Parcial 2 Criptografía

Integrantes:

Brancys Barrios

Nathalia De La Rans

Diego Linero

Docente:

Eduardo Angulo Madrid

Barranquilla

Universidad del Norte

17/10/2024

GitHub:
https://github.com/Brancys/Parcial2-Cripto

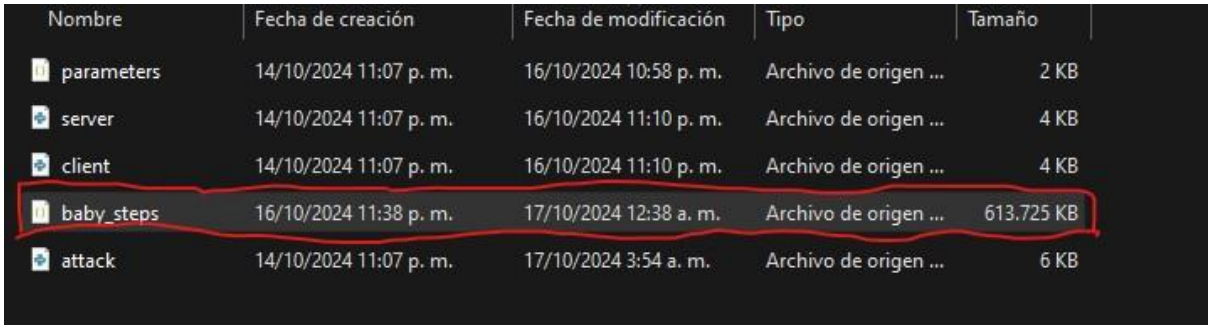
Escenario 1

1. **¿Qué factores influyen en la dificultad de resolver el problema del logaritmo discreto utilizando el algoritmo de "pasos de bebé, pasos de gigante"? ¿Cuáles fueron los resultados del ataque y qué conclusiones puedes extraer?**

Al analizar la dificultad de resolver el problema del logaritmo discreto usando el algoritmo de "pasos de bebé, pasos de gigante", nos damos cuenta que influyen varios factores que no solo están relacionados con el tamaño de los números que usamos, sino también con cómo se implementan esos cálculos. En la práctica, el algoritmo es una solución eficiente para resolver el logaritmo discreto en grupos cíclicos de tamaño pequeño o moderado, pero cuando tratamos con valores realmente grandes como los usados en el intercambio de claves de Diffie-Hellman, la cosa cambia radicalmente.
El algoritmo básicamente busca encontrar el logaritmo discreto dividiendo el problema en dos partes: primero se generan los "pasos de bebé", que son todos los posibles valores$g^j mod\ p$, y luego se recorren los "pasos de gigante" buscando colisiones con estos valores precomputados, pero si nos preguntamos ¿en qué radica la dificultad de esto? Pues, todo se reduce al tamaño de los números con los que trabajamos. En nuestro caso, al usar un número primo pequeño como 227, podemos hacer las cuentas relativamente rápido, pero si nos trasladamos a un entorno real, cuando usamos números primos de más de 1024 bits, esos cálculos se vuelven mucho más pesados. Resolver el logaritmo discreto con estos valores grandes requiere muchísima memoria y tiempo de cómputo. Pensamos que otro factor que influye es la eficiencia de las operaciones modulares que usamos en el código. Por ejemplo, en el código del atacante, la función $pow(\square)$se usa para hacer cálculos modulares de manera más rápida, pero cuando los números son realmente grandes, incluso estas optimizaciones comienzan a sentirse lentas. Eso nos lleva a darnos cuenta de que no solo se trata de qué tan inteligente sea el algoritmo, sino de qué tan bien optimizado esté el código para lidiar con números enormes. Cuando lanzamos el ataque en los archivos, vemos que, si todo está bien configurado, el algoritmo es capaz de encontrar la clave privada del servidor en relativamente poco tiempo. Pero claro, estos valores son bastante chicos, y aunque funciona cuando los valores de $p\ q\ g$ son pequeños, lo que aprendemos aquí es que en sistemas reales, el mismo ataque podría fallar simplemente porque no es factible hacer las operaciones dentro de un tiempo razonable. Incluso, con un tiempo límite de una hora, si los números son lo suficientemente grandes, podríamos no ver resultados en absoluto que es lo que sucede al probar los últimos valores de $p\ q\ g$ que aparecen en el **Json.**
Así que, al final, lo que concluimos de este ejercicio es que el algoritmo de "pasos de bebé, pasos de gigante" funciona, pero está limitado por el tamaño de los números que utilizamos.

```
(c) Microsoft Corporation. Todos los derechos reservados.

D:\Parcial2-Cripto>C:/Users/USUARIO/AppData/Local/Programs/Python/Python311/python.exe d:/Parcial2-Cripto/Escenario1/attack.py
Creando y guardando los baby steps en el archivo...
Tiempo limite alcanzado (3600.00 segundos).
No fue posible encontrar la llave privada.
Tiempo total transcurrido: 3600.00 segundos

D:\Parcial2-Cripto>
```

