#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <openssl/evp.h>

#include <openssl/aes.h>

#define KEY\_SIZE 32

int aes\_encrypt(FILE \*in\_file, FILE \*out\_file, unsigned char \*key, unsigned char \*iv) {

EVP\_CIPHER\_CTX \*ctx;

int len;

int out\_len;

unsigned char out\_buf[1024];

// Create and initialize the encryption context

ctx = EVP\_CIPHER\_CTX\_new();

EVP\_EncryptInit\_ex(ctx, EVP\_aes\_256\_cbc(), NULL, key, iv);

// Process the input file in blocks and write the encrypted blocks to the output file

while ((len = fread(out\_buf, 1, 1024, in\_file)) > 0) {

if (EVP\_EncryptUpdate(ctx, out\_buf, &out\_len, out\_buf, len) != 1) {

EVP\_CIPHER\_CTX\_free(ctx);

return 0;

}

fwrite(out\_buf, 1, out\_len, out\_file);

}

// Finalize the encryption process and write the final block to the output file

if (EVP\_EncryptFinal\_ex(ctx, out\_buf, &out\_len) != 1) {

EVP\_CIPHER\_CTX\_free(ctx);

return 0;

}

fwrite(out\_buf, 1, out\_len, out\_file);

// Free the encryption context and return success

EVP\_CIPHER\_CTX\_free(ctx);

return 1;

}

int aes\_decrypt(FILE \*in\_file, FILE \*out\_file, unsigned char \*key, unsigned char \*iv) {

EVP\_CIPHER\_CTX \*ctx;

int len;

int out\_len;

unsigned char out\_buf[1024];

// Create and initialize the decryption context

ctx = EVP\_CIPHER\_CTX\_new();

EVP\_DecryptInit\_ex(ctx, EVP\_aes\_256\_cbc(), NULL, key, iv);

// Process the input file in blocks and write the decrypted blocks to the output file

while ((len = fread(out\_buf, 1, 1024, in\_file)) > 0) {

if (EVP\_DecryptUpdate(ctx, out\_buf, &out\_len, out\_buf, len) != 1) {

EVP\_CIPHER\_CTX\_free(ctx);

return 0;

}

fwrite(out\_buf, 1, out\_len, out\_file);

}

// Finalize the decryption process and write the final block to the output file

if (EVP\_DecryptFinal\_ex(ctx, out\_buf, &out\_len) != 1) {

EVP\_CIPHER\_CTX\_free(ctx);

return 0;

}

fwrite(out\_buf, 1, out\_len, out\_file);

// Free the decryption context and return success

EVP\_CIPHER\_CTX\_free(ctx);

return 1;

}

int main(int argc, char \*argv[]) {

FILE \*in\_file, \*out\_file;

unsigned char key[KEY\_SIZE];

unsigned char iv[AES\_BLOCK\_SIZE] = {0};

// Check command line arguments

if (argc != 4) {

printf("Usage: %s <input file> <output file> <key>\n", argv[0]);

return 1;

}

// Open the input and output files

in\_file = fopen(argv[1], "rb");

if (!in\_file) {

printf("Error: could not open input file\n");

return 1;

}

out\_file = fopen(argv[2], "wb");

if (!out\_file) {

printf ("Error: could not open output file\n");

fclose(in\_file);

return 1;

}

// Check the length of the key

if (strlen(argv[3]) != KEY\_SIZE \* 2) {

printf("Error: key must be %d bytes long\n", KEY\_SIZE);

fclose(in\_file);

fclose(out\_file);

return 1;

}

// Convert the key from a hex string to binary

for (int i = 0; i < KEY\_SIZE; i++) {

sscanf(argv[3] + i \* 2, "%2hhx", &key[i]);

}

// Encrypt or decrypt the input file and write the result to the output file

if (aes\_encrypt(in\_file, out\_file, key, iv)) {

printf("Encryption successful\n");

} else if (aes\_decrypt(in\_file, out\_file, key, iv)) {

printf("Decryption successful\n");

} else {

printf("Error: encryption or decryption failed\n");

}

// Close the input and output files

fclose(in\_file);

fclose(out\_file);

return 0;

}

This code uses the OpenSSL library to perform the encryption and decryption. The `aes\_encrypt` and `aes\_decrypt` functions are responsible for encrypting and decrypting the input file, respectively. The main function takes three command line arguments: the input file name, the output file name, and the key as a hex string. The key is converted from a hex string to binary using the `sscanf` function.

here's an example of how the output might look when you run the program:

$ ./aes\_encrypt\_decrypt input\_file output\_file 0123456789abcdef0123456789abcdef

Encryption successful

In this example, input\_file is the name of the file you want to encrypt or decrypt, output\_file is the name of the file where you want to save the result, and 0123456789abcdef0123456789abcdef is the key as a hex string. The output indicates whether the encryption or decryption was successful.

