Tristan Erney

October 26th, 2021

Intro to Cryptology

Hands On Exercise 8 – AES

1)

a) Original State:

|  |  |  |  |
| --- | --- | --- | --- |
| 00 | 01 | 02 | 03 |
| 04 | 05 | 06 | 07 |
| 08 | 09 | 0A | 0B |
| 0C | 0D | 0E | 0F |

b) State after initial AddRoundKey:

|  |  |  |  |
| --- | --- | --- | --- |
| 01 | 00 | 03 | 02 |
| 05 | 04 | 07 | 06 |
| 09 | 08 | 0B | 0A |
| 0D | 0C | 0F | 0E |

c) State after SubByte:

|  |  |  |  |
| --- | --- | --- | --- |
| 7C | 63 | 7B | 77 |
| 6B | F2 | C5 | 6F |
| 1 | 30 | 2B | 67 |
| D7 | FE | 76 | AB |

d) State after ShiftRows:

|  |  |  |  |
| --- | --- | --- | --- |
| 7C | 63 | 7B | 77 |
| F2 | C5 | 6F | 6B |
| 2B | 67 | 01 | 30 |
| AB | D7 | FE | 76 |

e) State after MixColumns:

|  |  |  |  |
| --- | --- | --- | --- |
| 75 | 22 | B8 | 15 |
| 55 | 8C | 58 | 87 |
| 3E | 0A | 0F | E6 |
| 10 | B2 | 04 | 2E |

f) K[4], K[5], K[6], K[7]:

|  |  |  |  |
| --- | --- | --- | --- |
| 7c | 7d | 7c | 7d |
| 7d | 7c | 7d | 7c |
| 7d | 7c | 7d | 7c |
| 7d | 7c | 7d | 7c |

g) State after AddRoundKey:

|  |  |  |  |
| --- | --- | --- | --- |
| 9 | 5f | c4 | 68 |
| 28 | f0 | 25 | fb |
| 43 | 76 | 72 | 9a |
| 6d | ce | 79 | 52 |

AES.cpp

#include <iostream>

#include <iomanip>

#include <stdlib.h>

#include <stdint.h>

#include <string.h>

uint8\_t gf\_add(uint8\_t a, uint8\_t b) {

return a ^ b;

}

uint8\_t gf\_mul(uint8\_t a, uint8\_t b) {

uint8\_t res = 0;

for (; b; b >>= 1) {

if (0x01 & b) res ^= a;

if (0x80 & a) a = (a << 1) ^ 0b11011;

else a <<= 1;

}

return res;

}

uint64\_t sbox[] = {

0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67, 0x2b, 0xfe, 0xd7, 0xab, 0x76,

0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2, 0xaf, 0x9c, 0xa4, 0x72, 0xc0,

0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5, 0xf1, 0x71, 0xd8, 0x31, 0x15,

0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12, 0x80, 0xe2, 0xeb, 0x27, 0xb2, 0x75,

0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b, 0xd6, 0xb3, 0x29, 0xe3, 0x2f, 0x84,

0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb, 0xbe, 0x39, 0x4a, 0x4c, 0x58, 0xcf,

0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9, 0x02, 0x7f, 0x50, 0x3c, 0x9f, 0xa8,

0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6, 0xda, 0x21, 0x10, 0xff, 0xf3, 0xd2,

0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7, 0x7e, 0x3d, 0x64, 0x5d, 0x19, 0x73,

0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8, 0x14, 0xde, 0x5e, 0x0b, 0xdb,

0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3, 0xac, 0x62, 0x91, 0x95, 0xe4, 0x79,

0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4, 0xea, 0x65, 0x7a, 0xae, 0x08,

0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd, 0x74, 0x1f, 0x4b, 0xbd, 0x8b, 0x8a,

0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35, 0x57, 0xb9, 0x86, 0xc1, 0x1d, 0x9e,

0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e, 0x87, 0xe9, 0xce, 0x55, 0x28, 0xdf,

0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d, 0x0f, 0xb0, 0x54, 0xbb, 0x16

};

uint64\_t sboxInv[] = {

0x52, 0x09, 0x6a, 0xd5, 0x30, 0x36, 0xa5, 0x38, 0xbf, 0x40, 0xa3, 0x9e, 0x81, 0xf3, 0xd7, 0xfb,

0x7c, 0xe3, 0x39, 0x82, 0x9b, 0x2f, 0xff, 0x87, 0x34, 0x8e, 0x43, 0x44, 0xc4, 0xde, 0xe9, 0xcb,

