cada sesión tenga una clave única y temporal que no se base en claves de largo plazo directamente.

  Esto beneficiaría en: Protección contra compromisos futuros de claves, seguridad en el acceso a datos y confianza en la plataforma.

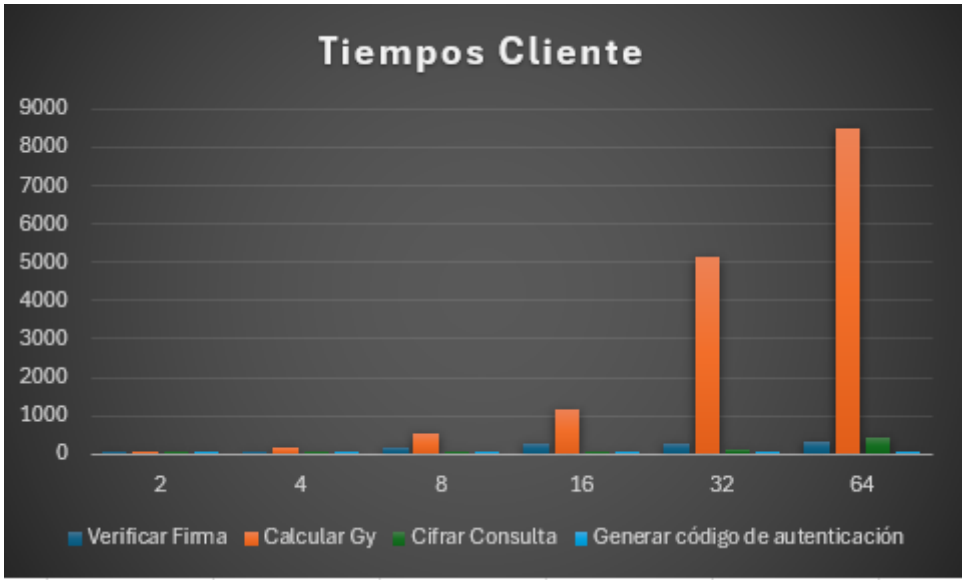
Resultados Obtenidos

Con el fin de medir cómo varían y crecen los tiempos de cifrado y descifrado en el protocolo de comunicación establecido. Se diseñaron 6 escenarios en dónde se variaba el número de clientes delegados en el sistema, y se tomaba el tiempo que duraba cada operación crítica. En ese órden de ideas, los resultados obtenidos fueron:

Tiempos Cliente

Aquí se presentan los tiempos tomados en las operaciones realizadas por los clientes.

	Tiempo Cliente (ms)			
Número de Clientes	Verificar Firma	Calcular Gy	Cifrar Consulta	Generar código de autenticación
2	34,29	74,37	2,92	1,25
4	50,84	194,56	9,05	1,15
8	167,76	524,47	17,74	3,34
16	260,15	1140,42	53,84	9,36
32	282,1	5155,9	121,26	18,63
64	302,91	8478,55	443,61	13,35

