Tabla de Contenido

1. Introducción 2
2. Problema principal (Incluye posibles ataques) 2
3. Intercambio de llaves con ECDH 2
4. Uso de PSK para asegurar el entorno 3
4.1 Motivación de uso de las PSK 3
4.2 ¿Que son las PSK? 3
4.3 Modos de compartición de las PSK 3
4.4 Autenticación con PSK 4
5. Estrategia de cifrado 5
La estrategia de cifrado de la solución propuesta consiste en dividir el archivo en bloques más pequeños y cifrarlos individualmente utilizando el algoritmo AES en modo GCM. A continuación, se explica detalladamente las partes del cifrado de datos. 5
Se propone un entorno seguro inicializando el RNG que provee de generación segura de números aleatorios para establecer vectores de inicialización (IV) seguros para cada uno de los bloques de datos a cifrar. Luego, el archivo de entrada se divide en bloques de tamaño uniforme. Este tamaño se calcula en función de: 5
5.1 PSK + ECCDH 6
5.2 Optimización del encriptado con threads 7
6. Estrategia de descifrado 7
7. Estrategia para uso de llaves dinámicas 8
8. Librerías utilizadas 8
9. Estrategia de verificación y validación usada para medir la calidad interna del código 9
9.1 Pruebas de rendimiento: por threads 9
9.1.1 100MB 9
9.1.2 500MB 9
9.1.3 1GB 9
9.1.4 5GB 9
9.1.5 10GB 10
10. Referencias 11

PowerCoders

Autores:

Diego Alejandro Pulido

Danny Camilo Muñoz Sanabria

Johan Bautista

Juan Camilo López

Solución en seguridad y eficiencia para comunicaciones satelitales

1. Introducción

Las soluciones de criptografía generalmente toman bastantes recursos computacionales para llegar a cumplir el objetivo brindar comunicaciones seguras. En general, los métodos de comunicación usan certificados y algoritmos de encriptación asimétricos que son bastante costosos para asegurar la comunicación entre dispositivos (Véase el funcionamiento de TLS1.2). Sin embargo, no todos los dispositivos tienen mucho poder de cómputo para asegurar comunicaciones seguras, por lo que se necesitan otras estrategias óptimas para asegurar comunicaciones en dispositivos con pocos recursos. Por esta razón, en este documento explicamos nuestra propuesta de solución que brinda comunicación segura para dispositivos de bajo poder de cómputo como los satélites, usando algoritmos criptográficos eficientes como AES256/GCM y estrategias como el uso de PSK(Pre-Shared-Keys) para autenticación de los dispositivos.

2. Problema principal (Incluye posibles ataques)

Antes de empezar con la solución, es necesario repasar los posibles problemas que fueron pieza importante para el diseño de nuestra solución. El primer problema es bastante claro, y corresponde al límite computacional propuesto por el reto, de este principal problema subyacen los demás. Entre ellos se encuentran:

1. ¿Como podíamos optimizar el intercambio con Diffie-Hellman?
2. ¿Como hacíamos que el intercambio de llaves Diffie-Hellman se compartiera en un entorno inseguro (evitar Man-In-The-Middle)?
3. ¿Como podemos evitar uso de certificados digitales con llaves asimétricas?
4. ¿Como optimizamos el encriptado y desencriptado a pesar de usar AES256/GCM?

Estos problemas los resuelve nuestra solución a la perfección, y a lo largo de este documento vamos a explicar a detalle la solución de cada una.

3. Intercambio de llaves con ECDH

Como se propuso anteriormente, intercambiar las llaves con Diffie-Hellman era un problema por dos razones, el primero su eficiencia y el segundo como compartirlas en un entorno inseguro. Para atacar el problema de la eficiencia, fue necesario buscar algoritmos que usaran

Curvas Elípticas como ECDH (Elliptic Curve Diffie–Hellman), ya que nos proporcionaban dos grandes ventajas de rendimiento y seguridad. En cuanto al rendimiento, la generación de claves con curvas elípticas utiliza multiplicaciones de puntos en lugar de operaciones modulares. Estas últimas son, en principio, más costosas para los computadores, lo que hace que las curvas elípticas sean perfectas para nuestra solución. Como se menciona en Blue Goat Cyber (2022), "The elegance of its mathematical framework allows for streamlined key generation and exchange processes, making it ideal for resource-constrained environments such as IoT devices and mobile applications."