0x54, 0x7b, 0x94, 0x32, 0xa6, 0xc2, 0x23, 0x3d, 0xee, 0x4c, 0x95, 0x0b, 0x42, 0xfa, 0xc3, 0x4e,

0x08, 0x2e, 0xa1, 0x66, 0x28, 0xd9, 0x24, 0xb2, 0x76, 0x5b, 0xa2, 0x49, 0x6d, 0x8b, 0xd1, 0x25,

0x72, 0xf8, 0xf6, 0x64, 0x86, 0x68, 0x98, 0x16, 0xd4, 0xa4, 0x5c, 0xcc, 0x5d, 0x65, 0xb6, 0x92,

0x6c, 0x70, 0x48, 0x50, 0xfd, 0xed, 0xb9, 0xda, 0x5e, 0x15, 0x46, 0x57, 0xa7, 0x8d, 0x9d, 0x84,

0x90, 0xd8, 0xab, 0x00, 0x8c, 0xbc, 0xd3, 0x0a, 0xf7, 0xe4, 0x58, 0x05, 0xb8, 0xb3, 0x45, 0x06,

0xd0, 0x2c, 0x1e, 0x8f, 0xca, 0x3f, 0x0f, 0x02, 0xc1, 0xaf, 0xbd, 0x03, 0x01, 0x13, 0x8a, 0x6b,

0x3a, 0x91, 0x11, 0x41, 0x4f, 0x67, 0xdc, 0xea, 0x97, 0xf2, 0xcf, 0xce, 0xf0, 0xb4, 0xe6, 0x73,

0x96, 0xac, 0x74, 0x22, 0xe7, 0xad, 0x35, 0x85, 0xe2, 0xf9, 0x37, 0xe8, 0x1c, 0x75, 0xdf, 0x6e,

0x47, 0xf1, 0x1a, 0x71, 0x1d, 0x29, 0xc5, 0x89, 0x6f, 0xb7, 0x62, 0x0e, 0xaa, 0x18, 0xbe, 0x1b,

0xfc, 0x56, 0x3e, 0x4b, 0xc6, 0xd2, 0x79, 0x20, 0x9a, 0xdb, 0xc0, 0xfe, 0x78, 0xcd, 0x5a, 0xf4,

0x1f, 0xdd, 0xa8, 0x33, 0x88, 0x07, 0xc7, 0x31, 0xb1, 0x12, 0x10, 0x59, 0x27, 0x80, 0xec, 0x5f,

0x60, 0x51, 0x7f, 0xa9, 0x19, 0xb5, 0x4a, 0x0d, 0x2d, 0xe5, 0x7a, 0x9f, 0x93, 0xc9, 0x9c, 0xef,

0xa0, 0xe0, 0x3b, 0x4d, 0xae, 0x2a, 0xf5, 0xb0, 0xc8, 0xeb, 0xbb, 0x3c, 0x83, 0x53, 0x99, 0x61,

0x17, 0x2b, 0x04, 0x7e, 0xba, 0x77, 0xd6, 0x26, 0xe1, 0x69, 0x14, 0x63, 0x55, 0x21, 0x0c, 0x7d

};

uint8\_t rcon[] = {

0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a,

0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39,

0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a,

0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8,

0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef,

0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc,

0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b,

0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3,

0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94,

0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20,

0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35,

0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f,

0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04,

0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63,

0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd,

0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb

};

struct Mat44 {

uint8\_t data[4][4];

};

void mat44\_print(Mat44\* m) {

for (int i = 0; i < 4; i += 1) {

std::cout << std::hex << "| ";

for (int j = 0; j < 4; j += 1) {

std::cout << std::setw(3) << (uint32\_t)m->data[i][j] << " ";

}

std::cout << "|\n" << std::dec;

}

std::cout << "\n";

}

Mat44\* mat44\_set(const uint8\_t (&a)[4], const uint8\_t (&b)[4], const uint8\_t (&c)[4], const uint8\_t (&d)[4]) {

Mat44\* m = (Mat44\*)malloc(sizeof(Mat44));

memcpy(m->data[0], a, sizeof(uint8\_t) \* 4);

memcpy(m->data[1], b, sizeof(uint8\_t) \* 4);

memcpy(m->data[2], c, sizeof(uint8\_t) \* 4);

memcpy(m->data[3], d, sizeof(uint8\_t) \* 4);

return m;

