**Code for ChaCha20 implementation**

**Team 6**

**Students :**

**Sarthaka Mitra GB (1BM23CS305)**

**Shaikh Uzair Ahmed (1BM23CS307)**

**Shashank U (1BM23CS314)**

**Subramanya J (1BM23CS343)**

/\*\*

\* @file chacha20.h

\* @brief Header file for ChaCha20 stream cipher implementation.

\*

\* @author Subramanya J

\* subramanyajaradhya@gmail.com

\*

\* Implements the functions defined in chacha20.h

\*/

#include <fcntl.h>

#include <unistd.h>

#include <stdlib.h>

#include <string.h>

#include <stdio.h>

#include <stdint.h>

#include "chacha20.h"

uint8\_t \*get\_key() {

char buffer[256];

uint8\_t \*key = calloc(32, sizeof(uint8\_t));

fprintf(stdout, "Enter password: ");

fflush(stdout);

if (!fgets(buffer, sizeof(buffer), stdin)) {

fprintf(stderr, "Error reading input.\n");

free(key);

return NULL;

}

size\_t len = strlen(buffer);

if (len > 0 && buffer[len - 1] == '\n') {

buffer[len - 1] = '\0';

len--;

}

memcpy(key, buffer, len > 32 ? 32 : len);

return key;

}

uint8\_t \*get\_nonce() {

uint8\_t \*nonce = calloc(12, sizeof(uint8\_t));

int fd = open("/dev/urandom", O\_RDONLY);

if (fd < 0) {

random\_failure:

fprintf(stderr, "Warning: /dev/urandom could not be used.\nUsing stdlib's rand() instead.\n");

for (int i = 0; i < 12; i++) {

nonce[i] = rand();

}

return nonce;

}

ssize\_t bytes\_read = read(fd, nonce, 12);

close(fd);

if (bytes\_read != 12) {

goto random\_failure;

}

return nonce;

}

uint32\_t load32\_le(const uint8\_t \*src) {

return ((uint32\_t)src[0]) |

((uint32\_t)src[1] << 8) |

((uint32\_t)src[2] << 16) |

((uint32\_t)src[3] << 24);

}

void store32\_le(uint8\_t \*dst, uint32\_t w) {

dst[0] = w & 0xff;

dst[1] = (w >> 8) & 0xff;

dst[2] = (w >> 16) & 0xff;

dst[3] = (w >> 24) & 0xff;

}

void quarterround(uint32\_t \*a, uint32\_t \*b, uint32\_t \*c, uint32\_t \*d) {

\*a += \*b; \*d ^= \*a; \*d = ROTL32(\*d, 16);

\*c += \*d; \*b ^= \*c; \*b = ROTL32(\*b, 12);

\*a += \*b; \*d ^= \*a; \*d = ROTL32(\*d, 8);

\*c += \*d; \*b ^= \*c; \*b = ROTL32(\*b, 7);

}

void chacha20\_block(uint32\_t state[16], uint8\_t output[64]) {

uint32\_t working\_state[16];

memcpy(working\_state, state, 64);

for (int i = 0; i < 10; i++) {

quarterround(&working\_state[0], &working\_state[4], &working\_state[8], &working\_state[12]);

quarterround(&working\_state[1], &working\_state[5], &working\_state[9], &working\_state[13]);

quarterround(&working\_state[2], &working\_state[6], &working\_state[10], &working\_state[14]);

quarterround(&working\_state[3], &working\_state[7], &working\_state[11], &working\_state[15]);

quarterround(&working\_state[0], &working\_state[5], &working\_state[10], &working\_state[15]);

quarterround(&working\_state[1], &working\_state[6], &working\_state[11], &working\_state[12]);

quarterround(&working\_state[2], &working\_state[7], &working\_state[8], &working\_state[13]);

quarterround(&working\_state[3], &working\_state[4], &working\_state[9], &working\_state[14]);

}

for (int i = 0; i < 16; i++) {

working\_state[i] += state[i];

}

memcpy(output, working\_state, 64);