Para tener en cuenta. Debido a que la creación del diccionario en baby steps estaba consumiendo mucha memoria RAM, optamos por cambiar el código para que cree un archivo y acceda cada vez que este fuese a agregar un nuevo dato. También, al finalizar la hora no terminó ni siquiera de llenar el diccionario.
El tamaño final del diccionario al terminar la hora fue de 613.725 KB.
Eso probando con el penúltimo set de parámetros, por lo que, si no fue posible con este conjunto, menos aún lo sería con el último.



Nombre	Fecha de creación	Fecha de modificación	Tipo	Tamaño
parameters	14/10/2024 11:07 p. m.	16/10/2024 10:58 p. m.	Archivo de origen ...	2 KB
server	14/10/2024 11:07 p. m.	16/10/2024 11:10 p. m.	Archivo de origen ...	4 KB
client	14/10/2024 11:07 p. m.	16/10/2024 11:10 p. m.	Archivo de origen ...	4 KB
baby_steps	16/10/2024 11:38 p. m.	17/10/2024 12:38 a. m.	Archivo de origen ...	613.725 KB
attack	14/10/2024 11:07 p. m.	17/10/2024 3:54 a. m.	Archivo de origen ...	6 KB

2. **¿Cuáles son los beneficios y desventajas de utilizar Diffie-Hellman sobre un grupo cíclico $\mathbb{F}_p^*$ en comparación con otros métodos de intercambio de llaves (investigue otros métodos)?**

Al usar Diffie-Hellman sobre un grupo cíclico F_p^*, encontramos varias ventajas que lo hacen popular para el intercambio de llaves. Un gran beneficio es que es simple y eficiente para compartir una llave secreta entre dos partes, incluso si no se conocen previamente. Se basa en la dificultad del logaritmo discreto, lo que lo hace seguro, siempre que los números primos sean lo suficientemente grandes como se implementó durante este Escenario. No requiere preacuerdos complicados más allá de compartir parámetros públicos, lo que facilita su implementación en distintas situaciones. Sin embargo, Diffie-Hellman tiene algunas desventajas, una de las más importantes es que no ofrece autenticación por sí mismo, lo que lo hace vulnerable a ataques de intermediario. Para resolver esto, se pueden añadir firmas digitales u otros esquemas de autenticación, pero eso complica su uso. Además, con la llegada de la computación cuántica, hay preocupaciones porque algoritmos como Shor podrían romper su seguridad.

Cuando lo comparamos con otros métodos como RSA, vemos que Diffie-Hellman es más eficiente, ya que RSA requiere más cálculos relacionados con la factorización de números

grandes. Aunque RSA tiene la ventaja de incluir autenticación, también puede ser vulnerable si no se eligen bien los números. Otro método que investigamos y que resulta muy interesante es el de curvas elípticas. Este ofrece la misma seguridad que Diffie-Hellman pero con claves más pequeñas, lo que lo hace mucho más eficiente, especialmente para dispositivos con recursos limitados. Además, ECC es más resistente a la computación cuántica, lo que lo convierte en una opción muy potente en el contexto actual.