Además de ser rápido, como se menciona en Blue Goat Cyber (2022), "ECDH offers several advantages over traditional key exchange methods. For starters, it provides a higher level of security with shorter key lengths. This means faster computations and reduced computational overhead." Esto significa que, aunque se usen llaves más pequeñas, ECDH es más seguro. Por ejemplo, una llave de 256 bits generada por ECDH es mucho más fuerte que una llave de 256 bits generada por Diffie-Hellman común con aritmética modular. En general, decidimos usar ECDH para generar un secreto compartido más rápido, eficiente y seguro.

4. Uso de PSK para asegurar el entorno

4.1 Motivación de uso de las PSK

Hasta acá, ya tenemos las llaves generadas con ECDH. Sin embargo, como antes mencionamos hay un problema, puesto que las compartimos en un entorno inseguro. Cualquiera que este escuchando la transmisión entre el satélite y la base, puede hacerse pasar por cualquiera de los dos, interceptando la comunicación, un ataque conocido como Man-In-The-Middle (MitM).

Para solucionar este problema, generalmente se recurre al uso clásico de certificados digitales, como en TLS 1.2 (véase " SSL, TLS, HTTPS Explained" de ByteByteGo, 2022). O también se utiliza infraestructura de clave pública (PKI) para autenticar los dispositivos. Sin embargo, el uso de certificados digitales en TLS 1.2 requiere llaves y encriptación asimétricas, que son la parte más pesada, lenta y complicada de cualquier protocolo seguro de autenticación. Por esta razón, recurrimos al uso de PSK como alternativa a los certificados digitales para autenticar ambos nodos de la comunicación, tal como se hace en TLS 1.3.

4.2 ¿Que son las PSK?

En general, vamos a autenticar los dispositivos con PSK, ¿pero que son? Las PSK, como su nombre indica, son llaves que se comparten antes de iniciar el HandShake entre el cliente y el servidor. Esto quiere decir que, el cliente y el servidor ya tienen acordada una llave que los identifica a ambos antes de enviarse información importante, al igual de cómo funciona un certificado. Todo ello, sin necesidad de intercambiar claves durante el proceso de establecimiento de la conexión.

4.3 Modos de compartición de las PSK

La compartición de las PSK antes de la comunicación se puede hacer de dos maneras: una es llamada "External PSK-Share" y la otra es llamada "Session Resumption". Para ser breves, solo vamos a explicar la que nosotros usamos, que es el método "External PSK-Share". El método

"Session Resumption" se puede consultar mejor en IBM (2024), que en resumen es el método usado en conexiones con TLS 1.3.

El método "External PSK-Share", como se menciona en Housley, Hoyland, Sethi y Wood (2022), “Las PSK externas son claves secretas simétricas que se proporcionan a la implementación del protocolo TLS como entradas externas. Los PSK externos se proporcionan fuera de banda. *Traducido*”. Hay varios métodos para proporcionar las PSK fuera de banda, que se explican mejor en la sección 5 "External PSKs in Practice" del mismo documento. Sin embargo, nosotros decidimos implementar el método cargando la PSK manualmente en la memoria de ambos dispositivos, ya que es mucho más eficiente para la comunicación entre el satélite y la base, y además es más seguro que cualquier otra posible opción.

4.4 Autenticación con PSK

Hasta este punto, ya sabemos que son las PSK y como las pueden conocer el cliente y el servidor con anterioridad, pero no hemos explicado porque funcionan como mecanismo de autenticación. Para una mejor compresión, usaremos la figura 1:

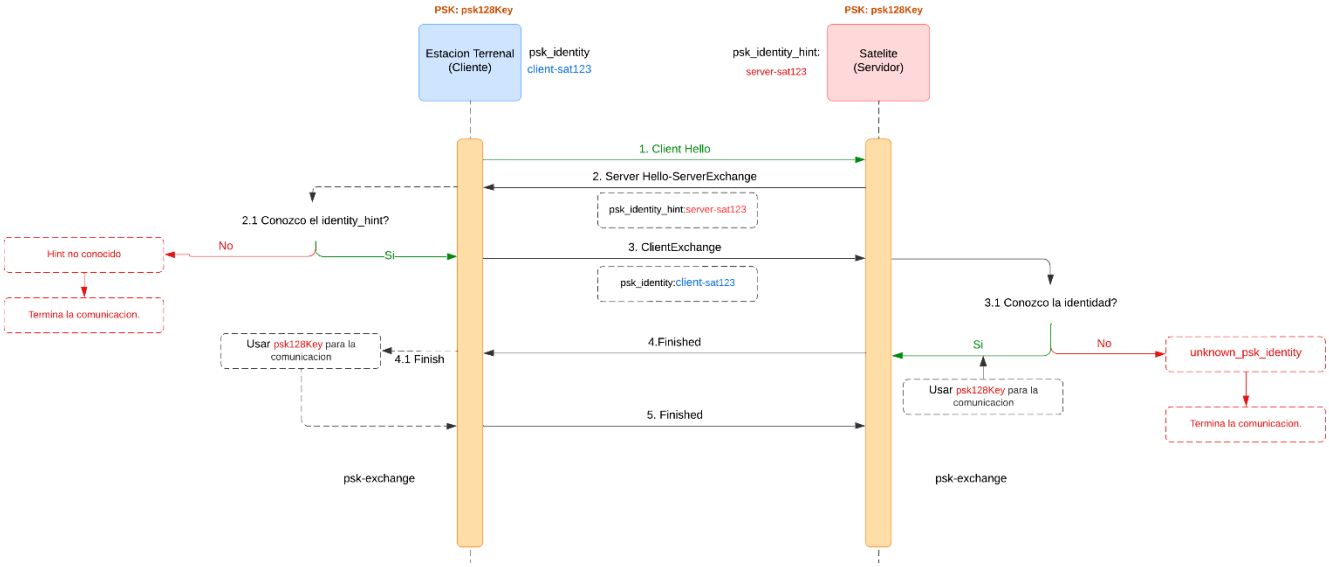


Figura 1: *Proceso de autenticación con PSK*

En orden de explicar la figura 1, profundizaremos cada parte de la comunicación. Empecemos con algunos conceptos básicos.

- **psk_identity:** Es la identidad del cliente, la identidad puede ser la ip del cliente, el DNS en caso de una web o simplemente un nombre predeterminado elegido por el cliente. En este caso la identidad se elige manualmente y corresponde a cliebt-sat123. La identidad del cliente se usa en ClienExchange para que el servidor la pueda autenticar.
- **psk_identity_hint:** Funciona como una ayuda para que el cliente pueda escoger la PSK correspondiente para la comunicación con el servidor que se quiere comunicar. Al igual que las PSK, el cliente debe conocer con anterioridad los identity_hint válidos, con el fin de que pueda asociar la PSK correspondiente con el servidor que conoce.

Aclaración: En la práctica común del uso de PSK, los servidores tienen un conjunto de identidades conocidas de clientes, y solo aceptan las que tienen guardadas o las que ya conocen. Pasa exactamente lo mismo con los clientes, tienen un conjunto de PSK y usan la que está asociada al hint que el servidor les proporciono. Como nuestro reto solo implica la comunicación de un cliente y un solo servidor, cualquier tercero que intente pasar hints o identidades diferentes, sera rechazado por la comunicacion, debido a que el cliente solo conoce un unico Hint y el servidor solo conoce una única identidad.

Sigamos ahora con cada comunicación:

1. **ClientHello:** Esta comunicación solamente es para indicarle al servidor que hay una comunicación entrante a responder.
2. **ServerHello-ServerExchange:** En esta oportunidad, el servidor responde a la solicitud entrante por el cliente, donde a la vez aprovecha esta comunicación para enviar su identity_hint con el fin de que el cliente elija la PSK correspondiente al servidor.
    2.1 **Validación del hint:** El Cliente internamente valida que si conozca el hint proporcionado por el servidor para elegir la PSK correspondiente. Si no lo conoce, termina la comunicación porque no puede autenticarlo.
3. **ClientExchange:** El cliente envía su identidad al servidor para que el servidor la pueda validar e identifique correctamente al cliente.
    3.1. **Validación de la identidad:** El servidor valida si conoce la identidad enviada por el cliente. Si la conoce, elige su PSK correspondiente y continua la comunicación. Si no lo hace, termina la comunicación, ya que no puede identificar al cliente
4. **Finish:** Una vez que ambos saben cuál PSK elegir, se mandan un mensaje de finalización.

Note que esta fase explicada solamente es la de autenticación, la que nos permite evitar ataques MitM en futuras conexiones seguras. Las PSK acordadas en la autenticación explicada, serán usadas en la estrategia de cifrado con ECDH, tal como se va a explicar a continuación.

