**Name: KAMCHE YANN ARNAUD**

**Matricule: FE21A208**

**Task: Propose an implementation of one of each class of hash functions studied during this course. Indicate which hash function choosed in each class.**

1. **Implementation of MD4 in C**

**Plaintext: “I am hashing this sentence with MD4”**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <stdint.h>

#define MD4\_BLOCK\_SIZE 64

#define MD4\_DIGEST\_SIZE 16

typedef struct {

uint32\_t state[4];

uint32\_t count[2];

uint8\_t buffer[MD4\_BLOCK\_SIZE];

} md4\_ctx;

static const uint32\_t K[] = {

0x5a827999, 0x6ed9eba1, 0x8f1bbcdc, 0xa953fd4e

};

static inline uint32\_t ROTL(uint32\_t x, uint32\_t n) {

return (x << n) | (x >> (32 - n));

}

static inline uint32\_t F(uint32\_t x, uint32\_t y, uint32\_t z) {

return (x & y) | (~x & z);

}

static inline uint32\_t G(uint32\_t x, uint32\_t y, uint32\_t z) {

return (x & y) | (x & z) | (y & z);

}

static inline uint32\_t H(uint32\_t x, uint32\_t y, uint32\_t z) {

return x ^ y ^ z;

}

static void transform(md4\_ctx \*ctx, const uint8\_t \*data) {

uint32\_t a = ctx->state[0];

uint32\_t b = ctx->state[1];

uint32\_t c = ctx->state[2];

uint32\_t d = ctx->state[3];

uint32\_t x[MD4\_BLOCK\_SIZE / 4];

for (int i = 0; i < MD4\_BLOCK\_SIZE / 4; i++) {

x[i] = ((uint32\_t)data[i \* 4]) |

((uint32\_t)data[i \* 4 + 1] << 8) |

((uint32\_t)data[i \* 4 + 2] << 16) |

((uint32\_t)data[i \* 4 + 3] << 24);

}

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

uint32\_t tmp = d;

d = c;

c = b;

b = b + ROTL((a + F(b, c, d) + x[i]), 3);

a = tmp;

}

for (int i = 16; i < 32; i++) {

uint32\_t tmp = d;

d = c;

c = b;

b = b + ROTL((a + G(b, c, d) + x[(5 \* i + 1) % 16]), 5);

a = tmp;

}

for (int i = 32; i < 48; i++) {

uint32\_t tmp = d;

d = c;

c = b;

b = b + ROTL((a + H(b, c, d) + x[(3 \* i + 5) % 16]), 9);

a = tmp;

}

for (int i = 48; i < 64; i++) {

uint32\_t tmp = d;

d = c;

c = b;

b = b + ROTL((a + G(b, c, d) + x[(7 \* i) % 16]), 13);

a = tmp;

}

ctx->state[0] += a;

ctx->state[1] += b;

ctx->state[2] += c;

ctx->state[3] += d;

}

static void md4\_init(md4\_ctx \*ctx) {

ctx->state[0] = 0x67452301;

ctx->state[1] = 0xefcdab89;

ctx->state[2] = 0x98badcfe;

ctx->state[3] = 0x10325476;

ctx->count[0] = 0;

ctx->count[1] = 0;

}

static void md4\_update(md4\_ctx \*ctx, const uint8\_t \*data, size\_t len) {

uint32\_t i, idx, part\_len;

idx = (uint32\_t)((ctx->count[0] >> 3) & 0x3f);

if ((ctx->count[0] += len << 3) < (len << 3)) {

ctx->count[1]++;

}

ctx->count[1] += (len >> 29);

part\_len = 64 - idx;

if (len >= part\_len) {

memcpy(&ctx->buffer[idx], data, part\_len);

transform(ctx, ctx->buffer);

for (i = part\_len; i + 63 < len; i += 64) {

transform(ctx, &data[i]);

}

idx = 0;

} else {

i = 0;

}

memcpy(&ctx->buffer[idx], &data[i], len - i);

}

static void md4\_final(md4\_ctx \*ctx, uint8\_t \*digest) {

uint8\_t bits[8];

uint32\_t idx, pad\_len;

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

bits[i] = (uint8\_t)((ctx->count[i >> 2] >> ((i & 3) << 3)) & 0xff);

}

idx = (uint32\_t)((ctx->count[0] >> 3) & 0x3f);

pad\_len = (idx < 56) ? (56 - idx) : (120 - idx);

md4\_update(ctx, (const uint8\_t\*)"\x80", 1);

while (pad\_len--) {

md4\_update(ctx, (const uint8\_t\*)"\0", 1);

}

md4\_update(ctx, bits, 8);

