AES.h

/\*MIT License

Copyright (c) 2019 SergeyBel

Permission is hereby granted, free of charge, to any person obtaining a copy

of this software and associated documentation files (the "Software"), to deal

in the Software without restriction, including without limitation the rights

to use, copy, modify, merge, publish, distribute, sublicense, and/or sell

copies of the Software, and to permit persons to whom the Software is

furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all

copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR

IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,

FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE

AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER

LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,

OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE

SOFTWARE\*/

#ifndef AES\_H

#define AES\_H

#include <cstring>

#include <iostream>

#include <stdio.h>

**using** **namespace** std;

**class** AES {

**private**:

**int** Nb;

**int** Nk;

**int** Nr;

**unsigned** **int** blockBytesLen;

**void** SubBytes(**unsigned** **char** \*\*state);

**void** ShiftRow(**unsigned** **char** \*\*state, **int** i, **int** n); // shift row i on n positions

**void** ShiftRows(**unsigned** **char** \*\*state);

**unsigned** **char** xtime(**unsigned** **char** b); // multiply on x

**unsigned** **char** mul\_bytes(**unsigned** **char** a, **unsigned** **char** b);

**void** MixColumns(**unsigned** **char** \*\*state);

**void** MixSingleColumn(**unsigned** **char** \*r);

**void** AddRoundKey(**unsigned** **char** \*\*state, **unsigned** **char** \*key);

**void** SubWord(**unsigned** **char** \*a);

**void** RotWord(**unsigned** **char** \*a);

**void** XorWords(**unsigned** **char** \*a, **unsigned** **char** \*b, **unsigned** **char** \*c);

**void** Rcon(**unsigned** **char** \* a, **int** n);

**void** InvSubBytes(**unsigned** **char** \*\*state);

**void** InvMixColumns(**unsigned** **char** \*\*state);

**void** InvShiftRows(**unsigned** **char** \*\*state);

**unsigned** **char**\* PaddingNulls(**unsigned** **char** **in**[], **unsigned** **int** inLen, **unsigned** **int** alignLen);

**unsigned** **int** GetPaddingLength(**unsigned** **int** len);

**void** KeyExpansion(**unsigned** **char** key[], **unsigned** **char** w[]);

**void** EncryptBlock(**unsigned** **char** **in**[], **unsigned** **char** **out**[], **unsigned** **char** key[]);

**void** DecryptBlock(**unsigned** **char** **in**[], **unsigned** **char** **out**[], **unsigned** **char** key[]);

**void** XorBlocks(**unsigned** **char** \*a, **unsigned** **char** \* b, **unsigned** **char** \*c, **unsigned** **int** len);

**public**:

AES(**int** keyLen = 256);

**unsigned** **char** \*EncryptECB(**unsigned** **char** **in**[], **unsigned** **int** inLen, **unsigned** **char** key[], **unsigned** **int** &outLen);

**unsigned** **char** \*DecryptECB(**unsigned** **char** **in**[], **unsigned** **int** inLen, **unsigned** **char** key[]);

**unsigned** **char** \*EncryptCBC(**unsigned** **char** **in**[], **unsigned** **int** inLen, **unsigned** **char** key[], **unsigned** **char** \* iv, **unsigned** **int** &outLen);

**unsigned** **char** \*DecryptCBC(**unsigned** **char** **in**[], **unsigned** **int** inLen, **unsigned** **char** key[], **unsigned** **char** \* iv);

**unsigned** **char** \*EncryptCFB(**unsigned** **char** **in**[], **unsigned** **int** inLen, **unsigned** **char** key[], **unsigned** **char** \* iv, **unsigned** **int** &outLen);

**unsigned** **char** \*DecryptCFB(**unsigned** **char** **in**[], **unsigned** **int** inLen, **unsigned** **char** key[], **unsigned** **char** \* iv);

**void** printHexArray (**unsigned** **char** a[], **unsigned** **int** n);

};

**const** **unsigned** **char** sbox[16][16] = {

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

};

**const** **unsigned** **char** inv\_sbox[16][16] = {

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,

};

#endif // AES\_H

AES.cpp

/\*MIT License

Copyright (c) 2019 SergeyBel

Permission is hereby granted, free of charge, to any person obtaining a copy

of this software and associated documentation files (the "Software"), to deal

in the Software without restriction, including without limitation the rights

to use, copy, modify, merge, publish, distribute, sublicense, and/or sell

copies of the Software, and to permit persons to whom the Software is

furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all

copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR

IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,

FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE

AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER

LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,

OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE

SOFTWARE\*/

#include "AES.h"

AES::AES(**int** keyLen) {

**this**->Nb = 4;

**switch** (keyLen) {

**case** 128:

**this**->Nk = 4;

**this**->Nr = 10;

**break**;

**case** 192:

**this**->Nk = 6;

**this**->Nr = 12;

**break**;

**case** 256:

**this**->Nk = 8;

**this**->Nr = 14;

**break**;

**default**:

**throw** "Incorrect key length";

}

blockBytesLen = 4 \* **this**->Nb \* **sizeof**(**unsigned** **char**);

