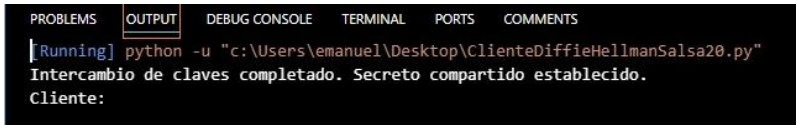
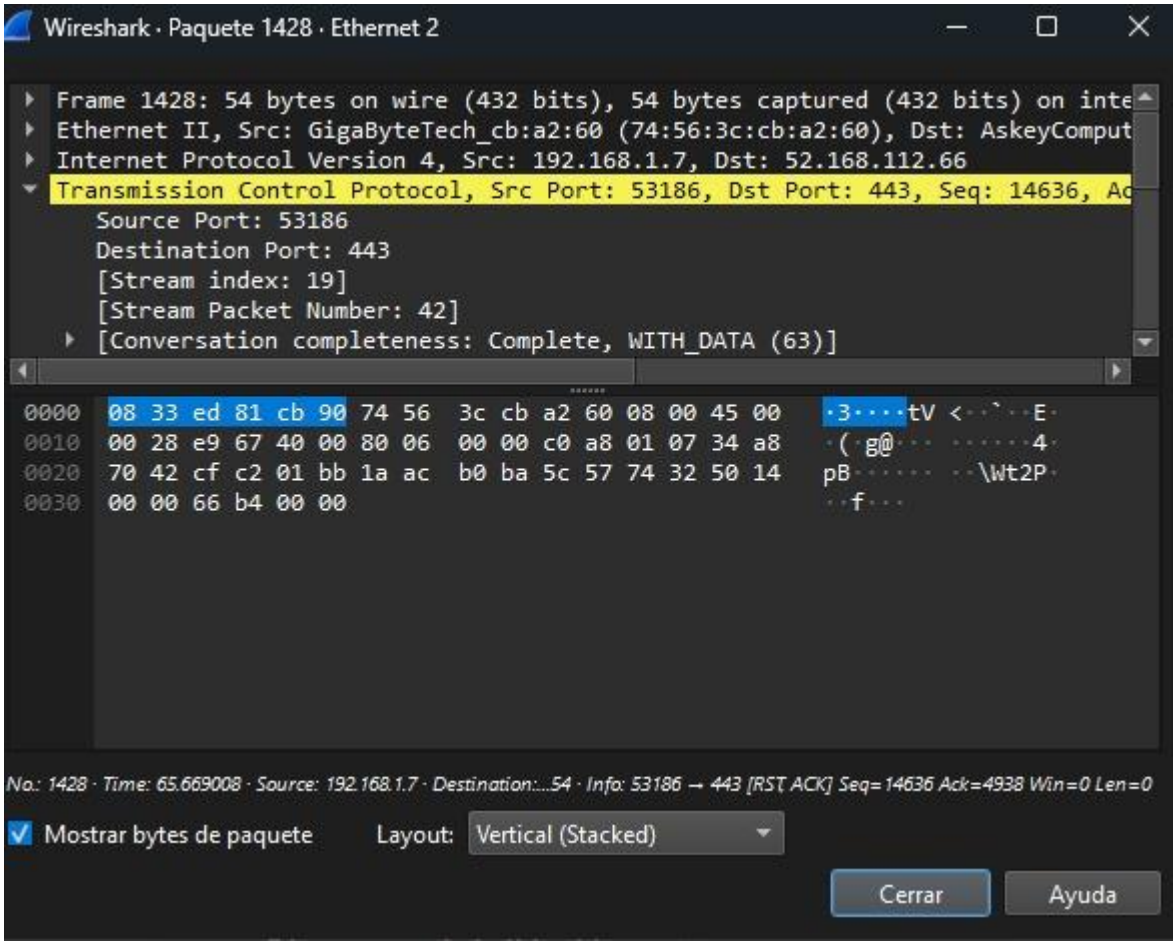
Análisis

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?