2)

To perform AES 256 algorithm encryption and decryption for a file in C, you can use a cryptography library such as OpenSSL. Here's an example code snippet that demonstrates how to use OpenSSL to encrypt and decrypt a file using AES 256:

c

Copy code

#include <stdio.h>

#include <openssl/aes.h>

#define AES\_BLOCK\_SIZE 16

int encrypt\_file(const char \*input\_file, const char \*output\_file, const char \*key)

{

FILE \*in\_file = fopen(input\_file, "rb");

FILE \*out\_file = fopen(output\_file, "wb");

if (in\_file == NULL || out\_file == NULL) {

perror("Failed to open file");

return -1;

}

fseek(in\_file, 0L, SEEK\_END);

long file\_size = ftell(in\_file);

rewind(in\_file);

unsigned char iv[AES\_BLOCK\_SIZE];

AES\_KEY aes\_key;

if (AES\_set\_encrypt\_key(key, 256, &aes\_key) != 0) {

fprintf(stderr, "Failed to set encryption key\n");

return -1;

}

// Generate a random IV

if (RAND\_bytes(iv, AES\_BLOCK\_SIZE) != 1) {

fprintf(stderr, "Failed to generate IV\n");

return -1;

}

fwrite(iv, 1, AES\_BLOCK\_SIZE, out\_file);

unsigned char in\_buf[AES\_BLOCK\_SIZE];

unsigned char out\_buf[AES\_BLOCK\_SIZE];

int num\_read = 0;

int num\_encrypted = 0;

while ((num\_read = fread(in\_buf, 1, AES\_BLOCK\_SIZE, in\_file)) > 0) {

AES\_cbc\_encrypt(in\_buf, out\_buf, num\_read, &aes\_key, iv, AES\_ENCRYPT);

fwrite(out\_buf, 1, num\_read, out\_file);

num\_encrypted += num\_read;

}

fclose(in\_file);

fclose(out\_file);

if (num\_encrypted != file\_size) {

fprintf(stderr, "Failed to encrypt file\n");

return -1;

}

return 0;

}

int decrypt\_file(const char \*input\_file, const char \*output\_file, const char \*key)

{

FILE \*in\_file = fopen(input\_file, "rb");

FILE \*out\_file = fopen(output\_file, "wb");

if (in\_file == NULL || out\_file == NULL) {

perror("Failed to open file");

return -1;

}

unsigned char iv[AES\_BLOCK\_SIZE];

AES\_KEY aes\_key;

if (AES\_set\_decrypt\_key(key, 256, &aes\_key) != 0) {

fprintf(stderr, "Failed to set decryption key\n");

return -1;

}

if (fread(iv, 1, AES\_BLOCK\_SIZE, in\_file) != AES\_BLOCK\_SIZE) {

fprintf(stderr, "Failed to read IV\n");

return -1;

}

unsigned char in\_buf[AES\_BLOCK\_SIZE];

unsigned char out\_buf[AES\_BLOCK\_SIZE];

int num\_read = 0;

int num\_decrypted = 0;

while ((num\_read = fread(in\_buf, 1, AES\_BLOCK\_SIZE, in\_file)) > 0) {

AES\_cbc\_encrypt(in\_buf, out\_buf, num\_read, &aes\_key, iv, AES\_DECRYPT);

fwrite(out\_buf, 1, num\_read, out\_file);

num\_decrypted += num\_read;

}

fclose(in\_file);

fclose(out\_file);

return 0;

}

int main()

{

const char \*input\_file = "input.txt";

const char \*encrypted\_file = "encrypted.bin";

const char \*dec

yogzz2022@gmail.com

continue

2 / 2

c

Copy code

rypted\_file = "decrypted.txt";

const char \*key = "01234567890123456789012345678901"; // 256-bit key

if (encrypt\_file(input\_file, encrypted\_file, key) != 0) {

fprintf(stderr, "Failed to encrypt file\n");

return 1;