5. Estrategia de cifrado

La estrategia de cifrado de la solución propuesta consiste en dividir el archivo en bloques más pequeños y cifrarlos individualmente utilizando el algoritmo AES en modo GCM. A continuación, se explica detalladamente las partes del cifrado de datos.

Se propone un entorno seguro inicializando el RNG que provee de generación segura de números aleatorios para establecer vectores de inicialización (IV) seguros para cada uno de los bloques de datos a cifrar. Luego, el archivo de entrada se divide en bloques de tamaño uniforme. Este tamaño se calcula en función de:

$$Min\left(\frac{Tamaño\ total\ del\ archivo\ a\ cifrar}{\#\ hilos\ disponibles\ para\ cifrado}, \frac{3GB}{\#\ hilos\ disponibles\ para\ cifrado}\right)$$

Asegurando que en ningún momento se cargue en memoria RAM más de 3gb de los datos a cifrar (respetando las limitaciones computacionales del sistema embebido para el que se diseñó la solución presentada). Por otro lado, si el archivo pesa menos de 3gb, se podrá hacer la división equivalente del tamaño total del archivo entre el número de threads. La optimización en términos de memoria-tiempo de ejecución mediante paralelismo con la que se cifra la información se explicará más adelante.

A continuación, se da un resumen del proceso de cifrado detallado:

1. Inicialización: se inicializa el generador de números aleatorios (RNG) y se prepara la clave criptográfica.
2. División de Datos: se calcula el tamaño de los bloques y se divide el archivo de entrada en estos bloques.
3. Cifrado Paralelo: cada hilo cifra un bloque de datos utilizando AES-GCM. Los IVs y tags de autenticación se generan para cada bloque.
4. Escritura Secuencial: los bloques cifrados se escriben en el archivo de salida en orden secuencial. Se sincroniza la escritura para asegurar el orden correcto de los bloques.

5.1 PSK + ECCDH

Las llaves secretas que vamos a utilizar para el modo de cifrado AES/GCM256 es una combinación entre las PSK de los nodos de la comunicación y las llaves generadas por ECC. Hay varios motivos del porque utilizar esta combinación. Primero, si la PSK es comprometida en el futuro, las sesiones pasadas permanecen seguras porque cada sesión utiliza claves de cifrados temporales generadas dinámicamente con ECDH. Segundo, aunque un atacante pueda conocer la PSK, sin la capacidad de realizar el intercambio Diffie-Hellman correctamente, no podrá derivar las claves de sesión. En Akhmetzyanova, Alekseev, Smyshlyaeva y Sokolov (2019), se recomienda encarecidamente el uso de Diffie-Hellman combinado con PSK, ya que evita ataques de tipo Impersonation attack y Selfie Attack explicados en el paper de Liliya Akhmetzyanova.

Las propiedades de generar una llave de sesion con ECDH y PSK son las siguientes:

- Las PSK deben tener una longitud de 16 bytes. Véase los requisitos en la sección 5.3 de Eronen y Tschofenig (2005).
- Como vamos a usar AES256-GCM, las llaves generadas por ECDH también tendrán una longitud 16 bytes.
- Una vez tengamos disponibles ambas llaves, las concatenamos y generamos una llave completa de 32 bytes para AES256-GCM.

LLave de sesión final:

AA BB CC DD EE FF 11 22 33 44 55 66 77 88 99 00 00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF

16 bytes ECDH 16 bytes PSK

Notese que como solo usamos 16 bytes de ECDH, la generación de llaves dinámicas va a ser mucho más optimo, ya que no tenemos que generar llaves de 32 bytes para usar AES256-GCM, lo que requiere menos poder de cómputo a la hora de generar llaves dinámicas con ECDH.

5.2 Optimización del encriptado con threads

Cada hilo cifrará su bloque de datos usando el algoritmo AES/GCM256, con los parámetros previamente generados. Además, este modo no solo cifra los datos y genera un tag de autenticación que se utilizará para verificar la integridad de los datos durante el descifrado. A continuación, cuando un hilo termina de cifrar, no podrá escribir en el archivo hasta que los hilos anteriores a él lo hagan, esto para garantizar que la escritura del cifrado sea correcta (ej: evitando que el bloque IV se escriba previo al bloque III). Es decir, para cifrar se usa un esquema paralelo, pero la escritura del mismo se debe hacer en ejecución secuencial.