}

void poly1305\_auth(uint8\_t \*mac, const uint8\_t \*msg, size\_t msg\_len, const uint8\_t key[32]) {

uint32\_t r[4], s[4], acc[5] = {0};

memcpy(r, key, 16);

memcpy(s, key + 16, 16);

r[0] &= 0x3FFFFFFF;

r[1] &= 0xFFFFFFC0;

r[2] &= 0xFFFFFFC0;

r[3] &= 0x0FFFFFFF;

while (msg\_len > 0) {

uint32\_t block[4] = {0};

size\_t block\_len = (msg\_len >= 16) ? 16 : msg\_len;

memcpy(block, msg, block\_len);

block[block\_len / 4] |= (1 << (8 \* (block\_len % 4))); // 1 bit padding

for (int i = 0; i < 4; i++) {

acc[i] += block[i];

}

uint64\_t t = 0;

for (int i = 0; i < 4; i++) {

t += (uint64\_t)acc[i] \* r[i];

acc[i] = (uint32\_t)t;

t >>= 32;

}

msg += block\_len;

msg\_len -= block\_len;

}

for (int i = 0; i < 4; i++) {

acc[i] += s[i];

}

memcpy(mac, acc, 16);

}

/\*\*

\* @file chacha20.h

\* @brief Header file for ChaCha20 stream cipher implementation.

\*

\* @author Subramanya J

\* subramanyajaradhya@gmail.com

\*

\* This file provides the function declarations and macros needed

\* to use the ChaCha20 encryption block function, along with helper

\* routines for key and nonce handling.

\*/

#ifndef CHACHA\_20\_H

#define CHACHA\_20\_H

#include <stdint.h>

#include <stdio.h>

#include <stdint.h>

/\*\*

\* @brief Prompts the user for a password and returns a 32-byte key.

\*

\* The password entered by the user is truncated or zero-padded to 32 bytes.

\*

\* @return Pointer to a heap-allocated 32-byte buffer containing the key.

\* Must be freed by the caller.

\*/

uint8\_t \*

get\_key();

/\*\*

\* @brief Generates a 12-byte nonce using /dev/urandom or fallback to rand().

\*

\* Tries to read from `/dev/urandom`; falls back to `rand()` if unavailable.

\*

\* @return Pointer to a heap-allocated 12-byte buffer containing the nonce.

\* Must be freed by the caller.

\*/

uint8\_t \*

get\_nonce();

/\*\*

\* @brief Loads a 32-bit unsigned integer from a little-endian byte array.

\*

\* @param src Pointer to 4 bytes in memory in little-endian order.

\* @return 32-bit unsigned integer.

\*/

uint32\_t

load32\_le(const uint8\_t \*src);

/\*\*

\* @brief Stores a 32-bit unsigned integer into a byte array in little-endian format.

\*

\* @param dst Destination pointer to a 4-byte memory location.

\* @param w 32-bit unsigned integer to store.

\*/

void

store32\_le(uint8\_t \*dst, uint32\_t w);

/\*\*

\* @brief Rotates a 32-bit unsigned integer `v` left by `n` bits.

\*/

#define ROTL32(v, n) ((v << n) | (v >> (32 - n)))

/\*\*

\* @brief Applies the ChaCha20 quarter round function to four 32-bit words.

\*

\* This function performs 4 rounds of ARX (Add-Rotate-XOR) operations on the inputs.

\* b ^= (a+d) <<< 7;

\* c ^= (b+a) <<< 9;

\* d ^= (c+b) <<< 13;

\* a ^= (d+c) <<< 18;

\*/

void quarterround(uint32\_t \*a, uint32\_t \*b, uint32\_t \*c, uint32\_t \*d);

/\*\*

\* @brief Executes the ChaCha20 block function on a given 512-bit state.

\*

\* The output is 64 bytes (512 bits) of keystream, generated from 10 double-rounds.

\*

\* @param state Pointer to the 16-word (64-byte) input state.

\* @param output Pointer to a 64-byte buffer where output will be stored.

\*/

void chacha20\_block(uint32\_t state[16], uint8\_t output[64]);

/\*\*

\* @brief Generates a Poly1305 authentication tag for the given message.

\*

\* This tag is used for message authentication in the ChaCha20-Poly1305 scheme.

\*

\* @param mac Pointer to a 16-byte buffer for the generated tag.

\* @param msg Pointer to the message to be authenticated.

\* @param msg\_len Length of the message in bytes.

\* @param key 32-byte key for Poly1305.

\*/

void poly1305\_auth(uint8\_t \*mac, const uint8\_t \*msg, size\_t msg\_len, const uint8\_t key[32]);

#endif /\* chacha20.h included \*/

/\*\*

\* @file main.c

\* @brief Entry point for the ChaCha20 file encryption/decryption utility.