}

**unsigned** **char** \* AES::EncryptECB(**unsigned** **char** in[], **unsigned** **int** inLen, **unsigned** **char** key[], **unsigned** **int** &outLen) {

outLen = GetPaddingLength(inLen);

**unsigned** **char** \*alignIn = PaddingNulls(in, inLen, outLen);

**unsigned** **char** \*out = **new** **unsigned** **char**[outLen];

**unsigned** **char** \*roundKeys = **new** **unsigned** **char**[4 \* Nb \* (Nr + 1)];

KeyExpansion(key, roundKeys);

**for** (**unsigned** **int** i = 0; i < outLen; i+= blockBytesLen) {

EncryptBlock(alignIn + i, out + i, roundKeys);

}

**delete**[] alignIn;

**delete**[] roundKeys;

**return** out;

}

**unsigned** **char** \* AES::DecryptECB(**unsigned** **char** in[], **unsigned** **int** inLen, **unsigned** **char** key[]) {

**unsigned** **char** \*out = **new** **unsigned** **char**[inLen];

**unsigned** **char** \*roundKeys = **new** **unsigned** **char**[4 \* Nb \* (Nr + 1)];

KeyExpansion(key, roundKeys);

**for** (**unsigned** **int** i = 0; i < inLen; i+= blockBytesLen) {

DecryptBlock(in + i, out + i, roundKeys);

}

**delete**[] roundKeys;

**return** out;

}

**unsigned** **char** \*AES::EncryptCBC(**unsigned** **char** in[], **unsigned** **int** inLen, **unsigned** **char** key[], **unsigned** **char** \* iv, **unsigned** **int** &outLen) {

outLen = GetPaddingLength(inLen);

**unsigned** **char** \*alignIn = PaddingNulls(in, inLen, outLen);

**unsigned** **char** \*out = **new** **unsigned** **char**[outLen];

**unsigned** **char** \*block = **new** **unsigned** **char**[blockBytesLen];

**unsigned** **char** \*roundKeys = **new** **unsigned** **char**[4 \* Nb \* (Nr + 1)];

KeyExpansion(key, roundKeys);

memcpy(block, iv, blockBytesLen);

**for** (**unsigned** **int** i = 0; i < outLen; i+= blockBytesLen) {

XorBlocks(block, alignIn + i, block, blockBytesLen);

EncryptBlock(block, out + i, roundKeys);

memcpy(block, out + i, blockBytesLen);

}

**delete**[] block;

**delete**[] alignIn;

**delete**[] roundKeys;

**return** out;

}

**unsigned** **char** \*AES::DecryptCBC(**unsigned** **char** in[], **unsigned** **int** inLen, **unsigned** **char** key[], **unsigned** **char** \* iv) {

**unsigned** **char** \*out = **new** **unsigned** **char**[inLen];

**unsigned** **char** \*block = **new** **unsigned** **char**[blockBytesLen];

**unsigned** **char** \*roundKeys = **new** **unsigned** **char**[4 \* Nb \* (Nr + 1)];

KeyExpansion(key, roundKeys);

memcpy(block, iv, blockBytesLen);

**for** (**unsigned** **int** i = 0; i < inLen; i+= blockBytesLen) {

DecryptBlock(in + i, out + i, roundKeys);

XorBlocks(block, out + i, out + i, blockBytesLen);

memcpy(block, in + i, blockBytesLen);

}

**delete**[] block;

**delete**[] roundKeys;

**return** out;

}

**unsigned** **char** \*AES::EncryptCFB(**unsigned** **char** in[], **unsigned** **int** inLen, **unsigned** **char** key[], **unsigned** **char** \* iv, **unsigned** **int** &outLen) {

outLen = GetPaddingLength(inLen);

**unsigned** **char** \*alignIn = PaddingNulls(in, inLen, outLen);

**unsigned** **char** \*out = **new** **unsigned** **char**[outLen];

**unsigned** **char** \*block = **new** **unsigned** **char**[blockBytesLen];

**unsigned** **char** \*encryptedBlock = **new** **unsigned** **char**[blockBytesLen];

**unsigned** **char** \*roundKeys = **new** **unsigned** **char**[4 \* Nb \* (Nr + 1)];

KeyExpansion(key, roundKeys);

memcpy(block, iv, blockBytesLen);

**for** (**unsigned** **int** i = 0; i < outLen; i+= blockBytesLen) {

EncryptBlock(block, encryptedBlock, roundKeys);

XorBlocks(alignIn + i, encryptedBlock, out + i, blockBytesLen);

memcpy(block, out + i, blockBytesLen);

}

**delete**[] block;

**delete**[] encryptedBlock;

**delete**[] alignIn;

**delete**[] roundKeys;

**return** out;

}

**unsigned** **char** \*AES::DecryptCFB(**unsigned** **char** in[], **unsigned** **int** inLen, **unsigned** **char** key[], **unsigned** **char** \* iv) {

**unsigned** **char** \*out = **new** **unsigned** **char**[inLen];

**unsigned** **char** \*block = **new** **unsigned** **char**[blockBytesLen];

**unsigned** **char** \*encryptedBlock = **new** **unsigned** **char**[blockBytesLen];

**unsigned** **char** \*roundKeys = **new** **unsigned** **char**[4 \* Nb \* (Nr + 1)];

KeyExpansion(key, roundKeys);

memcpy(block, iv, blockBytesLen);