Los beneficios de la optimización por threads planteada son:

Control del Tamaño del Bloque: dividir el archivo en bloques más pequeños permite manejar eficientemente archivos grandes sin necesidad de cargar todo el archivo en memoria (requerimiento crucial debido al límite estricto de memoria del sistema embebido).

Memoria Temporal por Hilo: cada hilo utiliza una cantidad fija y predecible de memoria para procesar su bloque de datos. Al asegurar que cada bloque y su procesamiento se mantengan dentro de límites específicos de memoria, se evita superar los 4 GB de RAM. Además, una vez el hilo haya escrito los datos cifrados en el archivo, se libera la memoria del mismo para que puedan ejecutarse nuevos threads. Es decir, la memoria se irá liberando dinámicamente a medida que avance la ejecución del programa.

Se logra una reducción del tiempo de procesamiento debido a la paralelización, ya que se reduce el tiempo necesario para cifrar grandes volúmenes de datos al distribuir el trabajo entre múltiples hilos. Esto gracias a asignar bloques de tamaño uniforme a los hilos, se evita la sobrecarga en cualquier hilo individual, garantizando un uso eficiente de los recursos del sistema.

6. Estrategia de descifrado

La estrategia de descifrado de la solución propuesta consiste en leer los datos cifrados en bloques, descifrarlos individualmente utilizando el algoritmo AES en modo GCM y verificar la integridad de los datos mediante los tags de autenticación.

El descifrado sigue el siguiente flujo: Inicialmente, se lee el archivo cifrado en bloques del mismo tamaño que los utilizados durante el proceso de cifrado. Para cada bloque, se leen el vector de inicialización (IV) y el tag de autenticación que fueron generados durante el cifrado. Luego, se crean múltiples hilos de ejecución para manejar el descifrado de los bloques de datos. Cada hilo se encarga de descifrar un bloque específico, la estrategia del paralelismo empleado es similar al cifrado, así como todos sus beneficios. De manera similar al cifrado, aunque el descifrado se realiza en paralelo, la escritura de los bloques descifrados en el archivo de salida se realiza de manera secuencial para asegurar que los bloques se escriban en el orden correcto. Finalmente, los bloques descifrados se escriben en orden y secuencialmente en un archivo de salida, recuperando los datos originales en su forma íntegra.

7. Estrategia para uso de llaves dinámicas

La estrategia implementada para el uso de llaves dinámicas se basa en la generación de una nueva llave para cada sesión de comunicación. Específicamente, cada imagen transmitida entre el satélite y la base tiene asignada una llave de encriptación y desencriptación única. Este enfoque tiene múltiples ventajas, principalmente en términos de seguridad y eficiencia.

Uno de los beneficios clave de esta estrategia es la **secrecía hacia adelante** (Forward secrecy). Esto significa que incluso si una llave de encriptación se ve comprometida en algún momento, la información en las sesiones anteriores o futuras no queda expuesta. Cada llave es válida solo para una sesión específica, por lo que la violación de una llave no afecta las comunicaciones pasadas ni futuras. Esto mejora significativamente la seguridad de las comunicaciones al minimizar el impacto potencial de una brecha de seguridad.

Además, las llaves utilizadas son generadas dinámicamente mediante el uso de **ECDH** (Elliptic Curve Diffie-Hellman) y tienen una longitud de 16 bytes, gracias a que los otros 16 bytes son elegidos por las PSK de ambos dispositivos. Este tamaño de llave es adecuado para garantizar un nivel adecuado de seguridad mientras mantiene la eficiencia operativa. Comparado con el uso de llaves completas de 32 bytes, las llaves de 16 bytes son mucho más ligeras en términos de procesamiento y almacenamiento. Esto resulta en una carga computacional reducida y una mayor velocidad en las operaciones de encriptación y desencriptación, lo cual es crucial para sistemas con recursos limitados como los satélites.

8. Librerías utilizadas

- **WolfSSL:** Hemos elegido la biblioteca wolfSSL para nuestro proyecto de encriptación en C++ que utiliza PSK, AES256 y GCM debido a su sólida seguridad y cumplimiento de estándares modernos, su rendimiento eficiente en sistemas con recursos limitados, y su amplia compatibilidad con diversas plataformas. La biblioteca ofrece una implementación optimizada de algoritmos criptográficos robustos y proporciona una

API bien documentada que facilita la integración y el desarrollo. Además, tiene varios módulos de implementación para dispositivos IoT y funciones de optimización de recursos para kernel de linux y CPU's ARM, convirtiéndose en una herramienta muy versátil que podemos seguir usando en siguientes retos.