\*

\* This program encrypts or decrypts a file using the ChaCha20 stream cipher.

\* It reads from an input file, processes 64-byte blocks, and writes to an

\* output file. It uses a password-derived key and a 12-byte nonce.

\*

\* Usage: ./program <filename> <e|d>

\* - 'e' to encrypt the file

\* - 'd' to decrypt a previously encrypted file

\*

\* @author Subramanya J

\* subramanyajaradhya@gmail.com

\*/

#include <stdint.h>

#include <string.h>

#include <unistd.h>

#include <stdlib.h>

#include <stdbool.h>

#include <stdio.h>

#include <fcntl.h>

#include "chacha20.h"

/\*\*

\* @brief Main function to handle encryption/decryption based on command-line arguments.

\*

\* The program uses a password-derived key for encryption or decryption of a file.

\* For encryption, it generates a random 12-byte nonce and writes it to the output file.

\* For decryption, it reads the nonce from the input file.

\*

\* @param argc Argument count.

\* @param argv Argument vector.

\* argv[1]: input filename

\* argv[2]: 'e' for encryption, 'd' for decryption

\*

\* @return int Exit status code (0 for success, 1 for argument error, or other error codes).

\*/

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

if (argc != 3 || (strcmp(argv[2], "e") != 0 && strcmp(argv[2], "d") != 0)) {

fprintf(stderr, "Usage: %s <filename> <e|d>\n", argv[0]);

return 1;

}

bool encrypting = (strcmp(argv[2], "e") == 0);

int in\_fd = open(argv[1], O\_RDONLY);

if (in\_fd < 0) {

fprintf(stderr, "Could not open the file %s\n", argv[1]);

exit(EXIT\_FAILURE);

}

char \*out\_filename = calloc(256, sizeof(char));

snprintf(out\_filename, 256, "%s.%s", argv[1], encrypting ? "enc" : "dec");

int out\_fd = open(out\_filename, O\_WRONLY | O\_CREAT | O\_TRUNC, 0644);

free(out\_filename);

uint8\_t \*key = get\_key();

uint8\_t nonce[12];

uint8\_t mac[16];

if (encrypting) {

uint8\_t \*nonce\_ptr = get\_nonce();

memcpy(nonce, nonce\_ptr, 12);

write(out\_fd, nonce, 12);

free(nonce\_ptr);

} else {

if (read(in\_fd, nonce, 12) != 12) {

fprintf(stderr, "Failed to read nonce from input file.\n");

exit(EXIT\_FAILURE);

}

}

uint8\_t \*in\_buf = calloc(64, sizeof(uint8\_t));

uint8\_t \*ciphertext = calloc(64, sizeof(uint8\_t));

uint8\_t keystream[64];

ssize\_t in\_read;

uint32\_t counter = 0;

while ((in\_read = read(in\_fd, in\_buf, 64)) > 0) {

uint32\_t state[16] = {

0x61707865, 0x3320646E, 0x79622D32, 0x6B206574,

};

for (int i = 0; i < 8; i++) {

state[4 + i] = load32\_le(key + i \* 4);

}

state[12] = counter++;

state[13] = load32\_le(nonce);

state[14] = load32\_le(nonce + 4);

state[15] = load32\_le(nonce + 8);

chacha20\_block(state, keystream);

for (int i = 0; i < in\_read; i++) {

ciphertext[i] = in\_buf[i] ^ keystream[i];

}

write(out\_fd, ciphertext, in\_read);

}

if (encrypting) {

lseek(out\_fd, 0, SEEK\_END);

poly1305\_auth(mac, ciphertext, in\_read, key);

write(out\_fd, mac, 16);

} else {

lseek(in\_fd, -16, SEEK\_END);

read(in\_fd, mac, 16);

uint8\_t computed\_mac[16];

poly1305\_auth(computed\_mac, ciphertext, in\_read, key);

if (memcmp(mac, computed\_mac, 16) != 0) {

fprintf(stderr, "Authentication failed. File may be corrupted.\n");

exit(EXIT\_FAILURE);

}

}

free(in\_buf);

free(ciphertext);

free(key);

close(in\_fd);

close(out\_fd);

return 0;

}

# Makefile

CC= clang

CFLAGS= -O3 -g -Wall -Werror -Wextra -fsanitize=address

all:

$(CC) main.c chacha20.c -o chacha20 $(CFLAGS)

clean:

rm chacha20