El algoritmo de "pasos de bebé, pasos de gigante" es un método para resolver el problema del logaritmo discreto en tiempo $O(p)$, donde p es el tamaño del grupo en el que se realiza el intercambio de llaves. Sin embargo, varios factores influyen en la dificultad de resolver este problema:

- Tamaño del número primo p: Un valor grande de p incrementa exponencialmente la cantidad de cálculos que el atacante necesita realizar, lo

que hace más difícil y lento resolver el logaritmo discreto. Los valores pequeños de *p*, como en este ejemplo, pueden ser vulnerables a ataques computacionales en un tiempo razonable.

- Elección del generador *g*: Si el generador *g* no está bien elegido (por ejemplo, si *g* es un número pequeño o tiene ciertas propiedades que faciliten las operaciones), puede simplificar la resolución del logaritmo discreto.
- Recursos computacionales: La potencia de cálculo disponible para el atacante influye significativamente en el tiempo necesario para resolver el problema. Aunque el algoritmo de "pasos de bebé, pasos de gigante" es más eficiente que otros métodos como la fuerza bruta, sigue siendo costoso en términos de tiempo y espacio.
- Resultados del ataque: En nuestro ataque, usando un pequeño valor de *p* y el generador *g=5*, logramos resolver el logaritmo discreto utilizando el algoritmo "Baby-step, Giant-step". Esto permitió al atacante derivar la clave simétrica compartida entre el cliente y el servidor y, posteriormente, descifrar los mensajes cifrados con Salsa20. Este resultado demuestra que el uso de parámetros criptográficos pequeños puede comprometer la seguridad del intercambio de llaves Diffie-Hellman.

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)?

Beneficios de Diffie-Hellman sobre $Fp*$:

- Seguridad basada en el problema del logaritmo discreto: El protocolo Diffie-Hellman ofrece una seguridad probada basada en la dificultad de resolver el problema del logaritmo discreto, que sigue siendo un problema difícil de resolver cuando los parámetros (especialmente *p*) son lo suficientemente grandes.
- Intercambio seguro sin necesidad de intercambiar claves privadas: Una de las principales ventajas de Diffie-Hellman es que permite a las partes establecer una clave secreta compartida sin necesidad de enviar ninguna información confidencial a través del canal (solo se intercambian claves públicas).
- Eficiencia en términos de computación: En comparación con otros algoritmos asimétricos como RSA, Diffie-Hellman es relativamente eficiente en términos de la cantidad de operaciones matemáticas necesarias para generar la clave compartida.

Desventajas de Diffie-Hellman sobre $Fp*$:

- Vulnerabilidad a ataques de Hombre en el Medio (MitM): A menos que se implemente una autenticación de claves públicas (por ejemplo, usando

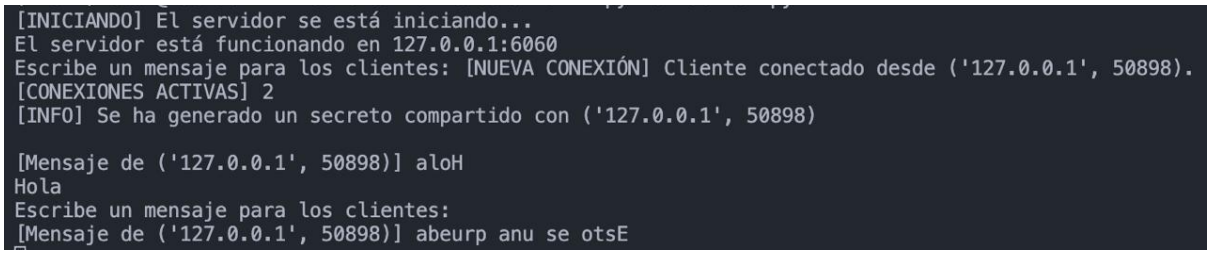
certificados), Diffie-Hellman es vulnerable a ataques MitM, en los que un atacante podría interceptar y modificar el intercambio de claves sin ser detectado.