- **GTest:** Esta librería se usó para hacer test-unitarios de las funciones críticas del programa, y de esta manera hacer mejor cobertura.

9. Estrategia de verificación y validación usada para medir la calidad interna del código

9.1 Pruebas de rendimiento: por threads

9.1.1 100MB

Tipo Archivo	**Tamaño** (MB)	**Threads**	**RAM consumida encriptar**	**RAM consumida desencriptar**	**Total RAM**	**Tiempo encriptación** (s)	**Tiempo Desencriptando** (s)
.tif	133,30	5,00	59904,00	56704,00	116608,00	1,29	1,28
.tif	133,30	10,00	33704,00	30592,00	64296,00	1,33	1,43
.tif	133,30	20,00	20864,00	23828,00	44692,00	1,40	1,44
.tif	133,30	50,00	13060,00	12116,00	25176,00	1,46	1,22

9.1.2 500MB

Tipo Archivo	**Tamaño** (MB)	**Threads**	**RAM consumida encriptar**	**RAM consumida desencriptar**	**Total RAM**
.tif	696,80	5,00	280064,00	276736,00	556800
.tif	696,80	10,00	143872,00	140736,00	284608
.tif	696,80	20,00	75904,00	72576,00	148480
.tif	696,80	50,00	34944,00	45376,00	80320

9.1.3 1GB

Tipo Archivo	**Tamaño** (MB)	**Threads**	**RAM consumida** encriptar	**RAM consumida desencriptar**	**Total RAM**
.jpg	1100,00	5,00	423040,00	419584,00	842624
.jpg	1100,00	10,00	215472,00	212204,00	427676
.jpg	1100,00	20,00	111616,00	108516,00	220132
.jpg	1100,00	50,00	49340,00	45952,00	95292

9.1.4 5GB

Tipo Archivo	**Tamaño** (MB)	**Threads**	**RAM consumida** encriptar	**RAM consumida desencriptar**	**Total RAM**

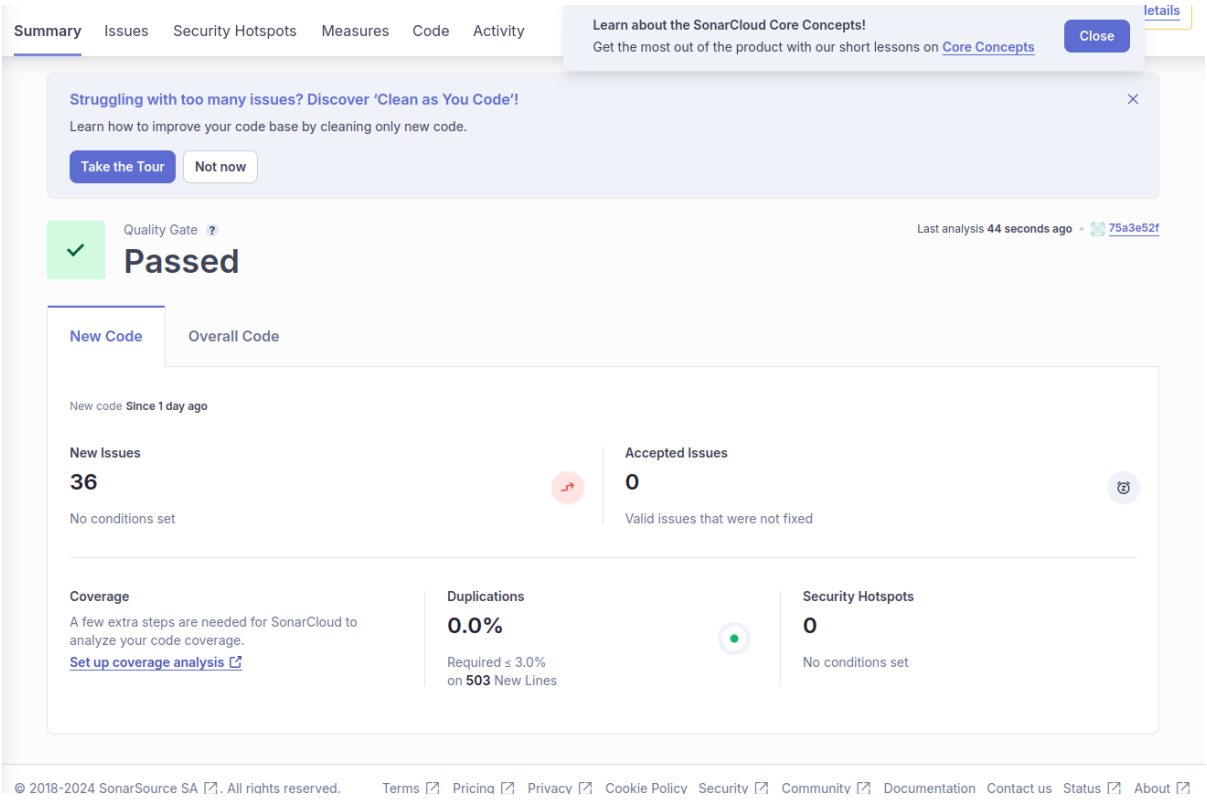
.jpg	5300	5	2083968,00	2080512,00	4164480,00
.jpg	5300	10	1045760,00	1042560,00	2088320,00
.jpg	5300	20	526848,00	526848,00	1053696,00
.jpg	5300	50	215456,00	212260,00	427716,00