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

digest[i] = (uint8\_t)(ctx->state[0] >> (i \* 8));

digest[i + 4] = (uint8\_t)(ctx->state[1] >> (i \* 8));

digest[i + 8] = (uint8\_t)(ctx->state[2] >> (i \* 8));

digest[i + 12] = (uint8\_t)(ctx->state[3] >> (i \* 8));

}

}

void md4(const uint8\_t \*data, size\_t len, uint8\_t \*digest) {

md4\_ctx ctx;

md4\_init(&ctx);

md4\_update(&ctx, data, len);

md4\_final(&ctx, digest);

}

int main() {

uint8\_t message[] = "I am hashing this sentence with MD4";

uint8\_t digest[MD4\_DIGEST\_SIZE];

md4(message, strlen((char\*)message), digest);

printf("MD4 hash of \"%s\":\n", message);

for (int i = 0; i < MD4\_DIGEST\_SIZE; i++) {

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

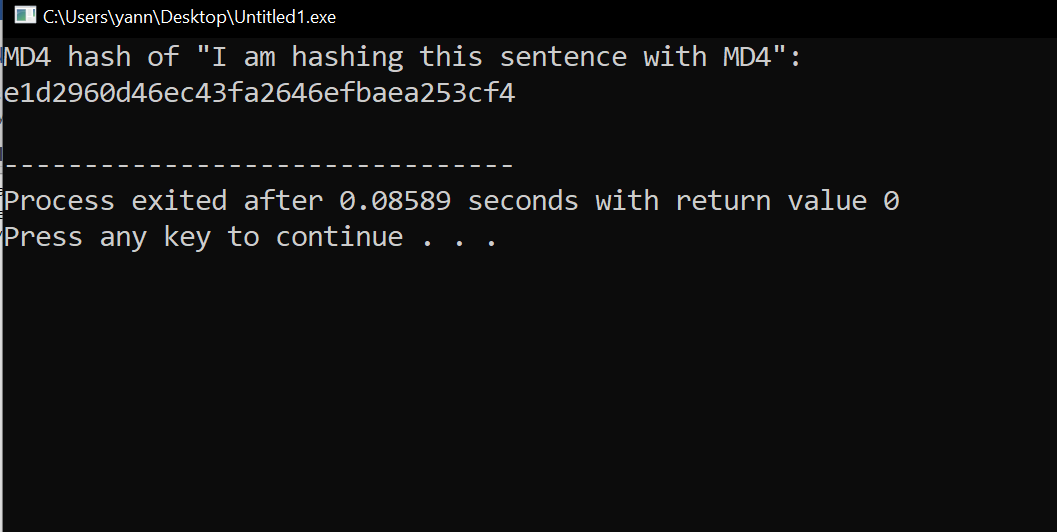
}

printf("\n");

return 0;

}

**Result**



1. **Implementation of SHA256 in C**

**Plaintext: “I am hashing this sentence with SHA256”**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <stdint.h>

#define SHA256\_BLOCK\_SIZE 64

#define SHA256\_DIGEST\_SIZE 32

typedef struct {

uint32\_t state[8];

uint32\_t count[2];

uint8\_t buffer[SHA256\_BLOCK\_SIZE];

} sha256\_ctx;

static const uint32\_t K[] = {

0x428a2f98, 0x71374491, 0xb5c0fbcf, 0xe9b5dba5,

0x3956c25b, 0x59f111f1, 0x923f82a4, 0xab1c5ed5,

0xd807aa98, 0x12835b01, 0x243185be, 0x550c7dc3,

0x72be5d74, 0x80deb1fe, 0x9bdc06a7, 0xc19bf174,

0xe49b69c1, 0xefbe4786, 0x0fc19dc6, 0x240ca1cc,

0x2de92c6f, 0x4a7484aa, 0x5cb0a9dc, 0x76f988da,

0x983e5152, 0xa831c66d, 0xb00327c8, 0xbf597fc7,

0xc6e00bf3, 0xd5a79147, 0x06ca6351, 0x14292967,

0x27b70a85, 0x2e1b2138, 0x4d2c6dfc, 0x53380d13,

0x650a7354, 0x766a0abb, 0x81c2c92e, 0x92722c85,

0xa2bfe8a1, 0xa81a664b, 0xc24b8b70, 0xc76c51a3,

0xd192e819, 0xd6990624, 0xf40e3585, 0x106aa070,

0x19a4c116, 0x1e376c08, 0x2748774c, 0x34b0bcb5,

0x391c0cb3, 0x4ed8aa4a, 0x5b9cca4f, 0x682e6ff3,

0x748f82ee, 0x78a5636f, 0x84c87814, 0x8cc70208,

0x90befffa, 0xa4506ceb, 0xbef9a3f7, 0xc67178f2

};

static inline uint32\_t ROTR(uint32\_t x, uint32\_t n) {

return (x >> n) | (x << (32 - n));