**for** (**unsigned** **int** i = 0; i < inLen; i+= blockBytesLen) {

EncryptBlock(block, encryptedBlock, roundKeys);

XorBlocks(in + i, encryptedBlock, out + i, blockBytesLen);

memcpy(block, in + i, blockBytesLen);

}

**delete**[] block;

**delete**[] encryptedBlock;

**delete**[] roundKeys;

**return** out;

}

**unsigned** **char** \* AES::PaddingNulls(**unsigned** **char** in[], **unsigned** **int** inLen, **unsigned** **int** alignLen) {

**unsigned** **char** \*alignIn = **new** **unsigned** **char**[alignLen];

memcpy(alignIn, in, inLen);

memset(alignIn + inLen, 0x00, alignLen - inLen);

**return** alignIn;

}

**unsigned** **int** AES::GetPaddingLength(**unsigned** **int** len) {

**unsigned** **int** lengthWithPadding = (len / blockBytesLen);

**if** (len % blockBytesLen) {

lengthWithPadding++;

}

lengthWithPadding \*= blockBytesLen;

**return** lengthWithPadding;

}

**void** AES::EncryptBlock(**unsigned** **char** in[], **unsigned** **char** out[], **unsigned** **char** \*roundKeys) {

**unsigned** **char** \*\*state = **new** **unsigned** **char** \*[4];

state[0] = **new** **unsigned** **char**[4 \* Nb];

**int** i, j, round;

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

state[i] = state[0] + Nb \* i;

}

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

**for** (j = 0; j < Nb; j++) {

state[i][j] = in[i + 4 \* j];

}

}

AddRoundKey(state, roundKeys);

**for** (round = 1; round <= Nr - 1; round++) {

SubBytes(state);

ShiftRows(state);

MixColumns(state);

AddRoundKey(state, roundKeys + round \* 4 \* Nb);

}

SubBytes(state);

ShiftRows(state);

AddRoundKey(state, roundKeys + Nr \* 4 \* Nb);

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

**for** (j = 0; j < Nb; j++) {

out[i + 4 \* j] = state[i][j];

}

}

**delete**[] state[0];

**delete**[] state;

}

**void** AES::DecryptBlock(**unsigned** **char** in[], **unsigned** **char** out[], **unsigned** **char** \*roundKeys) {

**unsigned** **char** \*\*state = **new** **unsigned** **char** \*[4];

state[0] = **new** **unsigned** **char**[4 \* Nb];

**int** i, j, round;

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

state[i] = state[0] + Nb \* i;

}

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

**for** (j = 0; j < Nb; j++) {

state[i][j] = in[i + 4 \* j];

}

}

AddRoundKey(state, roundKeys + Nr \* 4 \* Nb);

**for** (round = Nr - 1; round >= 1; round--) {

InvSubBytes(state);

InvShiftRows(state);

AddRoundKey(state, roundKeys + round \* 4 \* Nb);

InvMixColumns(state);

}

InvSubBytes(state);

InvShiftRows(state);

AddRoundKey(state, roundKeys);

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

**for** (j = 0; j < Nb; j++) {

out[i + 4 \* j] = state[i][j];

}

}

**delete**[] state[0];

**delete**[] state;

}

**void** AES::SubBytes(**unsigned** **char** \*\*state) {

**int** i, j;

**unsigned** **char** t;

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

**for** (j = 0; j < Nb; j++) {

t = state[i][j];

state[i][j] = sbox[t / 16][t % 16];

}

}

}

**void** AES::ShiftRow(**unsigned** **char** \*\*state, **int** i, **int** n) { // shift row i on n positions

**unsigned** **char** \*tmp = **new** **unsigned** **char**[Nb];

**for** (**int** j = 0; j < Nb; j++) {

tmp[j] = state[i][(j + n) % Nb];

}

memcpy(state[i], tmp, Nb \* **sizeof**(**unsigned** **char**));

**delete**[] tmp;

}

**void** AES::ShiftRows(**unsigned** **char** \*\*state) {

ShiftRow(state, 1, 1);

ShiftRow(state, 2, 2);

ShiftRow(state, 3, 3);

}

**unsigned** **char** AES::xtime(**unsigned** **char** b) { // multiply on x

**return** (b << 1) ^ (((b >> 7) & 1) \* 0x1b);

}

/\* Implementation taken from https://en.wikipedia.org/wiki/Rijndael\_mix\_columns#Implementation\_example \*/

**void** AES::MixSingleColumn(**unsigned** **char** \*r) {

**unsigned** **char** a[4];

**unsigned** **char** b[4];

**unsigned** **char** c;

**unsigned** **char** h;

/\* The array 'a' is simply a copy of the input array 'r'

\* The array 'b' is each element of the array 'a' multiplied by 2

\* in Rijndael's Galois field

\* a[n] ^ b[n] is element n multiplied by 3 in Rijndael's Galois field \*/

**for**(c=0;c<4;c++) {

a[c] = r[c];