En resumen, Diffie-Hellman sigue siendo una opción fuerte y eficiente, pero métodos como ECC están ganando terreno debido a sus ventajas en seguridad y rendimiento, especialmente pensando en futuras amenazas como los ataques cuánticos.

```python
# Función para generar llaves públicas y privadas con q
Tabnine | Edit | Test | Explain | Document | Ask
def diffie_hellman_keypair(p, g, q):
    private_key = int.from_bytes(get_random_bytes(16), 'big') % q  # Llave privada en [1, q-1]
    public_key = pow(g, private_key, p)  # g^private_key mod p
    return private_key, public_key
```

```python
Tabnine | Edit | Test | Explain | Document | Ask
def baby_step_giant_step_with_timeout(p, g, h, time_limit):
    m = math.isqrt(p) + 1  # Calcular m tal que m^2 >= p
```

Escenario 2

1. **¿Qué vulnerabilidades inherentes a Diffie-Hellman sobre curvas elípticas se pueden explotar en un ataque de hombre en el medio?**

Nosotros pensamos que la principal vulnerabilidad que explota un ataque de hombre en el medio en el intercambio de claves Diffie-Hellman con curvas elípticas (ECDH) es la falta de autenticación entre las partes. En ECDH sin autenticación, ni el cliente ni el servidor verifican la identidad del otro antes de intercambiar las claves públicas, lo que permite que un atacante intercepte y manipule la comunicación.

Colocar al atacante en el medio y realizar un ataque de intermediario. El atacante crea dos sesiones ECDH separadas seguras: una con el cliente y la otra con el servidor, pero encriptadas con su clave. Esto permite descifrar, leer, modificar e incluso reinsertar los mensajes. Lo peor es que tanto el cliente como el servidor creen que se comunican directamente entre sí, pero en realidad, la conexión es controlada por el atacante. En cuanto al Escenario 2, el cliente y el servidor se conectan directamente, y el código del Escenario 2 no tiene autenticación, por lo que un atacante podría interceptar las claves públicas y cambiar la clave pública por la suya.

Esto se puede observar en el código del atacante, donde intercepta las claves públicas del cliente y del servidor (client_public_bytes y server_public_bytes). Luego, en lugar de reenviarlas, el atacante envía su propia clave pública (attacker_public_bytes) a ambas partes. Así, establece dos conexiones ECDH independientes, engañando tanto al cliente como al

servidor, así el atacante puede descifrar, leer y modificar todos los mensajes, aprovechándose de la ausencia de autenticación en el protocolo.

Para evitar esto, nosotros creemos que es esencial implementar mecanismos de autenticación, como certificados digitales, que permitan verificar la identidad de ambas partes y prevenir ataques de este tipo.

```
# Interceptar la llave pública del cliente
client_public_bytes = conn.recv(1024)
client_public_key = ec.EllipticCurvePublicKey.from_encoded_point(ec.SECP256R1(), client_public_bytes)
print("Llave pública del cliente interceptada.")

# Enviar la llave pública del atacante al cliente en lugar de la del servidor
conn.sendall(attacker_public_bytes)

# Interceptar la llave pública del servidor
server_public_bytes = client_socket.recv(1024)
server_public_key = ec.EllipticCurvePublicKey.from_encoded_point(ec.SECP256R1(), server_public_bytes)
print("Llave pública del servidor interceptada.")

# Enviar la llave pública del atacante al servidor en lugar de la del cliente
client_socket.sendall(attacker_public_bytes)
```

2. **¿Qué contramedidas podrían implementarse para mitigar estos ataques en un entorno real?**