}

if (decrypt\_file(encrypted\_file, decrypted\_file, key) != 0) {

fprintf(stderr, "Failed to decrypt file\n");

return 1;

}

return 0;

}

In this example code, the `encrypt\_file` function takes the path to an input file, an output file, and a 256-bit key, and encrypts the input file using AES 256 CBC mode with a random IV. The encrypted file is written to the output file. The `decrypt\_file` function takes the path to an encrypted file, an output file, and the same key used to encrypt the file, and decrypts the input file using AES 256 CBC mode with the IV read from the encrypted file. The decrypted file is written to the output file. The `main` function shows how to use these functions to encrypt and decrypt a file. Note that this example code does not include error checking for simplicity.

3) Sure, here's an example implementation of AES-256 encryption and decryption for a file in C using the OpenSSL library:

c

Copy code

#include <openssl/aes.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define KEY\_SIZE 32 // AES-256 key size in bytes

#define BLOCK\_SIZE 16 // AES block size in bytes

void encrypt\_file(const char \*input\_file, const char \*output\_file, const unsigned char \*key) {

FILE \*in\_file = fopen(input\_file, "rb");

FILE \*out\_file = fopen(output\_file, "wb");

// Initialize the encryption key and IV

AES\_KEY aes\_key;

unsigned char iv[BLOCK\_SIZE];

memset(iv, 0, sizeof(iv));

AES\_set\_encrypt\_key(key, KEY\_SIZE \* 8, &aes\_key);

// Encrypt the file

unsigned char in\_buf[BLOCK\_SIZE];

unsigned char out\_buf[BLOCK\_SIZE];

int num\_bytes\_read, num\_bytes\_written;

while ((num\_bytes\_read = fread(in\_buf, 1, BLOCK\_SIZE, in\_file)) > 0) {

AES\_cfb128\_encrypt(in\_buf, out\_buf, num\_bytes\_read, &aes\_key, iv, &num\_bytes\_written, AES\_ENCRYPT);

fwrite(out\_buf, 1, num\_bytes\_written, out\_file);

}

// Close the files

fclose(in\_file);

fclose(out\_file);

}

void decrypt\_file(const char \*input\_file, const char \*output\_file, const unsigned char \*key) {

FILE \*in\_file = fopen(input\_file, "rb");

FILE \*out\_file = fopen(output\_file, "wb");

// Initialize the decryption key and IV

AES\_KEY aes\_key;

unsigned char iv[BLOCK\_SIZE];

memset(iv, 0, sizeof(iv));

AES\_set\_decrypt\_key(key, KEY\_SIZE \* 8, &aes\_key);

// Decrypt the file

unsigned char in\_buf[BLOCK\_SIZE];

unsigned char out\_buf[BLOCK\_SIZE];

int num\_bytes\_read, num\_bytes\_written;

while ((num\_bytes\_read = fread(in\_buf, 1, BLOCK\_SIZE, in\_file)) > 0) {

AES\_cfb128\_encrypt(in\_buf, out\_buf, num\_bytes\_read, &aes\_key, iv, &num\_bytes\_written, AES\_DECRYPT);

fwrite(out\_buf, 1, num\_bytes\_written, out\_file);

}

// Close the files

fclose(in\_file);

fclose(out\_file);

}

int main(int argc, char \*\*argv) {

if (argc != 4) {

printf("Usage: %s [encrypt/decrypt] [input file] [output file]\n", argv[0]);

return 1;

}

// Generate a random key

unsigned char key[KEY\_SIZE];

FILE \*urandom = fopen("/dev/urandom", "rb");

fread(key, 1, KEY\_SIZE, urandom);

fclose(urandom);

// Encrypt or decrypt the file

if (strcmp(argv[1], "encrypt") == 0) {

encrypt\_file(argv[2], argv[3], key);

printf("File encrypted with key:\n");

for (int i = 0; i < KEY\_SIZE; i++) {

printf("%02x", key[i]);

}

printf("\n");

} else if (strcmp(argv[1], "decrypt") == 0) {

decrypt\_file(argv[2], argv[3], key);

} else {

printf("Invalid operation: %s\n", argv[1]);

return 1;

}

return 0;

}

This code uses the

yogzz2022@gmail.com

continue

OpenSSL library to perform AES-256 encryption and decryption in C. It reads in the input file in blocks of 16 bytes (the AES block size), encrypts or decrypts each block using AES-CFB128 mode, and writes the output to the output file. The encryption and decryption functions take in the input file name, output file name, and the encryption/decryption key as parameters.

