Министерство образования и науки Украины

Харьковский национальный университет радиоэлектроники

Кафедра БИТ

Отчет

По лабораторной работе по ОАСК/МПА

Тема «Алгоритм Калина»

Выполнили: Проверил:

ст.гр БИКСм-18-1 Мельникова О. А.

Д. Тхао

Харьков 2018

#include <iostream>

#include <cstring>

#include <iomanip>

using namespace std;

#include "endecryption.hpp"

int main() {

testGenInterKey();

testGenInterKey2();

cout << endl << endl;

testGenRoundKeys();

//testGenRoundKeys2();

cout << endl << endl;

testEnDecrypt128128();

system("pause");

return 0;

}

#ifndef \_ENDECRYPTION\_H

#define \_ENDECRYPTION\_H

#include "Kalina/errands.hpp"

#include "Kalina/s\_box.hpp"

#include "Kalina/s\_row.hpp"

#include "Kalina/xor\_rkey.hpp"

#include "Kalina/r\_key.hpp"

void Encrypt128128(uchar \*M, uchar \*Key, uchar \*C);

void Decrypt128128(uchar \*C, uchar \*Key, uchar \*M);

void testEnDecrypt128128();

#endif // !\_ENDECRYPTION\_H

#include "s\_box.hpp"

unsigned char s\_box[4][256] = { ... };

unsigned char s\_box\_inv[4][256] = { ... };

void SubBytes(uchar \*src, int len, uchar s[4][256], uchar \*dst)

{

int n = 0, i = 0;

for (; i< len; i++)

{

n &= 0x3;

dst[i] = s[n][src[i]];

n++;

}

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#include "s\_row.hpp"

//when l = 128, shiftrow == shiftrow\_inv, others case will be added later

void ShiftRow(uchar \*state, int len8)

{

uchar tmp;

switch (len8) {

case 16:

for (int i = 4; i < 8; i++) {

tmp = state[i];

state[i] = state[i + 8];

state[i + 8] = tmp;

}

return;

case 32:

for (int i = 2; i < 8; i += 2) {

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

tmp = state[24 + j];

for (int k = 24 + j; k > 8; k -= 8) {

state[k] = state[k - 8];

}

state[j] = tmp;

}

}

return;

case 64: return;

}

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#include "m\_col.hpp"

#define POLYNOM 0x11D

#define M\_POLY 8 //2^m

#define POW2m\_POLY (1 << M\_POLY)

uchar modPolynom(int w) {

int div = log2(w >> M\_POLY);

int res = w ^ (POLYNOM << div);

if (res < POW2m\_POLY)

return (uchar)res;

else

return modPolynom(res);

}

int coeff\_mul(uchar uc, uchar coeff)

{

switch (coeff) {

case 0x01: return uc;

case 0x04: return uc << 2;

case 0x05: return (uc << 2) ^ uc;

case 0x06: return (uc << 2) ^ (uc << 1);

case 0x07: return (uc << 2) ^ (uc << 1) ^ uc;

case 0x08: return (uc << 3);

default: return 0;

}

}

//0xAD, 0x95, 0x76, 0xA8, 0x2F, 0x49, 0xD7, 0xCA

int coeff\_mul\_inv(uchar uc, uchar coeff)

{

switch (coeff) {

case 0xAD: //1010 1101

return (uc << 7) ^ (uc << 5) ^ (uc << 3) ^ (uc << 2) ^ uc;

case 0x95: //1001 0101

return (uc << 7) ^ (uc << 4) ^ (uc << 2) ^ uc;

case 0x76: //0111 0110

return (uc << 6) ^ (uc << 5) ^ (uc << 4) ^ (uc << 2) ^ (uc << 1);

case 0xA8: //1010 1000

return (uc << 7) ^ (uc << 5) ^ (uc << 3);

case 0x2F: //0010 1111

return (uc << 5) ^ (uc << 3) ^ (uc << 2) ^ (uc << 1) ^ uc;

case 0x49: //0100 1001

return (uc << 6) ^ (uc << 3) ^ uc;

case 0xD7: //1101 0111

return (uc << 7) ^ (uc << 6) ^ (uc << 4) ^ (uc << 2) ^ (uc << 1) ^ uc;

case 0xCA: //1100 1010

return (uc << 7) ^ (uc << 6) ^ (uc << 3) ^ (uc << 1);

default: return 0;