Por tanto, tal como se propuso en el punto anterior, nuestra elección para prevenir ataques de intermediario en ECDH implica un mecanismo de autenticación. El más rotundo sería un par de certificados digitales emitidos por una CA de confianza, ya que garantizarían que ambas partes tuvieran la posibilidad de verificar desde el superprincipio que esta clave pública vino de la entidad que pretende. Además, las partes podrían autenticarse entre sí iniciando el intercambio de claves con un secreto pre-compartido, si se sabe que ya lo poseen, o con protocolos de autenticación de clave pública, como Needham-Schroeder, que permiten verificar las entidades antes de realizar el intercambio de claves ECDH. Finalmente parece lógico considerar la ECDH autenticada en sí, que incorpora la autenticación dentro del protocolo propiamente dicho. A pesar, de todas estas soluciones que claramente aumentarían la complicación, creemos que no podrían dejar de estar presentes en la transferencia real por fuera de un experimento.

Escenario 3

1. **Cuáles son las principales diferencias en términos de eficiencia y seguridad entre RSA OAEP y ElGamal? Justifica cuál criptosistema asimétrico sería más adecuado en un contexto donde el tamaño de los mensajes y la rapidez de la comunicación son críticos.**

A pesar de tener el mismo principio de proteger la información, ElGamal y RSA-OAEP son dos criptosistemas asimétricos que difieren en sus bases matemáticas y en el rendimiento. Mientras el primer está basado en la dificultad de resolver el problema del logaritmo discreto, el segundo se basa en la dificultad de factorización de números primos grandes. En cuanto al rendimiento, ElGamal suele ser mucho más rápido a la hora de cifrar, sin embargo, su tamaño es mayor. En un caso, cuando la cantidad de texto cifrado es un brazo, por ejemplo, cuando se trata del limitado espacio del hardware, este hecho es muy importante. Por otro lado, RSA-OAEP en el ataque cifrar resulta ser más lento, sin embargo, sus textos cifrados son considerablemente más cortos y este cifrado es más resistente a los ataques, lo que lo hace más conveniente en términos de seguridad.

Cuando la eficiencia es la prioridad, ElGamal puede ser preferible, lo cual es particularmente relevante para aplicaciones específicas, como el Internet de las Cosas, que requiere altas tasas de procesamiento. Por otro lado, si la posibilidad de un ataque se considera más crítica, la mejor opción es RSA-OAEP. En general, la elección entre ElGamal y RS-OAEP dependerá del nivel deseado de eficiencia combinado con seguridad y los requisitos de aplicación necesarios. Los algoritmos tienen sus ventajas y desventajas.

Escenario3 > Comparación > {} comparison_results.json > ...


```
{
  "ElGamal": {
    "encryption_time": 0.010998964309692383,
    "decryption_time": 0.012001752853393555,
    "encrypted_message_size": 616
  },

  "RSA_OAEP": {
    "encryption_time": 0.0019958019256591797,
    "decryption_time": 0.0060007572174072266,
    "encrypted_message_size": 256
  }
}
```


2. **En base a la comparación con las comunicaciones simétricas de los escenarios anteriores, ¿qué conclusiones puedes extraer sobre el uso de cifrado simétrico vs. asimétrico en aplicaciones de comunicación de red?**

Ambas simetrías, simétrico y asimétrico, tienen sus ventajas y desventajas, y nuestra decisión sobre cuál de ellas emplear dependería casi por completo de cuán importante es la eficiencia en comparación con la seguridad. En general, la simetría, especialmente encriptaciones como Salsa20 y AES-256, es mucho más rápida y eficiente para cifrar y descifrar datos. Es muy

bueno para datos en tiempo real o sistemas que transmiten y reciben cantidades masivas de datos lo más rápido posible. Sin embargo, no podemos evitar pensar que su seguridad, mientras que muy sólida, sigue siendo por debajo de la asimetría alcanzable.

Por otra parte, el cifrado asimétrico se caracteriza por ser mucho más seguro, debido al uso de la clave pública y privada separada, al contrario, que es más fácil para autenticar y garantizar la integridad del mensaje, no puede hacerse mediante el cifrado simétrico. Sin embargo, no es rápidamente y crea el mensaje cifrado más grande y enlatado Latencia y ancho de banda significan más consumo.