/\* h is 0xff if the high bit of r[c] is set, 0 otherwise \*/

h = (**unsigned** **char**)((**signed** **char**)r[c] >> 7); /\* arithmetic right shift, thus shifting in either zeros or ones \*/

b[c] = r[c] << 1; /\* implicitly removes high bit because b[c] is an 8-bit char, so we xor by 0x1b and not 0x11b in the next line \*/

b[c] ^= 0x1B & h; /\* Rijndael's Galois field \*/

}

r[0] = b[0] ^ a[3] ^ a[2] ^ b[1] ^ a[1]; /\* 2 \* a0 + a3 + a2 + 3 \* a1 \*/

r[1] = b[1] ^ a[0] ^ a[3] ^ b[2] ^ a[2]; /\* 2 \* a1 + a0 + a3 + 3 \* a2 \*/

r[2] = b[2] ^ a[1] ^ a[0] ^ b[3] ^ a[3]; /\* 2 \* a2 + a1 + a0 + 3 \* a3 \*/

r[3] = b[3] ^ a[2] ^ a[1] ^ b[0] ^ a[0]; /\* 2 \* a3 + a2 + a1 + 3 \* a0 \*/

}

/\* Performs the mix columns step. Theory from: https://en.wikipedia.org/wiki/Advanced\_Encryption\_Standard#The\_MixColumns\_step \*/

**void** AES::MixColumns(**unsigned** **char**\*\* state) {

**unsigned** **char** \*temp = **new** **unsigned** **char**[4];

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

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

temp[j] = state[j][i]; //place the current state column in temp

}

MixSingleColumn(temp); //mix it using the wiki implementation

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

state[j][i] = temp[j]; //when the column is mixed, place it back into the state

}

}

**delete**[] temp;

}

**void** AES::AddRoundKey(**unsigned** **char** \*\*state, **unsigned** **char** \*key) {

**int** i, j;

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

**for** (j = 0; j < Nb; j++)

{

state[i][j] = state[i][j] ^ key[i + 4 \* j];

}

}

}

**void** AES::SubWord(**unsigned** **char** \*a) {

**int** i;

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

a[i] = sbox[a[i] / 16][a[i] % 16];

}

}

**void** AES::RotWord(**unsigned** **char** \*a) {

**unsigned** **char** c = a[0];

a[0] = a[1];

a[1] = a[2];

a[2] = a[3];

a[3] = c;

}

**void** AES::XorWords(**unsigned** **char** \*a, **unsigned** **char** \*b, **unsigned** **char** \*c) {

**int** i;

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

c[i] = a[i] ^ b[i];

}

}

**void** AES::Rcon(**unsigned** **char** \* a, **int** n) {

**int** i;

**unsigned** **char** c = 1;

**for** (i = 0; i < n - 1; i++) {

c = xtime(c);

}

a[0] = c;

a[1] = a[2] = a[3] = 0;

}

**void** AES::KeyExpansion(**unsigned** **char** key[], **unsigned** **char** w[]) {

**unsigned** **char** \*temp = **new** **unsigned** **char**[4];

**unsigned** **char** \*rcon = **new** **unsigned** **char**[4];

**int** i = 0;

**while** (i < 4 \* Nk) {

w[i] = key[i];

i++;

}

i = 4 \* Nk;

**while** (i < 4 \* Nb \* (Nr + 1)) {

temp[0] = w[i - 4 + 0];

temp[1] = w[i - 4 + 1];

temp[2] = w[i - 4 + 2];

temp[3] = w[i - 4 + 3];

**if** (i / 4 % Nk == 0) {

RotWord(temp);

SubWord(temp);

Rcon(rcon, i / (Nk \* 4));

XorWords(temp, rcon, temp);

}

**else** **if** (Nk > 6 && i / 4 % Nk == 4) {

SubWord(temp);

}

w[i + 0] = w[i - 4 \* Nk] ^ temp[0];

w[i + 1] = w[i + 1 - 4 \* Nk] ^ temp[1];

w[i + 2] = w[i + 2 - 4 \* Nk] ^ temp[2];

w[i + 3] = w[i + 3 - 4 \* Nk] ^ temp[3];

i += 4;

}

**delete** []rcon;

**delete** []temp;

}

**void** AES::InvSubBytes(**unsigned** **char** \*\*state) {

**int** i, j;

**unsigned** **char** t;

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

**for** (j = 0; j < Nb; j++) {

t = state[i][j];

state[i][j] = inv\_sbox[t / 16][t % 16];

}

}

}

**unsigned** **char** AES::mul\_bytes(**unsigned** **char** a, **unsigned** **char** b) { // multiplication a and b in galois field

**unsigned** **char** p = 0;

**unsigned** **char** high\_bit\_mask = 0x80;

**unsigned** **char** high\_bit = 0;

**unsigned** **char** modulo = 0x1B; /\* x^8 + x^4 + x^3 + x + 1 \*/

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

**if** (b & 1) {

p ^= a;

}

high\_bit = a & high\_bit\_mask;

a <<= 1;

**if** (high\_bit) {

a ^= modulo;

}

b >>= 1;