}

}

void MixColum(uchar \*state, int len8) {

uchar v[8][8] = { { 0x01, 0x01, 0x05, 0x01, 0x08, 0x06, 0x07, 0x04 },

{ 0x04, 0x01, 0x01, 0x05, 0x01, 0x08, 0x06, 0x07},

{ 0x07, 0x04, 0x01, 0x01, 0x05, 0x01, 0x08, 0x06},

{ 0x06, 0x07, 0x04, 0x01, 0x01, 0x05, 0x01, 0x08},

{ 0x08, 0x06, 0x07, 0x04, 0x01, 0x01, 0x05, 0x01},

{ 0x01, 0x08, 0x06, 0x07, 0x04, 0x01, 0x01, 0x05},

{ 0x05, 0x01, 0x08, 0x06, 0x07, 0x04, 0x01, 0x01},

{ 0x01, 0x05, 0x01, 0x08, 0x06, 0x07, 0x04, 0x01} };

int tmp[9];

for (int k = 0; k < len8; k += 8) {

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

tmp[i] = coeff\_mul(state[k], v[i][0]);

for (int j = 1; j < 8; j++)

tmp[i] ^= coeff\_mul(state[k + j], v[i][j]);

}

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

state[k + i] = modPolynom(tmp[i]);

}

}

void MixColum\_inv(uchar \*state, int len8)

{

uchar v\_inv[8][8] = { {0xAD, 0x95, 0x76, 0xA8, 0x2F, 0x49, 0xD7, 0xCA},

{ 0xCA, 0xAD, 0x95, 0x76, 0xA8, 0x2F, 0x49, 0xD7},

{ 0xD7, 0xCA, 0xAD, 0x95, 0x76, 0xA8, 0x2F, 0x49},

{ 0x49, 0xD7, 0xCA, 0xAD, 0x95, 0x76, 0xA8, 0x2F},

{ 0x2F, 0x49, 0xD7, 0xCA, 0xAD, 0x95, 0x76, 0xA8},

{ 0xA8, 0x2F, 0x49, 0xD7, 0xCA, 0xAD, 0x95, 0x76},

{ 0x76, 0xA8, 0x2F, 0x49, 0xD7, 0xCA, 0xAD, 0x95},

{ 0x95, 0x76, 0xA8, 0x2F, 0x49, 0xD7, 0xCA, 0xAD}};

int tmp[9];

for (int k = 0; k < len8; k += 8) {

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

tmp[i] = coeff\_mul\_inv(state[k], v\_inv[i][0]);

for (int j = 1; j < 8; j++)

tmp[i] ^= coeff\_mul\_inv(state[k + j], v\_inv[i][j]);

}

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

state[k + i] = modPolynom(tmp[i]);

}

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#include "xor\_rkey.hpp"

void XorKey(uchar \*state, int len8, uchar \*key)

{

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

state[i] ^= key[i];

}

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#include "r\_key.hpp"

//Function of addition modulo 2\*\*64

//note: less significant bytes have smaller indexes

void AddrKey(uchar \*dst, int len, uchar \*src1, uchar \*src2)

{

int rem = 0, tmp;

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

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

tmp = src1[j] + src2[j] + rem;

dst[j] = tmp & 0xff;

rem = tmp >> 8;

}

}

}

//Function of subtraction modulo 2\*\*64

//note: less significant bytes have smaller indexes

void SubrKey(uchar \*dst, int len, uchar \*src1, uchar \*src2)

{

int rem = 0, tmp;

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

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

tmp = src1[j] - src2[j] + rem;

rem = tmp >> 31; // if tmp < 0 ==> rem = -1

if (rem) tmp += 0x100;

dst[j] = tmp & 0xff;

}

}

}

inline void copymasuc(uchar \*src, int begin, int end, uchar \*dst)

{

for (int i = begin, j = 0; i < end; i++) {

dst[j++] = src[i];

}

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

void GenInterKey(uchar \*state, int len8, uchar \*Lk, uchar \*Rk)

{

AddrKey(state, len8, Lk, state);

cout << "AddrKey: "; printuc(state, len8);

SubBytes(state, len8, s\_box, state);

cout << "SubBytes: "; printuc(state, len8);

ShiftRow(state, len8);

cout << "ShiftRow: "; printuc(state, len8);

MixColum(state, len8);

cout << "MixColum: "; printuc(state, len8);

XorKey(state, len8, Rk);

cout << "XorKey: "; printuc(state, len8);

