#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <openssl/aes.h>

#include <dirent.h>

#include <sys/stat.h>

#define BLOCK\_SIZE 16

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

if (argc != 4) {

printf("Usage: %s <key> <iv> <directory>\n", argv[0]);

return 1;

}

unsigned char \*key = (unsigned char \*)argv[1];

unsigned char \*iv = (unsigned char \*)argv[2];

char \*directory = argv[3];

// Set up the AES key and IV

AES\_KEY aes\_key;

AES\_set\_encrypt\_key(key, 256, &aes\_key);

// Iterate through files in the directory

DIR \*dir;

struct dirent \*ent;

if ((dir = opendir(directory)) != NULL) {

while ((ent = readdir(dir)) != NULL) {

if (ent->d\_type == DT\_REG) {

// Construct input and output file paths

char \*in\_path = (char \*)malloc(strlen(directory) + strlen(ent->d\_name) + 2);

sprintf(in\_path, "%s/%s", directory, ent->d\_name);

char \*out\_path = (char \*)malloc(strlen(directory) + strlen(ent->d\_name) + 6);

sprintf(out\_path, "%s/%s.enc", directory, ent->d\_name);

// Open input and output files

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

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

// Determine input file size

struct stat st;

stat(in\_path, &st);

int in\_size = st.st\_size;

// Allocate input and output buffers

unsigned char \*in\_buf = (unsigned char \*)malloc(in\_size);

unsigned char \*out\_buf = (unsigned char \*)malloc(in\_size + BLOCK\_SIZE);

// Read input file into buffer

fread(in\_buf, 1, in\_size, in\_file);

// Perform padding on input buffer

int num\_blocks = (in\_size + BLOCK\_SIZE - 1) / BLOCK\_SIZE;

int padded\_size = num\_blocks \* BLOCK\_SIZE;

memset(in\_buf + in\_size, padded\_size - in\_size, padded\_size - in\_size);

// Encrypt input buffer and write to output file

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

fwrite(out\_buf, 1, padded\_size, out\_file);

// Close input and output files

fclose(in\_file);

fclose(out\_file);

// Free memory

free(in\_buf);

free(out\_buf);

free(in\_path);

free(out\_path);

}

}

closedir(dir);

} else {

perror("Failed to open directory");

return 1;

}

return 0;

}

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <openssl/aes.h>

#include <dirent.h>

#include <sys/stat.h>

#define BLOCK\_SIZE 16

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

if (argc != 5) {

printf("Usage: %s <key> <iv> <directory> <mode>\n", argv[0]);

printf("Mode: 0 for encryption, 1 for decryption\n");

return 1;

}

unsigned char \*key = (unsigned char \*)argv[1];

unsigned char \*iv = (unsigned char \*)argv[2];

char \*directory = argv[3];

int mode = atoi(argv[4]);

// Set up the AES key and IV

AES\_KEY aes\_key;

if (mode == 0) {

AES\_set\_encrypt\_key(key, 256, &aes\_key);

} else {

AES\_set\_decrypt\_key(key, 256, &aes\_key);

}

// Iterate through files in the directory

DIR \*dir;

struct dirent \*ent;

if ((dir = opendir(directory)) != NULL) {

while ((ent = readdir(dir)) != NULL) {

if (ent->d\_type == DT\_REG) {

// Construct input and output file paths

char \*in\_path = (char \*)malloc(strlen(directory) + strlen(ent->d\_name) + 2);

sprintf(in\_path, "%s/%s", directory, ent->d\_name);

char \*out\_path = (char \*)malloc(strlen(directory) + strlen(ent->d\_name));

strncpy(out\_path, ent->d\_name, strlen(ent->d\_name) - 4);

out\_path[strlen(ent->d\_name) - 4] = '\0';

char \*ext = strrchr(ent->d\_name, '.');

if (ext != NULL && strcmp(ext, ".enc") == 0) {

if (mode == 0) {

strcat(out\_path, ".enc");

}

} else {

if (mode == 1) {

continue;

}

}

// Open input and output files

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

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

// Determine input file size

struct stat st;

stat(in\_path, &st);

int in\_size = st.st\_size;

// Allocate input and output buffers

unsigned char \*in\_buf = (unsigned char \*)malloc(in\_size);

unsigned char \*out\_buf = (unsigned char \*)malloc(in\_size + BLOCK\_SIZE);

// Read input file into buffer

fread(in\_buf, 1, in\_size, in\_file);

// Decrypt or encrypt input buffer and write to output file

if (mode == 0) {

// Perform padding on input buffer

int num\_blocks = (in\_size + BLOCK\_SIZE - 1) / BLOCK\_SIZE;

int padded\_size = num\_blocks \* BLOCK\_SIZE;

memset(in\_buf + in\_size, padded\_size - in\_size, padded\_size - in\_size);

// Encrypt input buffer and write to output file

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

fwrite(out\_buf, 1, padded\_size, out\_file);

} else {

// Decrypt input buffer and write to output

// Decrypt input buffer and write to output file

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

// Remove padding from output buffer

int padding\_size = out\_buf[in\_size - 1];

fwrite(out\_buf, 1, in\_size - padding\_size, out\_file);

}

// Close files and free memory

fclose(in\_file);

fclose(out\_file);

free(in\_path);

free(out\_path);

free(in\_buf);

free(out\_buf);

}

}

closedir(dir);

} else {

printf("Unable to open directory\n");

return 1;

}

return 0;

}