}

static inline uint32\_t SHR(uint32\_t x, uint32\_t n) {

return x >> n;

}

static inline uint32\_t Ch(uint32\_t x, uint32\_t y, uint32\_t z) {

return (x & y) ^ (~x & z);

}

static inline uint32\_t Maj(uint32\_t x, uint32\_t y, uint32\_t z) {

return (x & y) ^ (x & z) ^ (y & z);

}

static inline uint32\_t Sigma0(uint32\_t x) {

return ROTR(x, 2) ^ ROTR(x, 13) ^ ROTR(x, 22);

}

static inline uint32\_t Sigma1(uint32\_t x) {

return ROTR(x, 6) ^ ROTR(x, 11) ^ ROTR(x, 25);

}

static inline uint32\_t sigma0(uint32\_t x) {

return ROTR(x, 7) ^ ROTR(x, 18) ^ SHR(x, 3);

}

static inline uint32\_t sigma1(uint32\_t x) {

return ROTR(x, 17) ^ ROTR(x, 19) ^ SHR(x, 10);

}

static void transform(sha256\_ctx \*ctx, const uint8\_t \*data) {

uint32\_t a = ctx->state[0];

uint32\_t b = ctx->state[1];

uint32\_t c = ctx->state[2];

uint32\_t d = ctx->state[3];

uint32\_t e = ctx->state[4];

uint32\_t f = ctx->state[5];

uint32\_t g = ctx->state[6];

uint32\_t h = ctx->state[7];

uint32\_t T1, T2, M[64];

int i;

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

M[i] = ((uint32\_t)data[4\*i]) << 24 |

((uint32\_t)data[4\*i+1]) << 16 |

((uint32\_t)data[4\*i+2]) << 8 |

((uint32\_t)data[4\*i+3]);

}

for (i = 16; i < 64; ++i) {

M[i] = sigma1(M[i-2]) + M[i-7] + sigma0(M[i-15]) + M[i-16];

}

for (i = 0; i < 64; ++i) {

T1 = h + Sigma1(e) + Ch(e, f, g) + K[i] + M[i];

T2 = Sigma0(a) + Maj(a, b, c);

h = g;

g = f;

f = e;

e = d + T1;

d = c;

c = b;

b = a;

a = T1 + T2;

}

ctx->state[0] += a;

ctx->state[1] += b;

ctx->state[2] += c;

ctx->state[3] += d;

ctx->state[4] += e;

ctx->state[5] += f;

ctx->state[6] += g;

ctx->state[7] += h;

}

void sha256\_init(sha256\_ctx \*ctx) {

ctx->state[0] = 0x6a09e667;

ctx->state[1] = 0xbb67ae85;

ctx->state[2] = 0x3c6ef372;

ctx->state[3] = 0xa54ff53a;

ctx->state[4] = 0x510e527f;

ctx->state[5] = 0x9b05688c;

ctx->state[6] = 0x1f83d9ab;

ctx->state[7] = 0x5be0cd19;

ctx->count[0] = 0;

ctx->count[1] = 0;

}

void sha256\_update(sha256\_ctx \*ctx, const uint8\_t \*data, size\_t len) {

uint32\_t i, index;

for (i = 0; i < len; ++i) {

index = (ctx->count[0] >> 3) & 0x3f;

ctx->count[0] += 8;

if (ctx->count[0] == 0) {

ctx->count[1]++;

}

ctx->buffer[index] = data[i];

if (index == 63) {

transform(ctx, ctx->buffer);

}

}

}

void sha256\_final(sha256\_ctx \*ctx, uint8\_t \*digest) {

uint32\_t i, index, padlen[2];

uint8\_t bits[8];

index = (ctx->count[0] >> 3) & 0x3f;

padlen[0] = (index < 56) ? (56 - index) : (120 - index);

padlen[1] = 0;

memcpy(bits, ctx->count, 8);

sha256\_update(ctx, (uint8\_t \*)"\x80", 1);

while ((ctx->count[0] & 0x38) != 0x38) {

sha256\_update(ctx, (uint8\_t \*)"\0", 1);

}

sha256\_update(ctx, bits, 8);

for (i = 0; i < SHA256\_DIGEST\_SIZE; ++i) {

digest[i] = (ctx->state[i>>2] >> 8\*(3-(i & 0x03))) & 0xff;