SubBytes(state, len8, s\_box, state);

cout << "SubBytes: "; printuc(state, len8);

ShiftRow(state, len8);

cout << "ShiftRow: "; printuc(state, len8);

MixColum(state, len8);

cout << "MixColum: "; printuc(state, len8);

AddrKey(state, len8, Lk, state);

cout << "AddrKey: "; printuc(state, len8);

SubBytes(state, len8, s\_box, state);

cout << "SubBytes: "; printuc(state, len8);

ShiftRow(state, len8);

cout << "ShiftRow: "; printuc(state, len8);

MixColum(state, len8);

cout << "MixColum: "; printuc(state, len8);

}

//when block size = key size, for this instance k = l = 128

void testGenInterKey()

{

uchar state[16] = { 0x05, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 };

uchar Key[16] = { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,

0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F };

cout << "state: "; printuc(state, 16);

cout << "Key : "; printuc(Key, 16);

cout << endl;

GenInterKey(state, 16, Key, Key);

cout << endl;

cout << "Intermediate Key: "; printuc(state, 16);

}

//when block size = 1/2 key size, for this instance k = 256, l = 128

void testGenInterKey2()

{

char sstate[33] = "07000000000000000000000000000000";

char skey[65] = "000102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F";

uchar state[16];

uchar Key[32];

hex2char(sstate, 16, state);

hex2char(skey, 32, Key);

cout << "State: "; printuc(state, 16);

cout << "Key : "; printuc(Key, 32);

uchar Rk[16], Lk[16];

copymasuc(Key, 0, 16, Lk);

for (int i = 16, j = 0; i < 32;)

Rk[j++] = Key[i++];

cout << endl << endl;

cout << "Lk: "; printuc(Lk, 16);

cout << "Rk: "; printuc(Rk, 16);

GenInterKey(state, 16, Lk, Rk);

}

void rotateR8x(uchar \*src, unsigned int len8, int val, uchar \*dst)

{

int sft = (val >> 3) & 0xF;

if (!sft) {

for (int i = 0; i < len8; i++)

dst[i] = src[i];

return;

}

uchar \*tmp = new uchar[sft];

for (int i = len8 - sft, j = 0; i < len8; i++)

tmp[j++] = src[i];

for (int i = len8 - 1; i >= sft; i--)

dst[i] = src[i - sft];

for (int i = 0; i < sft; i++)

dst[i] = tmp[i];

}

void rotateL8x(uchar \*src, unsigned int len8, int val, uchar \*dst)

{

int sft = (val >> 3) & 0xF;

if (!sft) {

for (int i = 0; i < len8; i++)

dst[i] = src[i];

return;

}

uchar \*tmp = new uchar[sft];

for (int i = 0; i < sft; i++)

tmp[i] = src[i];

for (int i = 0; i < len8 - sft; i++)

dst[i] = src[i + sft];

for (int i = len8 - sft, j = 0; i < len8; i++)

dst[i] = tmp[j++];

}

void evenKeys(uchar \*IK, int len8, uchar \*tmv, uchar \*rkey)

{

uchar \* fiIK = new uchar[len8];

AddrKey(fiIK, len8, tmv, IK);//fi(IntermediateKey) = AddroundKey IK with (v << i/2)

AddrKey(rkey, len8, fiIK, rkey);

SubBytes(rkey, len8, s\_box, rkey);

ShiftRow(rkey, len8);

MixColum(rkey, len8);

XorKey(rkey, len8, fiIK);

SubBytes(rkey, len8, s\_box, rkey);

ShiftRow(rkey, len8);

MixColum(rkey, len8);

AddrKey(rkey, len8, fiIK, rkey);

}

void lsft(uchar \*src, uchar \*dst)

{

int tmp, rem = 0;

for (int i = 15; i >= 0; i--) {

tmp = src[i] << 1;

dst[i] = tmp & 0xFF + rem;

rem = tmp >> 8;

}

}

void GenRoundKeys128128(uchar roundKey[11][16], uchar \* Key)

{

#ifndef BSIZE

#define BSIZE 128

#define KSIZE 128

#define BSIZE8 16

#define KSIZE8 16

#define ODDSFT (BSIZE8 << 1) + 24

uchar IK[BSIZE8] = { 0x05, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 };