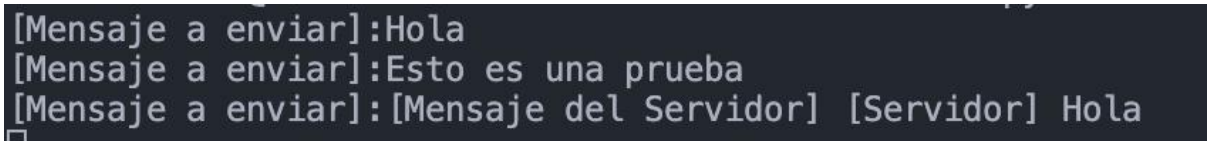
- Dependencia de la elección de *p* y *g*: La seguridad de Diffie-Hellman depende en gran medida de la elección de un primo *p* grande y un generador adecuado *g*. Si los valores son pequeños o mal elegidos, el protocolo se vuelve vulnerable a ataques de logaritmo discreto, como el que se realizó en este escenario.

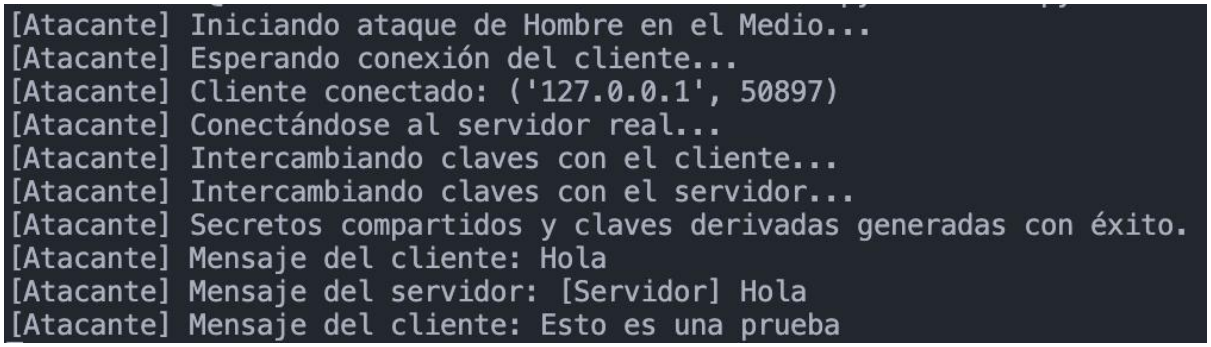
Comparación con otros métodos de intercambio de llaves:

- RSA (Rivest-Shamir-Adleman): RSA también permite el intercambio seguro de información cifrada, pero se basa en la dificultad de factorizar números grandes, en lugar del problema del logaritmo discreto. RSA tiene la ventaja de ser ampliamente compatible y fácil de implementar, pero su proceso de cifrado y descifrado es más lento que Diffie-Hellman, especialmente cuando se usan claves de gran tamaño.
- Intercambio de llaves con Curvas Elípticas (ECDH): El protocolo de intercambio de llaves basado en Curvas Elípticas (ECDH) es una alternativa que mejora significativamente la eficiencia y seguridad en comparación con Diffie-Hellman tradicional. ECDH utiliza un espacio de claves más pequeño, pero ofrece la misma seguridad, lo que reduce el tiempo de cómputo y la cantidad de datos transmitidos. Además, es más resistente a ataques de logaritmo discreto que Diffie-Hellman sobre $Fp*$.

Escenario 2:







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

El ataque de hombre en el medio en Diffie-Hellman sobre curvas elípticas tiene algunas vulnerabilidades que pueden ser aprovechadas. Por ejemplo, en este tipo de ataques, un hacker puede meterse en el intercambio de claves públicas entre dos personas, cambiarlas y crear claves diferentes con cada una de las partes. Así, puede espiar, modificar o incluso volver a cifrar los mensajes sin que nadie lo note. El problema principal es que Diffie-Hellman no autentica quién está al otro lado, lo que facilita este tipo de ataque.

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

- Autenticación de las claves públicas: Puedes usar certificados digitales y una infraestructura de clave pública (PKI) para estar seguro de que las claves públicas que se están intercambiando no vienen de un hacker, sino de alguien confiable.
- Firmas digitales: Las claves públicas se pueden firmar digitalmente con una clave privada conocida. Luego, se verifica la firma con la clave pública correcta. Esto garantiza que las claves no hayan sido modificadas por nadie más.
- Protocolos seguros: Usar protocolos como TLS (Transport Layer Security), que combina Diffie-Hellman con autenticación y cifrado robusto, asegura que las claves públicas estén autenticadas y la comunicación protegida contra ataques de hombre en el medio.
- Verificación de integridad: Mecanismos como HMAC (Hash-based Message Authentication Code) aseguran que los mensajes no sean cambiados mientras viajan.
- Perfect Forward Secrecy (PFS): PFS asegura que, incluso si alguna clave privada a largo plazo se ve comprometida en el futuro, la clave de sesión generada durante el intercambio de claves siga siendo segura.