Como resumen, si necesita la velocidad y eficiencia primero, el cifrado simétrico será la opción adecuada. Sin embargo, cuando necesita seguridad, sobre todo, como la firma digital o el intercambio de claves, el cifrado asimétrico será la mejor respuesta. En realidad, dependerá de la prioridad que necesita, mayor velocidad o seguridad.

Conclusiones

Seguridad y eficiencia de los esquemas criptográficos. A lo largo de este trabajo aprendimos que la seguridad y la eficiencia de los esquemas criptográficos son procesos dependientes. Desde el tamaño de las claves propias, la complejidad de los algoritmos y la claridad del código, todos tienen un impacto directo en que tan resistente será el esquema contra los ataques y qué tal será la forma en que se desempeñará el sistema. Esto nos enseñó la importancia de elegir un esquema apropiado para el caso basado en el contexto, priorizando la seguridad estricta para los dominios críticos y la eficiencia para los dominios en los que la velocidad de la comunicación es una restricción.

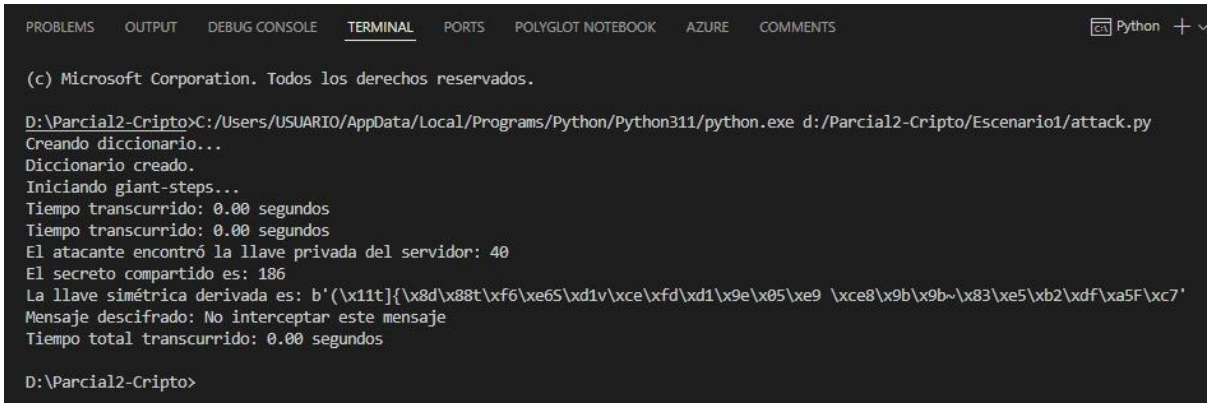
Relevancia de la correcta implementación y mitigación de ataques. Nosotros pensamos que cuando se habla de relevancia en la correcta implementación de protocolos criptográficos, se puede ser muy evidente casi que cualquier fallo, como la falta de autenticación en el intercambio de claves como se pudo evidenciar en los escenarios anteriores lo que hacía que pudiera ser explotado por atacantes, comprometiendo así la seguridad del sistema. Ataques como el **hombre en el medio** que se mencionó en el escenario2 resaltan la importancia de verificar la identidad de las partes involucradas en la comunicación y de utilizar mecanismos de protección adicionales, como certificados digitales o protocolos de autenticación de clave pública, solución que se mencionó anteriormente. La seguridad del sistema depende no solo de la robustez del algoritmo criptográfico, sino también de la correcta implementación y de la mitigación de posibles ataques.

