Descripción de la Solución

Portada:

- **Título**: Solución de Cifrado y Descifrado con AES-256
- **Autor**: Astrovia
- **Fecha**: 28-09-2024

---

1. Estrategia de Cifrado

1.1 Algoritmo de Cifrado

Para el proceso de cifrado, se utiliza el algoritmo **AES-256 en modo CTR (Counter)**, un esquema de cifrado simétrico que cifra los datos en bloques de 128 bits. **AES-256** asegura una longitud de clave de 256 bits, lo que proporciona un nivel alto de seguridad. El modo CTR es eficiente porque convierte el cifrado en una operación similar a la de un flujo, permitiendo la manipulación de grandes cantidades de datos sin necesidad de padding.

1.2 Implementación del Cifrado

El cifrado se realiza en bloques de **1MB** para optimizar el uso de la memoria y evitar la sobrecarga en sistemas con memoria limitada. La función principal utilizada es `EVP_EncryptUpdate` proporcionada por OpenSSL, que realiza el cifrado de los datos:

```
EVP_EncryptUpdate(ctx, buffer_out, &cipher_len, buffer_in, len);
```

Cada bloque de datos es cifrado y escrito directamente en el archivo de salida. El proceso se completa llamando a `EVP_EncryptFinal_ex` para manejar cualquier bloque restante.

2. Estrategia de Descifrado

2.1 Algoritmo de Descifrado

El algoritmo de descifrado utilizado es el mismo que para el cifrado, **AES-256 en modo CTR**. Dado que se trata de un cifrado simétrico, se utiliza la misma clave y el mismo vector de inicialización (IV) que se usaron durante el cifrado.

2.2 Implementación del Descifrado

Para descifrar los datos, se utiliza la función `EVP_DecryptUpdate`, que es equivalente a la de cifrado, pero en este caso, recupera los datos originales a partir de los datos cifrados. Similar al cifrado, el archivo se procesa en bloques de 1MB para optimizar el rendimiento y el uso de memoria.

3. Estrategia para el Uso de Llaves Dinámicas

3.1 Generación Dinámica de Llaves e IVs

La clave simétrica y el IV no se almacenan en disco, sino que se generan de manera dinámica utilizando un **número de secuencia**. La función `generate_symmetric_key_and_iv` deriva la clave y el IV utilizando **HMAC-SHA256** con un **secreto compartido**. Este enfoque garantiza que la misma entrada (número de secuencia) producirá siempre la misma clave e IV, evitando la necesidad de almacenamiento local de claves.

3.2 Proceso de Generación de Clave

El siguiente fragmento de código muestra cómo se deriva la clave y el IV a partir del número de secuencia:

```
HMAC_CTX *hctx = HMAC_CTX_new();

HMAC_Init_ex(hctx, shared_secret, 32, EVP_sha256(), NULL);

HMAC_Update(hctx, salt, sizeof(salt));

HMAC_Final(hctx, key, NULL);
```

Este enfoque asegura que la clave sea **dinámica** y **segura**, ya que cada sesión puede tener un número de secuencia diferente.

4. Estrategia para Gestión de Memoria en Sistema Embebido

4.1 Procesamiento por Bloques

El diseño está optimizado para sistemas embebidos que tienen recursos de memoria limitados. En lugar de cargar todo el archivo en la memoria, se utiliza una estrategia de procesamiento por **bloques de 1MB**, lo que permite manejar archivos grandes sin comprometer la memoria del sistema.

4.2 Liberación de Recursos

Para evitar fugas de memoria, todos los recursos se liberan correctamente después de su uso. Las llamadas a **OpenSSL** como `EVP_CIPHER_CTX_new()` y `HMAC_CTX_new()` son seguidas por la correspondiente liberación de recursos:

```
EVP_CIPHER_CTX_free(ctx);

HMAC_CTX_free(hctx);
```

Esto es crucial para garantizar la estabilidad en sistemas embebidos, especialmente aquellos con recursos de memoria restringidos.

5. Librerías Utilizadas

5.1 OpenSSL

- **libcrypto**: Utilizada para las funciones criptográficas, como AES-256 y HMAC-SHA256.
- **libssl**: Proporciona soporte para la gestión de claves y funciones adicionales de seguridad.

5.2 Librerías Estándar de C/C++

- `stdio.h`, `string.h`: Para operaciones básicas de entrada/salida y manejo de cadenas.
- `iostream`, `cstdlib`: Para la gestión de entradas y salidas en C++.

Estas librerías proporcionan las funcionalidades esenciales para el manejo de archivos, criptografía y generación dinámica de claves.

6. Estrategia de Verificación y Validación

6.1 Verificación del Cifrado y Descifrado

La verificación de la funcionalidad del sistema se realiza comparando el archivo original y el archivo descifrado utilizando la función **`md5sum`** para asegurar que ambos son idénticos:

md5sum archivo_original.jpg archivo_descifrado.jpg

Si los **hashes MD5** coinciden, esto indica que el proceso de cifrado y descifrado es correcto y que no hubo corrupción de datos.

6.2 Validación del Código

- **Pruebas Unitarias**: Se realizaron pruebas unitarias para cada función crítica, como la generación de claves y el cifrado/descifrado.
- **Valgrind**: Se utilizó para verificar la gestión de memoria, asegurando que no hubiera **fugas de memoria** durante la ejecución del programa.

7. Conclusión

El sistema propuesto de cifrado y descifrado con **AES-256 en modo CTR** cumple con los estándares de seguridad y eficiencia. La generación dinámica de claves asegura un alto nivel de seguridad sin necesidad de almacenar claves en el sistema. Además, el diseño está optimizado para su uso en sistemas embebidos con recursos limitados, asegurando un rendimiento eficiente y estable.