}

**return** p;

}

**void** AES::InvMixColumns(**unsigned** **char** \*\*state) {

**unsigned** **char** s[4], s1[4];

**int** i, j;

**for** (j = 0; j < Nb; j++) {

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

s[i] = state[i][j];

}

s1[0] = mul\_bytes(0x0e, s[0]) ^ mul\_bytes(0x0b, s[1]) ^ mul\_bytes(0x0d, s[2]) ^ mul\_bytes(0x09, s[3]);

s1[1] = mul\_bytes(0x09, s[0]) ^ mul\_bytes(0x0e, s[1]) ^ mul\_bytes(0x0b, s[2]) ^ mul\_bytes(0x0d, s[3]);

s1[2] = mul\_bytes(0x0d, s[0]) ^ mul\_bytes(0x09, s[1]) ^ mul\_bytes(0x0e, s[2]) ^ mul\_bytes(0x0b, s[3]);

s1[3] = mul\_bytes(0x0b, s[0]) ^ mul\_bytes(0x0d, s[1]) ^ mul\_bytes(0x09, s[2]) ^ mul\_bytes(0x0e, s[3]);

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

state[i][j] = s1[i];

}

}

}

**void** AES::InvShiftRows(**unsigned** **char** \*\*state) {

ShiftRow(state, 1, Nb - 1);

ShiftRow(state, 2, Nb - 2);

ShiftRow(state, 3, Nb - 3);

}

**void** AES::XorBlocks(**unsigned** **char** \*a, **unsigned** **char** \* b, **unsigned** **char** \*c, **unsigned** **int** len) {

**for** (**unsigned** **int** i = 0; i < len; i++) {

c[i] = a[i] ^ b[i];

}

}

**void** AES::printHexArray (**unsigned** **char** a[], **unsigned** **int** n) {

**for** (**unsigned** **int** i = 0; i < n; i++) {

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

}

}

Bundle.h

#ifndef Bundle\_h

#define Bundle\_h

#include <iostream>

#include <QDebug>

#include <QString>

#include "AES.h"

**using** **namespace** std;

**class** Bundle {

**char** name[128] = {};

**char** username[128] = {};

**char** password[128] = {};

**public**:

Bundle(string name = string(""), string username = string(""), string password = string("")) {

strlcpy(**this**->name, name.c\_str(), **sizeof**(name));

strlcpy(**this**->username, username.c\_str(), **sizeof**(username));

strlcpy(**this**->password, password.c\_str(), **sizeof**(password));

};

**void** setName(**char**\*);

**void** setUsername(**char**\*);

**void** setPassword(**char**\*);

**char**\* getName();

**char**\* getUsername();

**char**\* getPassword();

**void** encrypt(QString);

**void** decrypt(QString);

~Bundle(){};

};

#endif /\* Bundle\_h \*/

Bundle.cpp

#include "Bundle.h"

**void** Bundle::setName(**char**\* string) {

strlcpy(name, string, **sizeof**(name));

}

**void** Bundle::setUsername(**char**\* string) {

strlcpy(username, string, **sizeof**(username));

}

**void** Bundle::setPassword(**char**\* string) {

strlcpy(password, string, **sizeof**(password));

}

**char**\* Bundle::getName() {

**return** name;

}

**char**\* Bundle::getUsername() {

**return** username;

}

**char**\* Bundle::getPassword() {

**return** password;

}

**void** Bundle::encrypt(QString password){

**int** keyLenght = 128;

AES aes(keyLenght);

**unsigned** **int** outputSize;

**unsigned** **char** key[] = { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f };

**unsigned** **char**\* tempString = aes.EncryptECB((**unsigned** **char**\*)name, **sizeof**(name), key, outputSize);

strlcpy(name, (**char**\*)tempString, **sizeof**(name));

tempString = aes.EncryptECB((**unsigned** **char**\*)username, **sizeof**(username), key, outputSize);

strlcpy(username, (**char**\*)tempString, **sizeof**(username));

tempString = aes.EncryptECB((**unsigned** **char**\*)**this**->password, **sizeof**(**this**->password), key, outputSize);

strlcpy(**this**->password, (**char**\*)tempString, **sizeof**(**this**->password));

}

**void** Bundle::decrypt(QString password){

**int** keyLenght = 128;

AES aes(keyLenght);

**unsigned** **int** outputSize;

**unsigned** **char** key[] = { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f };

**unsigned** **char**\* tempString = aes.DecryptECB((**unsigned** **char**\*)name, **sizeof**(name), key);

strlcpy(name, (**char**\*)tempString, **sizeof**(name));

tempString = aes.DecryptECB((**unsigned** **char**\*)username, **sizeof**(username), key);

strlcpy(username, (**char**\*)tempString, **sizeof**(username));

tempString = aes.DecryptECB((**unsigned** **char**\*)**this**->password, **sizeof**(**this**->password), key);

strlcpy(**this**->password, (**char**\*)tempString, **sizeof**(**this**->password));

}

File.h

#ifndef FILE\_H

#define FILE\_H

#include <sys/stat.h>

#include <fcntl.h>

#include <unistd.h>

#include <aio.h>

#include <QString>

#include <QFileDialog>

#include <QDebug>

**using** **namespace** std;

**class** File: QObject {

QString fileName;