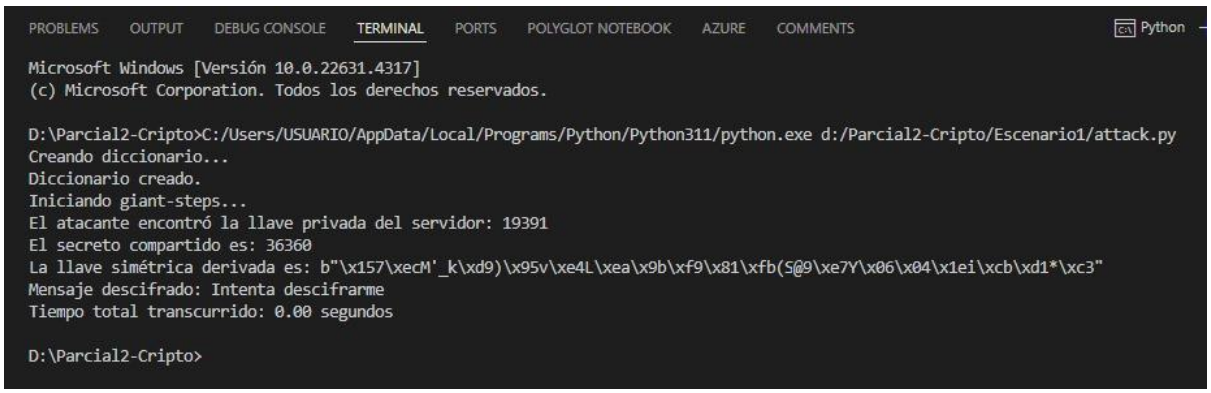
Recomendaciones para un intercambio seguro de llaves y comunicación cifrada robusta. Las recomendaciones que como grupo recomendamos, basándonos en nuestra investigación, incluyen utilizar algoritmos criptográficos robustos y actualizados, así como implementar mecanismos de autenticación para prevenir ataques de intermediario. También creemos que

elegir el tamaño de clave adecuado es crucial esto se evidencio en la anterior implementación de Diffie-Hellman, ya que este influye directamente en la seguridad del sistema. Además, es fundamental optimizar el código para mejorar la eficiencia punto que se mencionó anteriormente en las recomendaciones, especialmente en aplicaciones donde la velocidad de comunicación es crítica. Considerar el uso de criptografía híbrida puede ofrecer un equilibrio entre seguridad y eficiencia, aprovechando las ventajas del cifrado simétrico y asimétrico. Finalmente, realizar pruebas de seguridad es esencial para identificar vulnerabilidades en la implementación. Siguiendo estas recomendaciones, se pensamos que se puede lograr un intercambio seguro de llaves y una comunicación cifrada robusta, protegiendo la información sensible de posibles ataques.

Capturas

Escenario 1





```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  POLYGLOT NOTEBOOK  AZURE  COMMENTS                    Python

(c) Microsoft Corporation. Todos los derechos reservados.

D:\Parcial2-Cripto>C:/Users/USUARIO/AppData/Local/Programs/Python/Python311/python.exe d:/Parcial2-Cripto/Escenario1/attack.py
Creando diccionario...
Diccionario creado.
Iniciando giant-steps...
El atacante encontró la llave privada del servidor: 3175299
El secreto compartido es: 3233290
La llave simétrica derivada es: b'\x7f\x03`\xfc\xb2\xf2\\9\x98\x9b\xbe\xd0\x89\xed\xb6\xdbc\xa2J\xc5\xde\x94\xedf\xc8\x8f]\xa1k\xc2\x1d\x0b'
Mensaje descifrado: Apuesto a que no puedes descifrarlo
Tiempo total transcurrido: 0.00 segundos

D:\Parcial2-Cripto>
```

```
(c) Microsoft Corporation. Todos los derechos reservados.

D:\Parcial2-Cripto>C:/Users/USUARIO/AppData/Local/Programs/Python/Python311/python.exe d:/Parcial2-Cripto/Escenario1/attack.py
Creando y guardando los baby steps en el archivo...
Tiempo limite alcanzado (3600.00 segundos).
No fue posible encontrar la llave privada.
Tiempo total transcurrido: 3600.00 segundos

D:\Parcial2-Cripto>
```

Escenario 2

Consola atacante

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  POLYGLOT NOTEBOOK  AZURE  COMMENTS                    Python