GenInterKey(IK, BSIZE8, Key, Key);

uchar v[BSIZE8] = { 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0 };

copymasuc(Key, 0, BSIZE8, roundKey[0]);

evenKeys(IK, BSIZE8, v, roundKey[0]);

for (int i = 1; i < 10; i++) {

rotateL8x(roundKey[i - 1], BSIZE8, ODDSFT, roundKey[i]); //odd keys

lsft(v, v);

rotateR8x(Key, BSIZE8, i << 5, roundKey[++i]); //rotate = Key >>> 32\*i

evenKeys(IK, BSIZE8, v, roundKey[i]);

}

#endif

}

void GenRoundKeys128256(uchar roundKey[15][16], uchar \*Key)

{

//in processing

}

void testGenRoundKeys()

{

uchar roundKey[11][16] = { 0 };

uchar Key[16];

char sKey[33] = "000102030405060708090A0B0C0D0E0F";

hex2char(sKey, 16, Key);

cout << "Key: "; printuc(Key, 16);

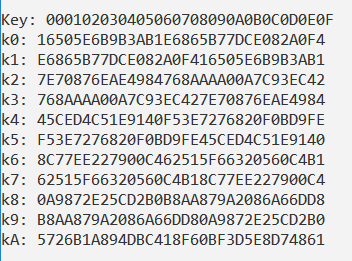
GenRoundKeys128128(roundKey, Key);

for (int i = 0; i < 11; i++) {

cout << "k" << i << ": "; printuc(roundKey[i], 16);

}

}



#include "endecryption.hpp"

void Encrypt128128(uchar \*M, uchar \* Key, uchar \*C)

{

uchar rkey[11][16];

int len8 = 16;

GenRoundKeys128128(rkey, Key);

AddrKey(C, len8, M, rkey[0]);

cout << "addrKey: "; printuc(C, len8);

for (int i = 1; i < 10; i++) {

SubBytes(C, len8, s\_box, C);

cout << "s\_box : "; printuc(C, len8);

ShiftRow(C, len8);

cout << "s\_row : "; printuc(C, len8);

MixColum(C, len8);

cout << "m\_col : "; printuc(C, len8);

XorKey(C, len8, rkey[i]);

cout << "xorKey : "; printuc(C, len8);

}

SubBytes(C, len8, s\_box, C);

cout << "s\_box : "; printuc(C, len8);

ShiftRow(C, len8);

cout << "s\_row : "; printuc(C, len8);

MixColum(C, len8);

cout << "m\_col : "; printuc(C, len8);

AddrKey(C, len8, C, rkey[10]);

cout << "addrKey: "; printuc(C, len8);

}

void Decrypt128128(uchar \*C, uchar \*Key, uchar \*M)

{

uchar rkey[11][16];

int len8 = 16;

GenRoundKeys128128(rkey, Key);

SubrKey(M, len8, C, rkey[10]);

cout << "sudrKey : "; printuc(M, len8);

MixColum\_inv(M, len8);

cout << "invm\_col: "; printuc(M, len8);

ShiftRow(M, len8);

cout << "invs\_row: "; printuc(M, len8);

SubBytes(M, len8, s\_box\_inv, M);

cout << "invs\_box: "; printuc(M, len8);

for (int i = 9; i > 0; i--) {

XorKey(M, len8, rkey[i]);

cout << "xorKey : "; printuc(M, len8);

MixColum\_inv(M, len8);

cout << "invm\_col: "; printuc(M, len8);

ShiftRow(M, len8);

cout << "invs\_row: "; printuc(M, len8);

SubBytes(M, len8, s\_box\_inv, M);

cout << "invs\_box: "; printuc(M, len8);

}

SubrKey(M, len8, M, rkey[0]);

cout << "sudrKey : "; printuc(C, len8);

}

void testEnDecrypt128128()

{

char sM[33] = "101112131415161718191A1B1C1D1E1F";

char sKey[33] = "000102030405060708090A0B0C0D0E0F";

uchar M[16], Key[16], C[16], M1[16];

hex2char(sM, 16, M);

hex2char(sKey, 16, Key);

cout << endl << "Encrypt: " << endl;

cout << "M : "; printuc(M, 16);

cout << "Key: "; printuc(Key, 16);

Encrypt128128(M, Key, C);

cout << "C : "; printuc(C, 16);

cout << endl;

cout << endl << "Decrypt: " << endl;

cout << "C : "; printuc(C, 16);

cout << "Key: "; printuc(Key, 16);

Decrypt128128(C, Key, M1);

cout << "M1 : "; printuc(M1, 16);

cout << "M : "; printuc(M, 16);

}