To use this code, compile it with the OpenSSL library and run it with the following command line arguments:

css

Copy code

./aes256 [encrypt/decrypt] [input file] [output file]

The first argument specifies whether to encrypt or decrypt the file. The second argument is the input file to encrypt/decrypt, and the third argument is the output file to write the encrypted/decrypted data to.

Note that this code generates a random encryption key using the /dev/urandom device. You may wish to modify this to use a user-provided key instead. Also, this code does not handle padding or other error conditions that may arise during encryption or decryption. You should test this code thoroughly to ensure it works correctly for your use case.

**AES**

Here is a sample implementation of AES 256 encryption and decryption for a file in C language using OpenSSL library:

#include <stdio.h>

#include <string.h>

#include <openssl/aes.h>

#define BLOCK\_SIZE 16

int encrypt\_file(const char\* input\_file, const char\* output\_file, const unsigned char\* key, const unsigned char\* iv) {

AES\_KEY aes\_key;

unsigned char buffer\_in[BLOCK\_SIZE], buffer\_out[BLOCK\_SIZE];

FILE\* fp\_in = fopen(input\_file, "rb");

FILE\* fp\_out = fopen(output\_file, "wb");

int bytes\_read, bytes\_written;

if (!fp\_in || !fp\_out) {

printf("Error opening file.\n");

return 0;

}

if (AES\_set\_encrypt\_key(key, 256, &aes\_key) != 0) {

printf("Error setting encryption key.\n");

return 0;

}

while ((bytes\_read = fread(buffer\_in, 1, BLOCK\_SIZE, fp\_in)) > 0) {

AES\_cbc\_encrypt(buffer\_in, buffer\_out, bytes\_read, &aes\_key, iv, AES\_ENCRYPT);

bytes\_written = fwrite(buffer\_out, 1, bytes\_read, fp\_out);

if (bytes\_written != bytes\_read) {

printf("Error writing to file.\n");

return 0;

}

}

fclose(fp\_in);

fclose(fp\_out);

return 1;

}

int decrypt\_file(const char\* input\_file, const char\* output\_file, const unsigned char\* key, const unsigned char\* iv) {

AES\_KEY aes\_key;

unsigned char buffer\_in[BLOCK\_SIZE], buffer\_out[BLOCK\_SIZE];

FILE\* fp\_in = fopen(input\_file, "rb");

FILE\* fp\_out = fopen(output\_file, "wb");

int bytes\_read, bytes\_written;

if (!fp\_in || !fp\_out) {

printf("Error opening file.\n");

return 0;

}

if (AES\_set\_decrypt\_key(key, 256, &aes\_key) != 0) {

printf("Error setting decryption key.\n");

return 0;

}

while ((bytes\_read = fread(buffer\_in, 1, BLOCK\_SIZE, fp\_in)) > 0) {

AES\_cbc\_encrypt(buffer\_in, buffer\_out, bytes\_read, &aes\_key, iv, AES\_DECRYPT);

bytes\_written = fwrite(buffer\_out, 1, bytes\_read, fp\_out);

if (bytes\_written != bytes\_read) {

printf("Error writing to file.\n");

return 0;

}

}

fclose(fp\_in);

fclose(fp\_out);

return 1;

}

int main() {

const char\* input\_file = "input.txt";

const char\* output\_file = "output.enc";

const char\* decrypted\_file = "output\_decrypted.txt";

unsigned char key[] = "01234567890123456789012345678901";

unsigned char iv[] = "0123456789012345";

if (!encrypt\_file(input\_file, output\_file, key, iv)) {

printf("Encryption failed.\n");

return 1;

}

if (!decrypt\_file(output\_file, decrypted\_file, key, iv)) {

printf("Decryption failed.\n");

return 1;

}

printf("Encryption and decryption successful.\n");

return 0;

}

note that this implementation uses AES-256 encryption with CBC mode and a 256-bit key. The key and IV are hardcoded in the example, but in practice they should be generated using a secure random number generator and securely stored. Additionally, error handling

5) AES WITH KEY AND IV GENERATED USING random nos

Sure! Here's an updated version of the code that generates a random key and IV using OpenSSL's RAND\_bytes() function:

Sure! Here's an updated version of the code that generates a random key and IV using OpenSSL's RAND\_bytes() function:

#include <stdio.h>

#include <string.h>

#include <openssl/aes.h>

#include <openssl/rand.h>

#define BLOCK\_SIZE 16

#define KEY\_SIZE 32

#define IV\_SIZE 16

int generate\_key\_and\_iv(unsigned char\* key, unsigned char\* iv) {

if (RAND\_bytes(key, KEY\_SIZE) != 1 || RAND\_bytes(iv, IV\_SIZE) != 1) {

printf("Error generating random bytes.\n");

return 0;

}

return 1;

}

int encrypt\_file(const char\* input\_file, const char\* output\_file, const unsigned char\* key, const unsigned char\* iv) {

AES\_KEY aes\_key;

unsigned char buffer\_in[BLOCK\_SIZE], buffer\_out[BLOCK\_SIZE];

FILE\* fp\_in = fopen(input\_file, "rb");

FILE\* fp\_out = fopen(output\_file, "wb");

int bytes\_read, bytes\_written;

if (!fp\_in || !fp\_out) {

printf("Error opening file.\n");

return 0;

}

if (AES\_set\_encrypt\_key(key, KEY\_SIZE\*8, &aes\_key) != 0) {

printf("Error setting encryption key.\n");

return 0;

}

while ((bytes\_read = fread(buffer\_in, 1, BLOCK\_SIZE, fp\_in)) > 0) {

AES\_cbc\_encrypt(buffer\_in, buffer\_out, bytes\_read, &aes\_key, iv, AES\_ENCRYPT);

bytes\_written = fwrite(buffer\_out, 1, bytes\_read, fp\_out);

if (bytes\_written != bytes\_read) {

printf("Error writing to file.\n");

return 0;

}

}

fclose(fp\_in);

fclose(fp\_out);

return 1;

}

int decrypt\_file(const char\* input\_file, const char\* output\_file, const unsigned char\* key, const unsigned char\* iv) {

AES\_KEY aes\_key;

unsigned char buffer\_in[BLOCK\_SIZE], buffer\_out[BLOCK\_SIZE];

FILE\* fp\_in = fopen(input\_file, "rb");

FILE\* fp\_out = fopen(output\_file, "wb");

int bytes\_read, bytes\_written;

if (!fp\_in || !fp\_out) {

printf("Error opening file.\n");

return 0;

}

if (AES\_set\_decrypt\_key(key, KEY\_SIZE\*8, &aes\_key) != 0) {

printf("Error setting decryption key.\n");

return 0;

}

while ((bytes\_read = fread(buffer\_in, 1, BLOCK\_SIZE, fp\_in)) > 0) {

AES\_cbc\_encrypt(buffer\_in, buffer\_out, bytes\_read, &aes\_key, iv, AES\_DECRYPT);

bytes\_written = fwrite(buffer\_out, 1, bytes\_read, fp\_out);

if (bytes\_written != bytes\_read) {

printf("Error writing to file.\n");

return 0;

}

}

fclose(fp\_in);

fclose(fp\_out);

return 1;

}

int main() {

const char\* input\_file = "input.txt";

const char\* output\_file = "output.enc";

const char\* decrypted\_file = "output\_decrypted.txt";

unsigned char key[KEY\_SIZE], iv[IV\_SIZE];

if (!generate\_key\_and\_iv(key, iv)) {

printf("Key and IV generation failed.\n");

return 1;

}

if (!encrypt\_file(input\_file, output\_file, key, iv)) {

printf("Encryption failed.\n");

return 1;

}

printf("File encrypted successfully.\n");

if (!decrypt\_file(output\_file, decrypted\_file, key, iv)) {

printf("Decryption failed.\n");

return 1;

}

printf("File decrypted successfully.\n");

return 0;

}

This code uses the `RAND\_bytes()` function from OpenSSL to generate a random key and IV. The `generate\_key\_and\_iv()` function calls this function twice, once to fill the `key` buffer and once to fill the `iv` buffer. If `RAND\_bytes()` returns a non-zero value, the function returns 1 (success), otherwise it returns 0 (failure).

Note that this code still uses the same encryption and decryption functions as the previous code, but with the addition of the `generate\_key\_and\_iv()` function. The `encrypt\_file()` and `decrypt\_file()` functions now take the `key` and `iv` buffers as arguments, which are used to set up the AES key and IV for encryption and decryption.