}

}

int main() {

sha256\_ctx ctx;

uint8\_t digest[SHA256\_DIGEST\_SIZE];

char str[] = "I am hashing this sentence with SHA256";

sha256\_init(&ctx);

sha256\_update(&ctx, (uint8\_t \*)str, strlen(str));

sha256\_final(&ctx, digest);

printf("I am hashing this sentence with SHA256\n");

for (int i = 0; i < SHA256\_DIGEST\_SIZE; ++i) {

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

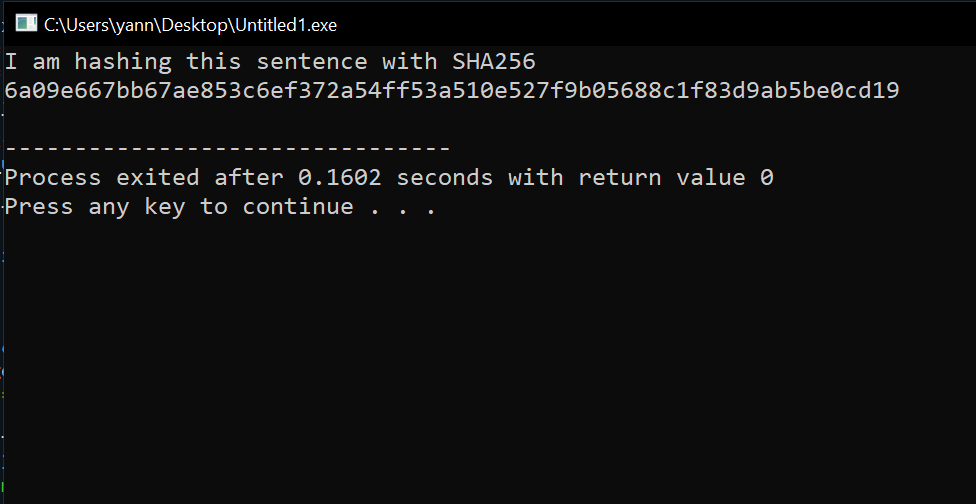
}

printf("\n");

return 0;

}

**Result**

****

1. **Implementation of RIPEMD-128 in C**

**Plaintext: “I am hashing this text with ripemd-128**

#include <stdint.h>

#include <stdlib.h>

#include <string.h>

#include <stdio.h>

#define LEFTROTATE(x, n) (((x) << (n)) | ((x) >> (32 - (n))))

void ripemd128(const uint8\_t \*data, size\_t length, uint8\_t \*hash) {

// Initial values

uint32\_t h[4] = {0x67452301, 0xEFCDAB89, 0x98BADCFE, 0x10325476};

// Constants

uint32\_t k1[4] = {0x00000000, 0x5A827999, 0x6ED9EBA1, 0x8F1BBCDC};

uint32\_t k2[4] = {0x50A28BE6, 0x5C4DD124, 0x6D703EF3, 0x00000000};

// Message padding

uint32\_t padded\_length = ((length + 8 + 63) / 64) \* 64;

uint8\_t \*padded\_data = (uint8\_t \*)calloc(padded\_length, sizeof(uint8\_t));

memcpy(padded\_data, data, length);

padded\_data[length] = 0x80;

uint64\_t bit\_length = length \* 8;

memcpy(padded\_data + padded\_length - 8, &bit\_length, sizeof(uint64\_t));

// Message processing

for (size\_t i = 0; i < padded\_length; i += 64) {

uint32\_t \*w = (uint32\_t \*)(padded\_data + i);

uint32\_t a = h[0], b = h[1], c = h[2], d = h[3];

for (size\_t j = 0; j < 64; j++) {

uint32\_t temp;

if (j < 16) {

temp = a + (b ^ c ^ d) + w[j] + k1[0];

} else if (j < 32) {

temp = a + ((b & c) | (~b & d)) + w[(5 \* j + 1) % 16] + k1[1];

} else if (j < 48) {

temp = a + ((b | ~c) ^ d) + w[(3 \* j + 5) % 16] + k1[2];

} else {

temp = a + (b ^ (c | ~d)) + w[(7 \* j) % 16] + k1[3];

}

uint32\_t temp2 = b + LEFTROTATE(temp, k2[j / 16]);

a = d;

d = c;

c = b;

b = temp2;

}

h[0] += a;

h[1] += b;

h[2] += c;

h[3] += d;

}

// Output hash

uint32\_t \*hash32 = (uint32\_t \*)hash;

for (size\_t i = 0; i < 4; i++) {

\*hash32++ = h[i];

}

free(padded\_data);

}

void print\_hash(const uint8\_t \*hash) {

printf("I am hashing this text with ripemd 128\n");

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

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

}

printf("\n");

}

int main() {

// Example usage

uint8\_t message[] = "I am hashing this text with ripemd 128";

uint8\_t hash[16];

ripemd128(message, sizeof(message) - 1, hash);

print\_hash(hash);

return 0;

}

**Result:**