D:\Parcial2-Cripto>C:/Users/USUARIO/AppData/Local/Programs/Python/Python311/python.exe d:/Parcial2-Cripto/Escenario2/attack_MitM.py
Llave pública del servidor interceptada.
Atacante en espera de conexiones...
Conexión interceptada con el cliente: ('192.168.1.15', 51762)
Llave pública del cliente interceptada.
Llave simétrica con el cliente: 29547c20b508c88324df861edad012d0e7db2ce6d12cd0e117fd214208b15bda
Llave simétrica con el servidor: 14eb4f06ee0c9c24166148351496880be260e05df4d12fdeee03330afaf50eda
Mensaje interceptado del cliente: Hola
Mensaje interceptado del servidor: Bienvenido señor usuario
Mensaje interceptado del cliente: 123456
Mensaje interceptado del servidor: 789 10
Conexión cerrada por el atacante.

D:\Parcial2-Cripto>
```

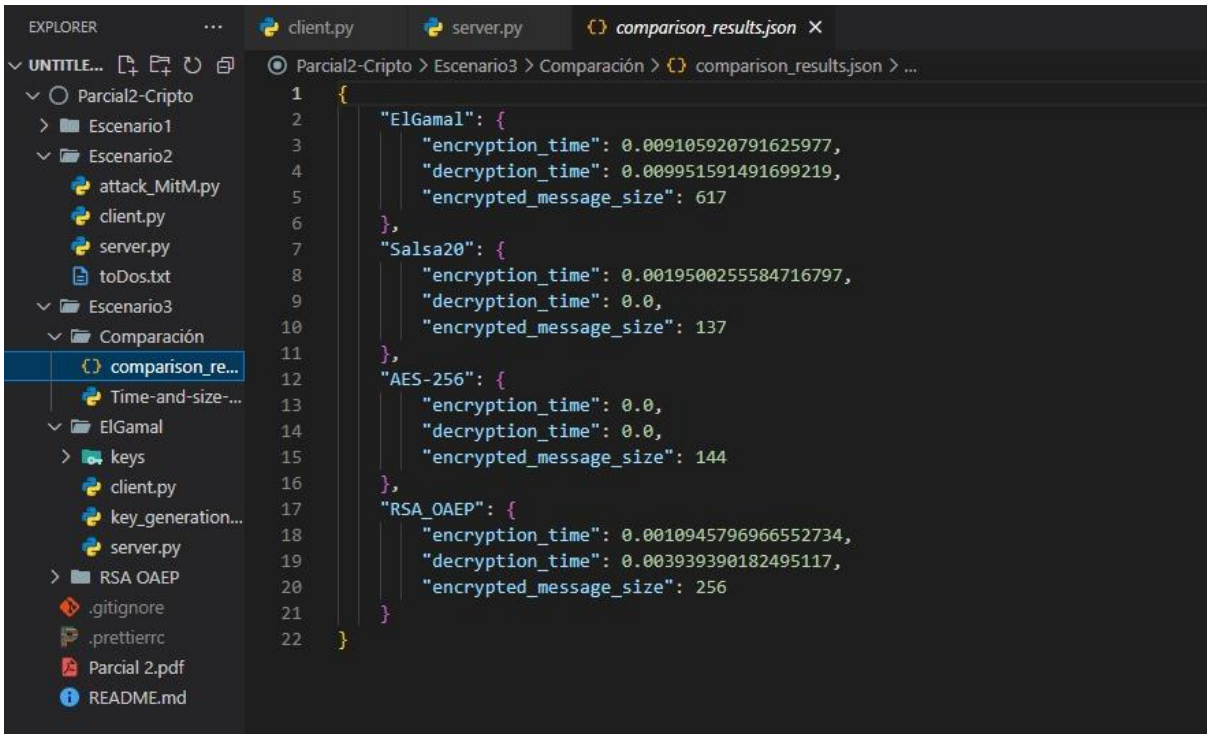
Consola server

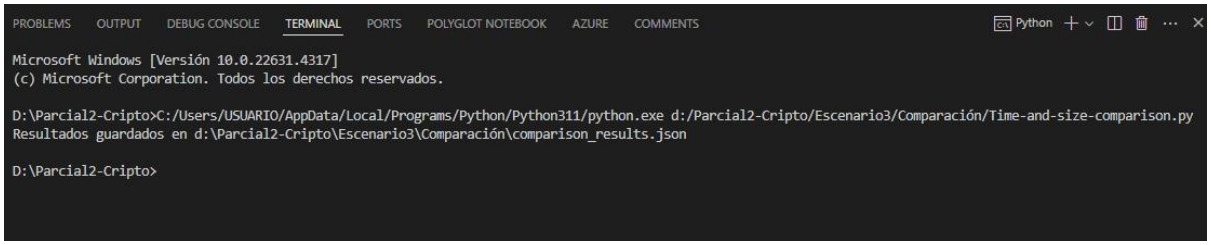
```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS                                                    Python