}

// SubByte Layer

Mat44\* sub\_byte(Mat44\* m) {

Mat44\* bs\_mat = NULL;

uint8\_t data[4][4] = {{0}, {0}, {0}, {0}};

for (uint32\_t i = 0; i < 4; i += 1) {

for (uint32\_t j = 0; j < 4; j += 1) {

uint8\_t a = 0x0f & m->data[i][j] >> 4,

b = 0x0f & m->data[i][j];

data[i][j] = sbox[a \* 16 + b]; // row a, col b

}

}

bs\_mat = mat44\_set(data[0], data[1], data[2], data[3]);

return bs\_mat;

}

enum : uint8\_t {

NORM,

INV

};

Mat44\* shift\_rows(Mat44\* m, uint8\_t mode = NORM) {

Mat44\* shift = NULL;

uint8\_t data[4][4] = {{0}, {0}, {0}, {0}};

switch (mode) {

case NORM:

{

memcpy(data[0], m->data[0], sizeof(uint32\_t) \* 4);

data[1][0] = m->data[1][1]; data[1][1] = m->data[1][2]; data[1][2] = m->data[1][3]; data[1][3] = m->data[1][0];

data[2][0] = m->data[2][2]; data[2][1] = m->data[2][3]; data[2][2] = m->data[2][0]; data[2][3] = m->data[2][1];

data[3][0] = m->data[3][3]; data[3][1] = m->data[3][0]; data[3][2] = m->data[3][1]; data[3][3] = m->data[3][2];

}

break;

case INV:

{

memcpy(data[0], m->data[0], sizeof(uint32\_t) \* 4);

data[1][0] = m->data[1][3]; data[1][1] = m->data[1][0]; data[1][2] = m->data[1][1]; data[1][3] = m->data[1][2];

data[2][0] = m->data[2][2]; data[2][1] = m->data[2][3]; data[2][2] = m->data[2][0]; data[2][3] = m->data[2][1];

data[3][0] = m->data[3][1]; data[3][1] = m->data[3][2]; data[3][2] = m->data[3][3]; data[3][3] = m->data[3][0];

}

break;

default: // code doesn't exist

return NULL;

}

shift = mat44\_set(data[0], data[1], data[2], data[3]);

return shift;

}

uint8\_t\* mix\_col(uint8\_t (&m)[4], uint8\_t mode = NORM) {

uint8\_t\* col = (uint8\_t\*)malloc(sizeof(uint8\_t) \* 4);

switch (mode) {

case NORM:

{

col[0] = gf\_mul(m[0], 2) ^ gf\_mul(m[1], 3) ^ gf\_mul(m[2], 1) ^ gf\_mul(m[3], 1);

col[1] = gf\_mul(m[0], 1) ^ gf\_mul(m[1], 2) ^ gf\_mul(m[2], 3) ^ gf\_mul(m[3], 1);

col[2] = gf\_mul(m[0], 1) ^ gf\_mul(m[1], 1) ^ gf\_mul(m[2], 2) ^ gf\_mul(m[3], 3);

col[3] = gf\_mul(m[0], 3) ^ gf\_mul(m[1], 1) ^ gf\_mul(m[2], 1) ^ gf\_mul(m[3], 2);

}

break;

case INV:

{

col[0] = gf\_mul(m[0], 14) ^ gf\_mul(m[1], 11) ^ gf\_mul(m[2], 13) ^ gf\_mul(m[3], 9);

col[1] = gf\_mul(m[0], 9) ^ gf\_mul(m[1], 14) ^ gf\_mul(m[2], 11) ^ gf\_mul(m[3], 13);

col[2] = gf\_mul(m[0], 13) ^ gf\_mul(m[1], 9) ^ gf\_mul(m[2], 14) ^ gf\_mul(m[3], 11);

col[3] = gf\_mul(m[0], 11) ^ gf\_mul(m[1], 13) ^ gf\_mul(m[2], 9) ^ gf\_mul(m[3], 14);

break;

}

}

return col;

}

Mat44\* mix\_columns(Mat44\* m, uint8\_t mode = NORM) {

Mat44\* m\_mat = NULL;

uint8\_t data[4][4] = {{0}, {0}, {0}, {0}};

for (int i = 0; i < 4; i += 1) {

uint8\_t\* col = NULL;

uint8\_t dat[4] = {0, 0, 0, 0};

for (int j = 0; j < 4; j += 1) {

dat[j] = m->data[j][i];

}

col = mix\_col(dat, mode);

data[0][i] = col[0];

data[1][i] = col[1];

data[2][i] = col[2];

data[3][i] = col[3];

free(col);