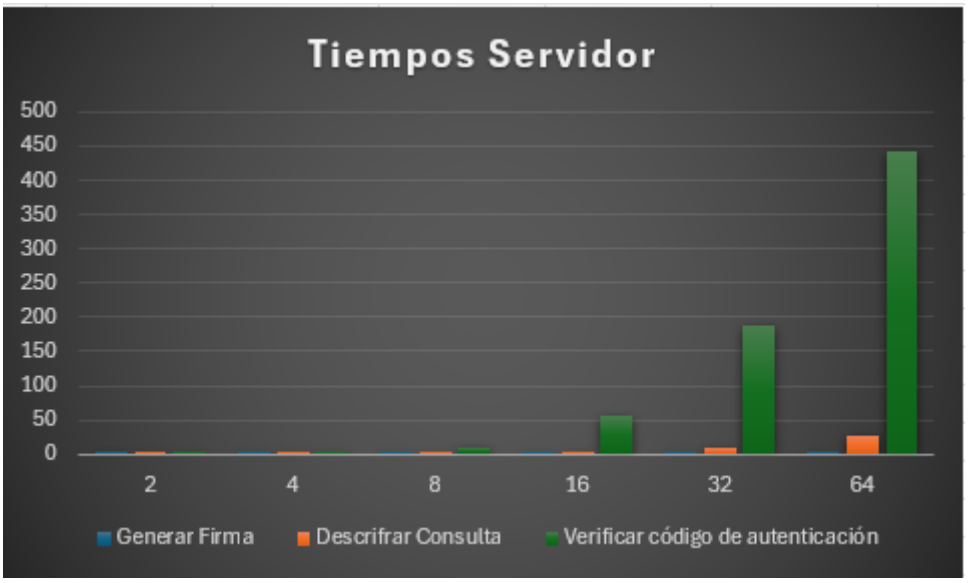


Como se puede observar, se ve que la operación que crece más rápido en tiempo es calcular Gy, esto se puede deber a varios factores relacionados con la complejidad computacional y el manejo de recursos que esta operación específica requiere. En primer lugar, calcular Gy en el contexto de algoritmos criptográficos, como el Diffie-Hellman, involucra operaciones de exponenciación modular, las cuales son intensivas en términos de procesamiento. Estas operaciones requieren un uso significativo de la CPU y tienden a escalar en complejidad con el aumento del tamaño de los números involucrados. Además, como los parámetros usados (como la base g y el módulo p) son de gran tamaño, el tiempo de cálculo se incrementa considerablemente.

Tiempos Servidor

Aquí se presentan los tiempos tomados en las operaciones realizadas por los servidores.

	Tiempo Servidor (ms)		
Número de Clientes	Generar Firma	Descrifrar Consulta	Verificar código de autenticación
2	0,01	0,49	0,76
4	0,01	0,87	1,82
8	0,02	1,75	9,94
16	0,1	4,25	56,45
32	0,22	10,02	188,75
64	0,34	26,86	443,61



Como se puede observar en la gráfica y tabla, la operación que más incrementa en tiempo según el número de clientes delegados es verificar el código de autenticación (HMAC), esto se debe a que la generación y verificación de HMAC implica el cálculo de funciones hash criptográficas, que son operaciones computacionalmente intensivas. A medida que aumenta el número de clientes, la cantidad de veces que esta operación debe ejecutarse crece linealmente, aumentando el tiempo total necesario para procesar todas las verificaciones de HMAC.

Además, como los mensajes que se están verificando son grandes, el tiempo necesario para procesarlos será mayor. En un entorno con muchos clientes, si cada cliente envía datos sustanciales para verificar, el impacto en el tiempo total de procesamiento puede ser significativo.

Estimación velocidad del procesador

Finalmente, con ayuda de los datos tomados, se puede estimar la velocidad del procesador. Para los siguientes cálculos se tomaron como base, los tiempos de 64 clientes.

- **Cifrar**

  Tiempo tomado: 443.61 ms

  Operaciones por segundo (OPS) = 1000 / 443,61 ≈ 2.25 operaciones por segundo
- **Verificar Firma**

  Tiempo tomado: 302.91 ms

  Operaciones por segundo (OPS) = 1000 / 302.91 ≈ 3.30 operaciones por segundo
- **Generar Código Autenticación**

  Tiempo tomado: 13.35 ms

  Operaciones por segundo (OPS) = 1000 / 13.35 ≈ 74.91 operaciones por segundo

Y las especificaciones del procesador:

Velocidad de base:	3,00 GHz
Sockets:	1
Núcleos:	6
Procesadores lógicos:	12
Virtualización:	Habilitado
Caché L1:	384 kB
Caché L2:	3,0 MB
Caché L3:	8,0 MB

Según esto, se podría multiplicar las tareas que se realizan por el número de procesadores lógicos (Threads) totales, lo que daría:

- **Cifrar**

  Operaciones por segundo (OPS) ≈ 2.25

  Operaciones en total = 2.25 * 12 = 27 operaciones por segundo
- **Verificar Firma**

  Operaciones por segundo (OPS) ≈ 3.30

  Operaciones en total = 3.30 * 12 = 39,6 operaciones por segundo
- **Generar Código Autenticación**

  Operaciones por segundo (OPS) ≈ 74.91

  Operaciones en total = 74.91 * 12 = 898.92 operaciones por segundo

Bibliografía

- https://letsencrypt.org/es/how-it-works/#:~:text=El%20objetivo%20de%20Let's%20Encrypt,certificados%20en%20el%20servidor%20web.
- https://csrc.nist.gov/projects/cryptographic-standards-and-guidelines
- https://wiki.openssl.org/index.php/Diffie_Hellman
- https://learn.microsoft.com/es-es/dotnet/api/java.security.signature?view=net-android-34.0
- https://www.ibm.com/docs/es/was-zos/9.0.5?topic=cksgc-example-developing-key-key-pair-generation-class-automated-key-generation