g***Министерство образования Республики Беларусь***

***Учреждение образования***

***«Брестский государственный технический университет»***

***Кафедра ИИТ***

**Лабораторная работа №2**

**По дисциплине КМЗИ за IV семестр**

**Тема: «Блочные шифры»**

**Выполнил:**

Студент группы ИИ-15 (1)

2-го курса

Волк И. А.

**Проверил:**

Хацкевич М. В.

Брест 2019

Цель работы: изучить работу блочных шифров.

Ход работы

Реализуем блочный шифр в режиме сцепления блоков. Для шифрования не представляется возможным использовать шифр замены по таблице: выход из функции шифрования должен составлять такое же количество бит как и вход (в данном случае n = 8), но таблица сама по себе имеет n \* m возможных выходов, где n – количество возможных выходов, а m – количество столбцов, а значит если вход лежит в приделах 0 .. 256, то выход будет лежать в пределах 0 .. 256 \* m. Поэтому условие выполняется только при m = 1, тогда как по варианту m = 4, что препятствует выполнению условия, приведенного выше.

Было принято решение использовать систему Виженера.

Листинг:

#include "pch.h"

#include <iostream>

#include <utility>

#include <string>

#include <cmath>

#include <algorithm>

#include <boost/numeric/ublas/io.hpp>

#include <boost/numeric/ublas/matrix.hpp>

#include <boost/random.hpp>

#include <boost/algorithm/string.hpp>

/\*\* **Definitions** \*\*/

using namespace boost::numeric::ublas;

typedef boost::random::mt19937 boost\_rand;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\* **Classes** \*\*/

class ICipher;

class VigenereCipher;

/\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\* **Functions (declarations)** \*\*/

std::string CBC(const std::string& sText, char cIV, char cMode, ICipher& cipher); // for one char

std::string CBC\_block(const std::string& sText, std::string sIV, char cMode, ICipher& cipher); // for string

std::vector<std::string> SplitIntoBlocks(const std::string& str, int nBlockSize);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\* **Classes (declarations)** \*\*/

class ICipher

{

public:

virtual char Encrypt(char cPlainBlock) = 0;

virtual char Decrypt(char cEncryptedBlock) = 0;

virtual std::string EncryptBlock(const std::string& sPlainBlock) = 0;

virtual std::string DecryptBlock(const std::string& sEncryptedBlock) = 0;

};

class VigenereCipher : public ICipher

{

matrix<char> m\_mxTable;

std::string m\_sKey;

int nCurrentSymbol; // state of the key for next plain symbol

public:

VigenereCipher(const std::string& sKey);

virtual char Encrypt(char cPlainBlock);

virtual char Decrypt(char cEncryptedBlock);

virtual std::string EncryptBlock(const std::string& sPlainBlock);

virtual std::string DecryptBlock(const std::string& sEncryptedBlock);

void Reload();

};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int main()

{

std::string sKey("key\_phrase");

std::string sIV; sIV.resize(sKey.length());

std::generate(sIV.begin(), sIV.end(), []() -> char {

static boost\_rand gen;

return gen() % 256;

});

std::string sPlainText;

std::getline(std::cin, sPlainText);

// Encryption

VigenereCipher vc(sKey);

std::string sCipherText = CBC\_block(sPlainText, sIV, 'e', vc);

std::cout << "Encrypted: " << sCipherText << std::endl;

// Decrypted

sPlainText = CBC\_block(sCipherText, sIV, 'd', vc);

std::cout << "Decrypted: " << sPlainText << std::endl;

}

/\*\* **Classes (definitions)** \*\*/

VigenereCipher::VigenereCipher(const std::string& sKey)

: m\_mxTable(sKey.length(), 256), nCurrentSymbol(0)

{

m\_sKey = boost::to\_lower\_copy(sKey);

// fill our matrix

boost\_rand gen;

for (int i = 0; i < m\_mxTable.size1(); i++)

{

std::string sAlphabet;

for (int i = 0; i < 256; i++)

sAlphabet.push\_back(i);

for (int j = 0; j < m\_mxTable.size2(); j++)

{

int idx = gen() % sAlphabet.length();

m\_mxTable(i, j) = sAlphabet[idx];

sAlphabet.erase(idx, 1);

}

}

}

char VigenereCipher::Encrypt(char cPlainBlock)