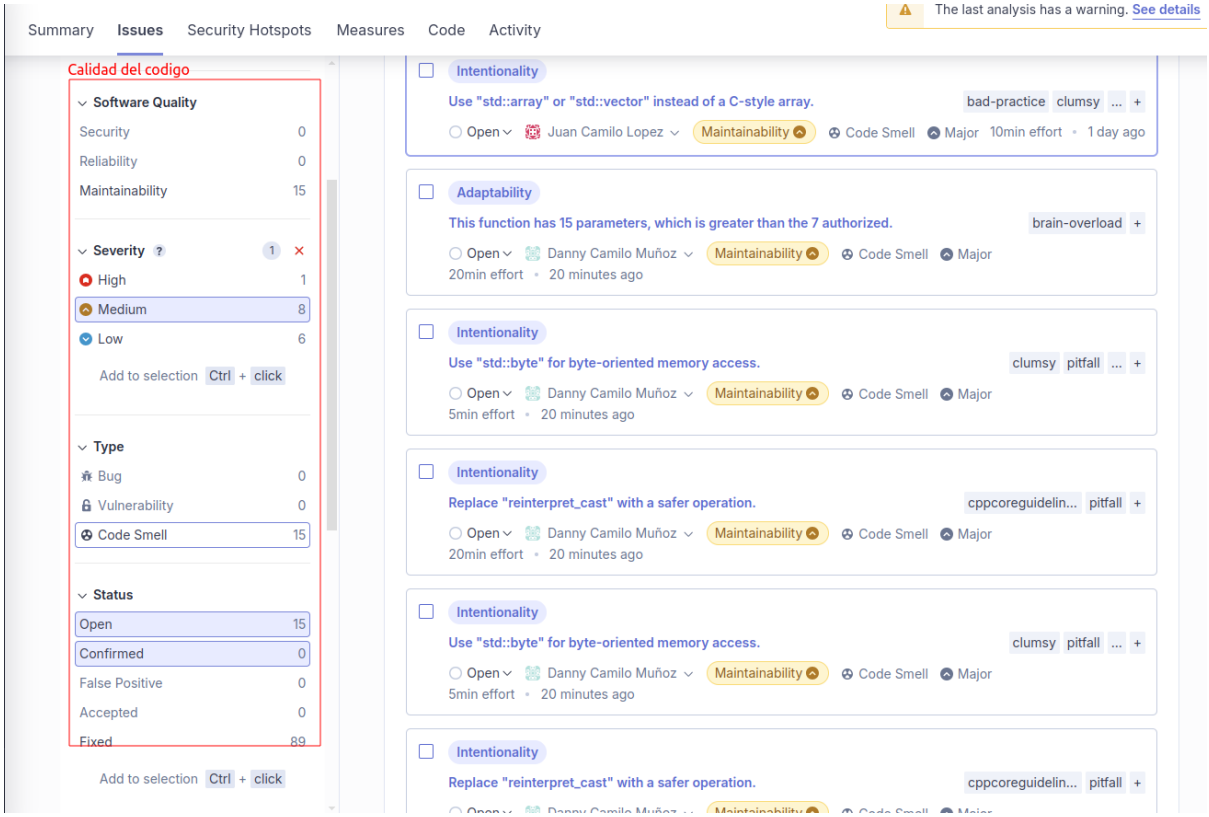
9.1.5 10GB

Tipo Archivo	Tamaño	Threads	RAM consumida	Tiempo encriptación	Tiempo Desencriptando
		5			
		10			
		20			
		30			