PS C:\Users\ASUS> & C:/Users/ASUS/AppData/Local/Programs/Python/Python312/python.exe c:/Users/ASUS/Downloads/Parcial2-Cripto-main/Parcial2-Cripto-main/Escenario2/server.py
Servidor en espera de conexiones...
Conexión establecida con: ('192.168.1.14', 52383)
Llave simétrica acordada con el cliente: 14eb4f06ee0c9c24166148351496880be260e05df4d12fdeee03330afaf50eda
Mensaje del cliente: Hola
Escribe tu respuesta al cliente (o 'exit' para salir): Bienvenido señor usuario
Mensaje del cliente: 123456
Escribe tu respuesta al cliente (o 'exit' para salir): 789 10
PS C:\Users\ASUS>
```

Consola cliente

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\ASUS> & C:/Users/ASUS/AppData/Local/Programs/Python/Python312/python.exe c:/Users/ASUS/Downloads/Parcial2-Cripto-main/Parcial2-Cripto-main/Escenario2/client.py
Llave simétrica acordada con el servidor: 29547c20b508c88324df861edad012d0e7db2ce6d12cd0e117fd214208b15bda
Escribe tu mensaje al servidor (o 'exit' para salir): Hola
Respuesta del servidor: Bienvenido señor usuario
Escribe tu mensaje al servidor (o 'exit' para salir): 123456
Respuesta del servidor: 789 10
Escribe tu mensaje al servidor (o 'exit' para salir): exit
PS C:\Users\ASUS>
```

Escenario3


EXPLORER
client.py
server.py
comparison_results.json
Parcial2-Cripto > Escenario3 > Comparación > comparison_results.json > ...
Parcial2-Cripto
Escenario1
Escenario2
attack_MitM.py
client.py
server.py
toDos.txt
Escenario3
Comparación
comparison_re...
Time-and-size-...
ElGamal
keys
client.py
key_generation...
server.py
RSA OAEP
.gitignore
.prettierrc
Parcial 2.pdf
README.md
{
"ElGamal": {
"encryption_time": 0.009105920791625977,
"decryption_time": 0.009951591491699219,
"encrypted_message_size": 617
},
"Salsa20": {
"encryption_time": 0.0019500255584716797,
"decryption_time": 0.0,
"encrypted_message_size": 137
},
"AES-256": {
"encryption_time": 0.0,
"decryption_time": 0.0,
"encrypted_message_size": 144
},
"RSA_OAEP": {
"encryption_time": 0.0010945796966552734,
"decryption_time": 0.003939390182495117,
"encrypted_message_size": 256
}
}


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
POLYGLOT NOTEBOOK
AZURE
COMMENTS
Python
Microsoft Windows [Versión 10.0.22631.4317]
(c) Microsoft Corporation. Todos los derechos reservados.
D:\Parcial2-Cripto>C:/Users/USUARIO/AppData/Local/Programs/Python/Python311/python.exe d:/Parcial2-Cripto/Escenario3/Comparación/Time-and-size-comparison.py
Resultados guardados en d:\Parcial2-Cripto\Escenario3\Comparación\comparison_results.json
D:\Parcial2-Cripto>