{

unsigned int nPlainBlock = cPlainBlock;

nPlainBlock &= 0xFF;

char cResult = m\_mxTable(nCurrentSymbol, nPlainBlock);

nCurrentSymbol = (nCurrentSymbol + 1) % m\_sKey.length();

return cResult;

}

char VigenereCipher::Decrypt(char cEncryptedBlock)

{

char cResult;

for (int i = 0; i < m\_mxTable.size2(); i++)

{

if (m\_mxTable(nCurrentSymbol, i) == cEncryptedBlock)

{

cResult = i;

break;

}

}

nCurrentSymbol = (nCurrentSymbol + 1) % m\_sKey.length();

return cResult;

}

std::string VigenereCipher::EncryptBlock(const std::string& sPlainBlock)

{

Reload();

std::string sResult;

for (char c : sPlainBlock)

sResult.push\_back(Encrypt(c));

return sResult;

}

std::string VigenereCipher::DecryptBlock(const std::string& sEncryptedBlock)

{

Reload();

std::string sResult;

for (char c : sEncryptedBlock)

sResult.push\_back(Decrypt(c));

return sResult;

}

void VigenereCipher::Reload()

{

nCurrentSymbol = 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\* **Function (definitions)** \*\*/

std::string CBC(const std::string& sText, char cIV, char cMode, ICipher& cipher)

{

std::string sResult;

char cXOR = cIV;

switch (cMode)

{

case 'e':

{

for (char ch : sText)

{

char cXOR\_res = ch ^ cXOR;

char cEncrypted = cipher.Encrypt(cXOR\_res);

sResult.push\_back(cEncrypted);

cXOR = cEncrypted;

}

}

break;

case 'd':

{

for (char ch : sText)

{

char cDecrypted = cipher.Decrypt(ch);

char cXOR\_res = cXOR ^ cDecrypted;

sResult.push\_back(cXOR\_res);

cXOR = ch;

}

}

break;

default:

break;

}

return sResult;

}

std::vector<std::string> SplitIntoBlocks(const std::string& str, int nBlockSize)

{

std::vector<std::string> vecResult;

std::string sCurrent;

for (char ch : str)

{

sCurrent.push\_back(ch);

if (sCurrent.length() == nBlockSize)

{

vecResult.push\_back(sCurrent);

sCurrent.clear();

}

}

if (!sCurrent.empty())

vecResult.push\_back(sCurrent);

return vecResult;

}

std::string operator^(const std::string& sLeft, const std::string& sRight)

{

std::string sResult;

int nLength = std::min(sLeft.length(), sRight.length());

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

{

sResult.push\_back(sLeft[i] ^ sRight[i]);

}

return sResult;

}

std::string CBC\_block(const std::string& sText, std::string sIV, char cMode, ICipher& cipher)

{

int nBlockSize = sIV.length();

std::vector<std::string> vecBlocks = SplitIntoBlocks(sText, nBlockSize);

std::string sResult;

std::string sXOR = sIV;

bool fShortBlock = false; // is the last block too short?

if (vecBlocks.back().length() < nBlockSize)

fShortBlock = true;

switch (cMode)

{

case 'e': // encryption

{

for (auto it = vecBlocks.begin(); it != vecBlocks.end(); it++)

{

if (fShortBlock && it == vecBlocks.end() - 1)

{

sXOR = cipher.EncryptBlock(sXOR);

sResult.append(\*it ^ sXOR);

break;

}

std::string sXORed = \*it ^ sXOR;

std::string sEncrypted = cipher.EncryptBlock(sXORed);

sResult.append(sEncrypted);

sXOR = sEncrypted;

}

}

break;

case 'd': // decryption

{

for (auto it = vecBlocks.begin(); it != vecBlocks.end(); it++)

{

if (fShortBlock && it == vecBlocks.end() - 1)

{

sXOR = cipher.EncryptBlock(sXOR);

sResult.append(\*it ^ sXOR);

break;

}

std::string sDecrypted = cipher.DecryptBlock(\*it);

sResult.append(sDecrypted ^ sXOR);

sXOR = \*it;

}

}

break;

default:

break;

}

return sResult;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Результаты:



Вывод: по ходу лабораторной работы научился использовать режим шифрования CBC и систему шифрования Виженера.