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.

```
-------- RSA OAEP --------
Longitud mensaje cifrado: 1024 bits
Tiempo cifrado: 19.877 ms
Tiempo descifrado: 47.559 ms

-------- ElGamal --------
Longitud mensaje cifrado: 1024 bits
Tiempo cifrado: 20.950 ms
Tiempo descifrado: 10.879 ms
```

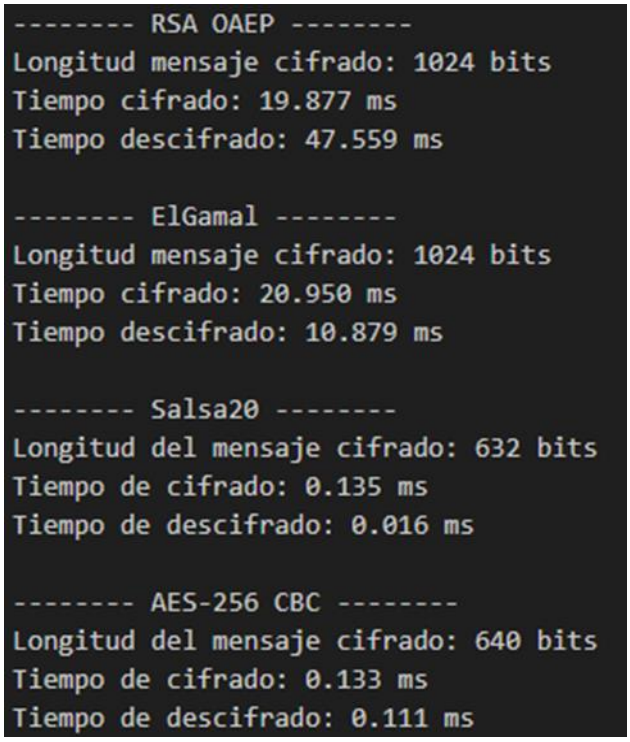
En los resultados, el cifrado con RSA OAEP es similar al cifrado con ElGamal, aunque en ElGamal el cifrado puede tomar más o menos tiempo dependiendo de los parámetros utilizados. Donde vemos una mayor diferencia es en el descifrado que es significativamente más lento en RSA OAEP. Esto es característico de RSA, ya que el proceso de descifrado implica una mayor carga computacional debido al uso de claves privadas más largas.