}

m\_mat = mat44\_set(data[0], data[1], data[2], data[3]);

return m\_mat;

}

Mat44\* add\_round\_key(Mat44\* m, Mat44\* k) {

Mat44\* n = mat44\_set(

{gf\_add(m->data[0][0], k->data[0][0]), gf\_add(m->data[0][1], k->data[0][1]), gf\_add(m->data[0][2], k->data[0][2]), gf\_add(m->data[0][3], k->data[0][3])},

{gf\_add(m->data[1][0], k->data[1][0]), gf\_add(m->data[1][1], k->data[1][1]), gf\_add(m->data[1][2], k->data[1][2]), gf\_add(m->data[1][3], k->data[1][3])},

{gf\_add(m->data[2][0], k->data[2][0]), gf\_add(m->data[2][1], k->data[2][1]), gf\_add(m->data[2][2], k->data[2][2]), gf\_add(m->data[2][3], k->data[2][3])},

{gf\_add(m->data[3][0], k->data[3][0]), gf\_add(m->data[3][1], k->data[3][1]), gf\_add(m->data[3][2], k->data[3][2]), gf\_add(m->data[3][3], k->data[3][3])}

);

return n;

}

Mat44\* get\_round\_key(Mat44\* key) {

Mat44\* new\_k = NULL;

uint8\_t new\_data[4][4] = {{0}, {0}, {0}, {0}};

for (int k = 0; k < 4; k += 1) {

uint8\_t data1[4] = {0, 0, 0, 0};

uint8\_t data2[4] = {0, 0, 0, 0};

if ((k + 4) % 4 == 0) {

for (int i = 0; i < 4; i += 1) {

data1[i] = key->data[i][0];

}

for (int i = 0; i < 4; i += 1) {

data2[i] = key->data[i][3];

}

// rotate

uint8\_t tmp = data2[0];

data2[0] = data2[1];

data2[1] = data2[2];

data2[2] = data2[3];

data2[3] = tmp;

// s-box

for (int i = 0; i < 4; i += 1) {

data2[i] = sbox[(data2[i] >> 4) \* 16 + (0x0f & data2[i])];

}

uint8\_t x = 0x01;

data2[0] = data2[0] ^ x;

new\_data[0][0] = data1[0] ^ data2[0];

new\_data[1][0] = data1[1] ^ data2[1];

new\_data[2][0] = data1[2] ^ data2[2];

new\_data[3][0] = data1[3] ^ data2[3];

} else {

for (int i = 0; i < 4; i += 1) {

data1[i] = key->data[i][k];

data2[i] = new\_data[i][k - 1];

}

new\_data[0][k] = data1[0] ^ data2[0];

new\_data[1][k] = data1[1] ^ data2[1];

new\_data[2][k] = data1[2] ^ data2[2];

new\_data[3][k] = data1[3] ^ data2[3];

}

}

new\_k = mat44\_set(

new\_data[0],

new\_data[1],

new\_data[2],

new\_data[3]

);

return new\_k;

}

int main() {

Mat44\* p\_mat = mat44\_set({0, 1, 2, 3}, {4, 5, 6, 7}, {8, 9, 10, 11}, {12, 13, 14, 15});

Mat44\* k\_mat = mat44\_set({1, 1, 1, 1}, {1, 1, 1, 1}, {1, 1, 1, 1}, {1, 1, 1, 1});

std::cout << "Original State :: \n";

mat44\_print(p\_mat);

Mat44\* ark0 = add\_round\_key(p\_mat, k\_mat);

std::cout << "State after initial AddRoundKey :: \n";

mat44\_print(ark0);

Mat44\* sb = sub\_byte(ark0);

std::cout << "State after SubByte :: \n";

mat44\_print(sb);

Mat44\* sr = shift\_rows(sb);

std::cout << "State after ShiftRows :: \n";

mat44\_print(sr);

Mat44\* mc = mix\_columns(sr);

std::cout << "State after MixColumns :: \n";

mat44\_print(mc);

// get next round key

Mat44\* new\_k\_mat = get\_round\_key(k\_mat);

std::cout << "K[4], K[5], K[6], K[7] :: \n";

mat44\_print(new\_k\_mat);

Mat44\* ark1 = add\_round\_key(mc, new\_k\_mat);

std::cout << "State after AddRoundKey :: \n";

mat44\_print(ark1);

std::cout << "Key :: \n";

mat44\_print(k\_mat);

return 0;

}