**int** descriptor;

aiocb aio = {};

**public**:

File() {}

**bool** newFile();

**bool** openFile();

**bool** saveFile();

**bool** saveFileAs();

**void** closeFile();

**bool** openFileAndGetDescriptor(**bool**);

**void** read(**void**\*, **long** **long** size, **long** **long** offset);

**void** write(**void**\*, **long** **long** size, **long** **long** offset);

~File() {

close(descriptor);

}

};

#endif // FILE\_H

File.cpp

#include "File.h"

**bool** File::newFile() {

fileName = QFileDialog::getSaveFileName(**NULL**, tr("Create new file"), "/", tr("Password Manager Data (\*.pmd)"));

**if**(!fileName.size()){

**return** **false**;

}

openFileAndGetDescriptor(**true**);

closeFile();

**return** **true**;

}

**bool** File::openFile() {

fileName = QFileDialog::getOpenFileName(**NULL**, tr("Open file"), "/", tr("Password Manager Data (\*.pmd)"));

**if**(!fileName.size()){

**return** **false**;

}

**return** openFileAndGetDescriptor(**false**);

}

**bool** File::saveFile() {

**return** openFileAndGetDescriptor(**true**);

}

**bool** File::saveFileAs() {

fileName = QFileDialog::getSaveFileName(**NULL**, tr("Choose or creare file"), "/", tr("Password Manager Data (\*.pmd)"));

**if**(!fileName.size()){

**return** **false**;

}

**return** openFileAndGetDescriptor(**true**);

}

**void** File::closeFile() {

**if**(!close(descriptor)) {

qDebug() << "Error at close: " << strerror(errno);

}

}

**bool** File::openFileAndGetDescriptor(**bool** clear) {

**if**(clear){

descriptor = open(fileName.toStdString().c\_str(), O\_RDWR | O\_CREAT | O\_TRUNC, S\_IROTH | S\_IRWXG | S\_IRWXU);

} **else** {

descriptor = open(fileName.toStdString().c\_str(), O\_RDWR | O\_CREAT, S\_IROTH | S\_IRWXG | S\_IRWXU);

}

aio.aio\_fildes = descriptor;

**if**(!descriptor) {

qDebug() << "Error at open: " << strerror(errno);

**return** **false**;

} **else** {

**return** **true**;

}

}

**void** File::read(**void**\* source, **long** **long** size = 16, **long** **long** offset = 0) {

aio.aio\_buf = source;

aio.aio\_offset = offset;

aio.aio\_nbytes = size;

aio.aio\_lio\_opcode = LIO\_READ;

**if** (aio\_read(&aio) == -1) {

qDebug() << "Error at aio\_read(): " << strerror(errno);

}

**while**(aio\_error(&aio) == EINPROGRESS);

aio\_return(&aio);

}

**void** File::write(**void**\* source, **long** **long** size = 16, **long** **long** offset = 0) {

aio.aio\_buf = source;

aio.aio\_offset = offset;

aio.aio\_nbytes = size;

aio.aio\_lio\_opcode = LIO\_WRITE;

**if** (aio\_write(&aio) == -1) {

qDebug() << "Error at aio\_write(): " << strerror(errno);

}

**while**(aio\_error(&aio) == EINPROGRESS);

aio\_return(&aio);

}

Settings.h

#ifndef SETTINGS\_H

#define SETTINGS\_H

**class** Settings {

**int** keySize;

**bool** openOnEnter;

**public**:

Settings();

**void** loadSettingsFromFile();

**void** saveSettingsToFile();

};

#endif // SETTINGS\_H

BundleUndo.h

#ifndef BUNDLEUNDO\_H

#define BUNDLEUNDO\_H

**enum** UndoOperation {

create,

remove,

edit,

};

**class** BundleUndo {

**enum** UndoOperation type;

**public**:

BundleUndo();

**void** undo();

**void** redo();

};

#endif // BUNDLEUNDO\_H

PasswordWindow.h

#ifndef PASSWORDWINDOW\_H

#define PASSWORDWINDOW\_H

#include <QDialog>

**namespace** Ui {

**class** passwordWindow;

}

**class** PasswordWindow : **public** QDialog {

Q\_OBJECT

**public**:

**explicit** PasswordWindow(QWidget \*parent = **nullptr**);

~PasswordWindow();

**private**:

Ui::passwordWindow \*ui;

QString password;

};

#endif // PASSWORDWINDOW\_H

PasswordWindow.cpp

#include "PasswordWindow.h"

#include "ui\_passwordWindow.h"

PasswordWindow::PasswordWindow(QWidget \*parent) : QDialog(parent), ui(**new** Ui::passwordWindow) {

ui->setupUi(**this**);

}

PasswordWindow::~PasswordWindow() {

**delete** ui;

}

EditWindow.h

#ifndef EDITWINDOW\_H

#define EDITWINDOW\_H

#include <QDialog>

#include <QRegularExpressionValidator>

#include <QMessageBox>

**namespace** Ui {

**class** EditWindow;

}

**class** EditWindow : **public** QDialog {

Q\_OBJECT

**public**:

**explicit** EditWindow(QString \*name = **NULL**, QString \*username = **NULL**, QString \*password = **NULL**, QWidget \*parent = **nullptr**);