9.2 Verificación calidad interna del código

Como estrategia de validación, usamos el analizador estatico SonarCloud que nos muestra las siguientes métricas:



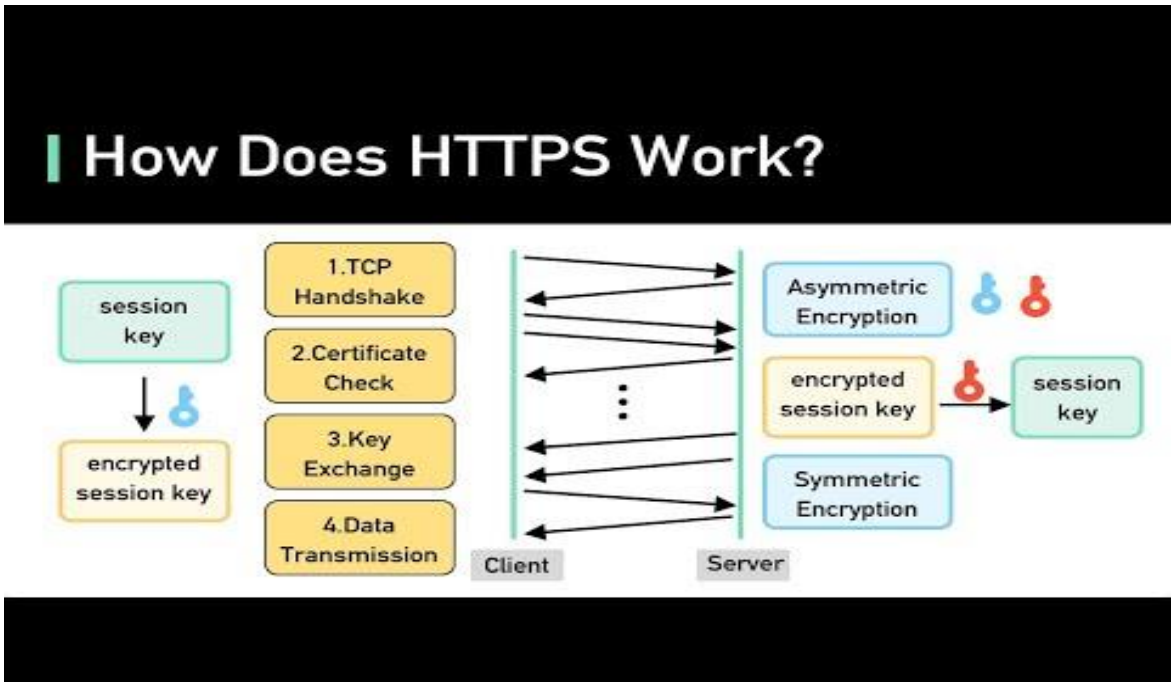


Igualmente, usamos la librería de test unitarios de código GTest de Google para verificar algunas de las funciones críticas de nuestra solución. Para revisarlas, asegúrese de revisar el readme.md del repositorio con las indicaciones propias para ver las pruebas realizadas.

10. Referencias

Blue Goat Cyber. (2022). *Elliptic Curve Diffie-Hellman (ECDH)*. Recuperado de https://bluegoatcyber.com/blog/elliptic-curve-diffie-hellman-ecdh/.

ByteByteGo. (2022, December 8). *SSL, TLS, HTTPS Explained* [Video]. YouTube. https://www.youtube.com/watch?v=j9QmMEWmcfo



IBM. (2024). *Session Resumption with Pre-shared Key*. Recuperado de https://www.ibm.com/docs/en/sdk-java-technology/8?topic=handshake-session-resumption-pre-shared-key.

Housley, R., Hoyland, J., Sethi, M., & Wood, C. A. (2022). *External PSK Usage Guidance*. RFC 9257. Internet Engineering Task Force (IETF). Recuperado de https://datatracker.ietf.org/doc/rfc9257/.

Eronen, P., & Tschofenig, H. (2005). *Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)*. RFC 4279. Network Working Group. Recuperado de https://www.rfc-editor.org/rfc/rfc4279.