Ahora en cuanto al tamaño del mensaje ya que es el mismo, no habría diferencia en teoría, pero analizando el mensaje cifrado, se puede observar que en ElGamal es un poco más largo el mensaje cifrado:

```
PS C:\Users\User\Documents\CRIPTOGRAFÍA\ESCENARIO3_P2> python rsa_oaep.py
Cargando llaves RSA_OAEP.
Mensaje cifrado: b'\x8e\xf5TL\xcfu\xed\x0e\xac\xf4e\xe5U\x82\x88\xdeq\x7f\xec%^yR\xde\xf5g\x03X\\\x8eEc6\x11_\xf3\xe2\xd1\xe8\x99+}\x1dN\x13\xdb\xb8\xd4\xd2p\x1aE-\xe7\x17\xb23s\x8f\xc5^\xf7\xe7\xb6e\xaa\xd1\xbe\xef\xf1l\x0bAj\x9b\xaai\x10\x1e\xcbS\xc2h\x08\x81\x06\n\xfa\xd3>\x86\xa6\xfc_\x19\xa4q\x07\xb7\xc1\xa7$\x7f/\x16]y3\xda\xf1?%k&\xcf\xbeX\x0c\xb1x\x1e\x00\xdf2\x87{\x1b\xd6'
Mensaje descifrado: Hello RSA_OAEP supongo que sabes cifrar
PS C:\Users\User\Documents\CRIPTOGRAFÍA\ESCENARIO3_P2> python elgamal.py
Cargando llaves ElGamal.
Parte pública (a): b'n\xd2\xe4\xf5\xaa\xd6\x16u09\x0c\xf7SOW*\xee\xd5.\x0e^\x02s\xce\t|?\xd3\xe6\xbf\xa7\xf7\xc4\x82$"\\\x9e\xaa\xac\x80\x10!\xa00\xb7u\xe4\x8e8\xef\x1c\x04)\x1f^rq}Hpy\xf1\x9e\xa2\x1ab\xe7\x86$\x0b\xdad\x86\xbd\xe15\xd9\x96\x87\xd0/\xcc\x88\x87\x10\xc1\x87\x14u\xcc\x14\x03mw\x9f\x034\x871\xa5?\x7f\xde\x87]\xae\x07\xb8\xba\x02}\xd0\xa5\x01\xcd\xfdc\xb3\xde\xe5\xb1\x84\xdf4\x86"I'
Mensaje cifrado (b): b'\xc4?\xe0\xc5\xe2\xc5hK\xe2\x0c\xa4\xe8\xf1\xf8ZM\x84\x9a\x99\x11\x1b\x1d\xee\xe6\xfa\x86\x84j\xbd\xcc\x86.\xb0\xc1\xde\xfa\x96K\xd1\xbefw#\x1fD4\xb03K\xe9\xfd\x05Y\x0c\x19\xa8]Q0\xde\xa9\x13h\xe0\xbe:\xe5\xf2q\xfa\x06\xa4c%\xcf\xf4i1:\xef\xbb\xab\xb7Ho\x0b~\x7f1\xfc\xd3\xaf\xb6\xbe9\xbd\x97\xa6\xf3\xcf\xacG\x97\xbb/\x8aBC\x8f$] 3\x86&\xdb"b\n\xbccs\xc1\xf2\xb7\x1a\x90\xee'
Mensaje descifrado: Este es el mensaje para cifrar con ElGamal
```

Y por último en temas de seguridad, RSA OAEP se basa en el problema de la factorización de enteros grandes, mientras que ElGamal depende del problema del logaritmo discreto, ambos considerados problemas complejos, sin embargo, RSA OAEP incorpora un relleno de OAEP que proporciona una capa adicional de seguridad, reduciendo las vulnerabilidades a ataques basados en texto cifrado.

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?



Como conclusión en general sobre los cifradores simétricos vs asimétricos podemos observar que RSA OAEP y ElGamal (los cifradores asimétricos) son más lentos en comparación con Salsa20 y AES-256 (los cifradores simétricos), lo cual es esperado ya que al ver cómo funcionan los cifradores asimétricos, podemos darnos cuenta de que estos involucran operaciones matemáticas más complejas. También se observa que en comparación de los 2 cifradores asimétricos en términos de tiempo de cifrado no hay mucha diferencia para un mensaje de la misma longitud, pero el tiempo de descifrado si hay una diferencia, que, aunque es en milisegundos, igualmente la hay; caso contrario a los cifradores simétricos, que los probados en este caso manejan tiempos de cifrado y descifrado casi similares, la diferencia no es tan grande en términos de milisegundos respecto a los asimétricos.

Conclusiones

RSA OAEP es más conveniente para mensajes pequeños y rápidos donde el cifrado debe ser ágil y el tamaño del mensaje no debe aumentar mucho. ElGamal sería más adecuado cuando la eficiencia en el descifrado sea crítica. Tanto RSA OAEP como ElGamal requieren un intercambio seguro de claves públicas. Para evitar ataques de intermediarios (como ataques Man-in-the-Middle), es crucial usar certificados digitales emitidos por una autoridad de certificación (CA). Esto garantiza que las claves públicas pertenecen realmente a las entidades con las que se desea comunicar. En aplicaciones donde se intercambian grandes volúmenes de datos, es recomendable utilizar algoritmos de cifrado simétrico (como AES-256 o Salsa20). Estos algoritmos son mucho más rápidos en cifrado y descifrado, como se observó en los tiempos de ejecución.

Link GitHub: https://github.com/ema0108/Cryptografia.git