~EditWindow();

**private** slots:

**void** on\_buttonBox\_accepted();

**private**:

Ui::EditWindow \*ui;

QString \*name;

QString \*username;

QString \*password;

};

#endif // EDITWINDOW\_H

EditWindow.cpp

#include "EditWindow.h"

#include "ui\_EditWindow.h"

EditWindow::EditWindow(QString \*name, QString \*username, QString \*password, QWidget \*parent) :QDialog(parent), ui(**new** Ui::EditWindow) {

**this**->name = name;

**this**->username = username;

**this**->password = password;

ui->setupUi(**this**);

ui->usernameLine->setValidator(**new** QRegularExpressionValidator( QRegularExpression("[^ ]{1,128}"), **this**));

ui->passwordLine->setValidator(**new** QRegularExpressionValidator( QRegularExpression("[^ ]{1,128}"), **this**));

ui->nameLine->setText(\*name);

ui->usernameLine->setText(\*username);

ui->passwordLine->setText(\*password);

}

EditWindow::~EditWindow() {

**delete** ui;

}

**void** EditWindow::on\_buttonBox\_accepted() {

**if**(!ui->nameLine->text().size()) {

QMessageBox::warning(**this**, "Error", "Name cannot be empty!");

**return**;

}

**if**(!ui->usernameLine->text().size()) {

QMessageBox::warning(**this**, "Error", "Username cannot be empty!");

**return**;

}

**if**(!ui->passwordLine->text().size()) {

QMessageBox::warning(**this**, "Error", "Password cannot be empty!");

**return**;

}

\*name = ui->nameLine->text();

\*username = ui->usernameLine->text();

\*password = ui->passwordLine->text();

accept();

}

StartWindow.h

#ifndef STARTWINDOW\_H

#define STARTWINDOW\_H

#include <QMainWindow>

#include <QStandardItemModel>

#include <QMessageBox>

#include <QFileDialog>

#include "PasswordWindow.h"

#include "EditWindow.h"

#include "Bundle.h"

#include "File.h"

#include "Settings.h"

#include "BundleUndo.h"

QT\_BEGIN\_NAMESPACE

**namespace** Ui { **class** StartWindow; }

QT\_END\_NAMESPACE

**class** StartWindow : **public** QMainWindow {

Q\_OBJECT

**public**:

StartWindow(QWidget \*parent = **nullptr**);

~StartWindow();

**private** slots:

**void** on\_actionOpen\_triggered();

**void** on\_actionSave\_triggered();

**void** on\_actionDelete\_triggered();

**void** on\_actionEdit\_triggered();

**void** on\_bundleTable\_doubleClicked(**const** QModelIndex &index);

**void** on\_actionSave\_as\_triggered();

**void** on\_actionClose\_triggered();

**void** on\_actionNew\_triggered();

**void** on\_actionUndo\_triggered();

**void** on\_actionAdd\_triggered();

**void** on\_actionRedo\_triggered();

**private**:

Ui::StartWindow \*ui;

PasswordWindow \*passwordWindow;

EditWindow \*editWindow;

QStandardItemModel \*bundlesModel;

File file;

vector<Bundle> bundles;

**void** updateTable();

**void** bundleFunctions(**bool**);

**void** undoFunctions(**bool**);

**void** fileFunctions(**bool**);

};

#endif // STARTWINDOW\_H

StartWindow.cpp

#include "startWindow.h"

#include "./ui\_startWindow.h"

StartWindow::StartWindow(QWidget \*parent): QMainWindow(parent), ui(**new** Ui::StartWindow) {

ui->setupUi(**this**);

updateTable();

fileFunctions(**false**);

bundleFunctions(**true**);

undoFunctions(**true**);

}

StartWindow::~StartWindow() {

**delete** ui;

}

**void** StartWindow::bundleFunctions(**bool** temp) {

ui->actionDelete->setEnabled(temp);

ui->actionEdit->setEnabled(temp);

}

**void** StartWindow::undoFunctions(**bool** temp) {

ui->actionRedo->setEnabled(temp);

ui->actionUndo->setEnabled(temp);

}

**void** StartWindow::fileFunctions(**bool** temp) {

bundleFunctions(temp);

undoFunctions(temp);

ui->actionClose->setEnabled(temp);

ui->actionSave->setEnabled(temp);

ui->actionSave\_as->setEnabled(temp);

}

**void** StartWindow::updateTable() {

QStandardItem \*item;

bundlesModel = **new** QStandardItemModel;

**const** QStringList bundleTableHeadersList = {"Name", "Username", "Password"};

bundlesModel->setHorizontalHeaderLabels(bundleTableHeadersList);

bundlesModel->setParent(**this**);

**for** (**int** i = 0; i < bundles.size(); i++) {

item = **new** QStandardItem(QString(bundles[i].getName()));

bundlesModel->setItem(i, 0, item);

item = **new** QStandardItem(QString(bundles[i].getUsername()));

bundlesModel->setItem(i, 1, item);

item = **new** QStandardItem(QString(bundles[i].getPassword()));

bundlesModel->setItem(i, 2, item);

}

ui->bundleTable->setModel(bundlesModel);

ui->bundleTable->horizontalHeader()->setSectionResizeMode(QHeaderView::Stretch);

ui->bundleTable->verticalHeader()->hide();

ui->bundleTable->sortByColumn(0, Qt::SortOrder::AscendingOrder);

}

**void** StartWindow::on\_actionNew\_triggered(){

**if**(file.newFile()) {

fileFunctions(**true**);

}

}

**void** StartWindow::on\_actionOpen\_triggered() {

**if**(file.openFile()) {

bundles.clear();

**int** rows = 0;

file.read(&rows, **sizeof**(**int**), 0);

**for** (**int** i = 0; i < rows; i++) {

Bundle\* bundle = **new** Bundle();

file.read(bundle, **sizeof**(Bundle), **sizeof**(**int**) + **sizeof**(Bundle) \* i);

bundles.push\_back(\*bundle);

}

updateTable();

file.closeFile();

fileFunctions(**true**);

}

}

**void** StartWindow::on\_actionSave\_triggered() {

**if**(file.saveFile()) {

**int** rows = bundlesModel->rowCount();

file.write(&rows, **sizeof**(**int**), 0);

**for** (**int** i = 0; i < rows; i++) {

Bundle\* bundle = **new** Bundle(bundlesModel->item(i, 0)->text().toStdString(), bundlesModel->item(i, 1)->text().toStdString(), bundlesModel->item(i, 2)->text().toStdString());

file.write(bundle, **sizeof**(Bundle), **sizeof**(**int**) + **sizeof**(Bundle) \* i);

}

file.closeFile();

}

}

**void** StartWindow::on\_actionSave\_as\_triggered() {

file.saveFileAs();

**int** rows = bundlesModel->rowCount();

file.write(&rows, **sizeof**(**int**), 0);

**for** (**int** i = 0; i < rows; i++) {

Bundle\* bundle = **new** Bundle(bundlesModel->item(i, 0)->text().toStdString(), bundlesModel->item(i, 1)->text().toStdString(), bundlesModel->item(i, 2)->text().toStdString());

file.write(bundle, **sizeof**(Bundle), **sizeof**(**int**) + **sizeof**(Bundle) \* i);

}

file.closeFile();

}

**void** StartWindow::on\_actionClose\_triggered() {

file.closeFile();

fileFunctions(**false**);

bundles.clear();

bundlesModel->clear();

updateTable();

}

**void** StartWindow::on\_actionUndo\_triggered(){

bundles[0].encrypt(tr(""));

updateTable();

}

**void** StartWindow::on\_actionRedo\_triggered() {

bundles[0].decrypt(tr(""));

updateTable();

}

**void** StartWindow::on\_actionAdd\_triggered(){

QModelIndexList selection = ui->bundleTable->selectionModel()->selectedRows();

**int** pasteRow = 0;

**if**(selection.size() >= 1){

pasteRow = selection.last().row() + 1;

} **else** {

pasteRow = 0;

}

QString name;

QString username;

QString password;

editWindow = **new** EditWindow(&name, &username, &password, **this**);

**if**(editWindow->exec()) {

QStandardItem \*item;

bundlesModel->insertRow(pasteRow);

item = **new** QStandardItem(name);

bundlesModel->setItem(pasteRow, 0, item);

item = **new** QStandardItem(username);

bundlesModel->setItem(pasteRow, 1, item);

item = **new** QStandardItem(password);

bundlesModel->setItem(pasteRow, 2, item);

}

}

**void** StartWindow::on\_actionDelete\_triggered() {

QModelIndexList selection = ui->bundleTable->selectionModel()->selectedRows();

**if**(selection.empty()) {

**return**;

}

**int** i = 0;

**for** (; i < selection.size(); i++) {

bundlesModel->removeRow(selection[i].row() - i);

}

**if**(bundlesModel->rowCount() != 0) { //set selection on item that now on delelted item place

i = selection.last().row() - i + 1; //deleted row index

**if**(i > bundlesModel->rowCount() - 1) {

i--;

}

ui->bundleTable->selectRow(i);

}

}

**void** StartWindow::on\_actionEdit\_triggered() {

QModelIndexList selection = ui->bundleTable->selectionModel()->selectedRows();

**if**(selection.size() > 1) {

QMessageBox::warning(**this**, "Error", "You cannot edit more than one item at a time!");

}

**int** row = selection[0].row();

QString name = bundlesModel->item(row, 0)->text();

QString username = bundlesModel->item(row, 1)->text();

QString password = bundlesModel->item(row, 2)->text();

editWindow = **new** EditWindow(&name, &username, &password, **this**);

editWindow->exec();

bundlesModel->item(0, 0)->setText(name);

bundlesModel->item(0, 1)->setText(username);

bundlesModel->item(0, 2)->setText(password);

updateTable();

}

**void** StartWindow::on\_bundleTable\_doubleClicked(**const** QModelIndex &index) {

on\_actionEdit\_triggered();

}