МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ



УНИВЕРСИТЕТ им. Р.Е.АЛЕКСЕЕВА

ОТЧЁТ

по лабораторной работе №4

по дисциплине

Защита информации

Вариант 5

РУКОВОДИТЕЛЬ:

Капранов С.Н.

СТУДЕНТ:

Докукин Д.В.

ГРУППА:

20-ПО

Нижний Новгород 2023-2024

Задание

Лабораторная работа 4 – Режимы шифра DES

Задание №4.2

Реализовать стандарт шифрования данных DES в режиме «Обратная связь по шифру».

Описание алгоритма

В 1977 году Национальное бюро Стандартов США (NBS) опубликовало стандарт шифрования данных Data Encryption Standard (DES), предназначенный для использования в государственных и правительственных учреждениях США для защиты от несанкционированного доступа важной, но несекретной информации. Алгоритм, положенный в основу стандарта, распространялся достаточно быстро, и уже в 1980 году был одобрен ANSI. С этого момента DES превращается в стандарт не только по названию (Data Encryption Standard), но и фактически.

Основные достоинства алгоритма DES:

* используется только один ключ длиной 56 битов;
* зашифровав сообщение с помощью одного пакета, для расшифровки вы можете использовать любой другой;
* относительная простота алгоритма обеспечивает высокую скорость обработки информации;
* достаточно высокая стойкость алгоритма.

DES осуществляет шифрование 64-битовых блоков данных с помощью 56-битового ключа. Расшифрование в DES является операцией обратной шифрованию и выполняется путем повторения операций шифрования в обратной последовательности

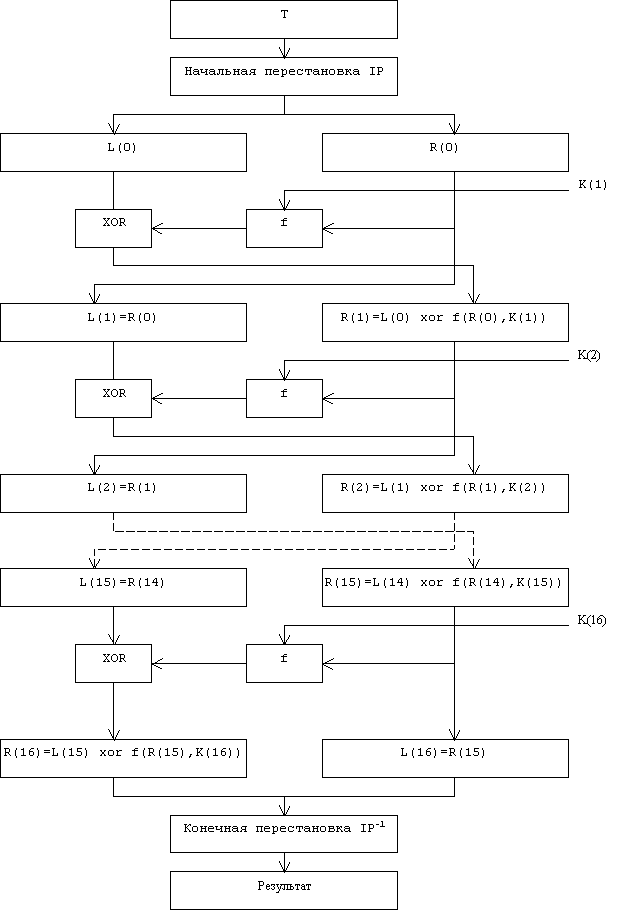
Процесс шифрования заключается в начальной перестановке битов 64-битового блока, шестнадцати циклах шифрования и, наконец, обратной перестановки битов (рис.1).

Рис.1. Обобщенная схема шифрования в алгоритме DES



Структура алгоритма DES приведена на рис.2.

Рис.2. Структура алгоритма шифрования DES



Пусть из файла считан очередной 8-байтовый блок T, который преобразуется с помощью матрицы начальной перестановки IP (табл.1) следующим образом: бит 58 блока T становится битом 1, бит 50 - битом 2 и т.д., что даст в результате: T(0) = IP(T).

Полученная последовательность битов T(0) разделяется на две последовательности по 32 бита каждая: L(0) - левые или старшие биты, R(0) - правые или младшие биты.

Затем выполняется шифрование, состоящее из 16 итераций. Результат i-й итерации описывается следующими формулами:

L(i) = R(i-1)

R(i) = L(i-1) xor f(R(i-1), K(i)),

где xor - операция ИСКЛЮЧАЮЩЕЕ ИЛИ.

На 16-й итерации получают последовательности R(16) и L(16) (без перестановки), которые конкатенируют в 64-битовую последовательность R(16)L(16).

Затем позиции битов этой последовательности переставляют в соответствии с обратной матрицей IP.

Процесс расшифрования данных является инверсным по отношению к процессу шифрования.

Итеративный процесс расшифрования может быть описан следующими формулами:

R(i-1)=L(i), i=1,2, ...,16;

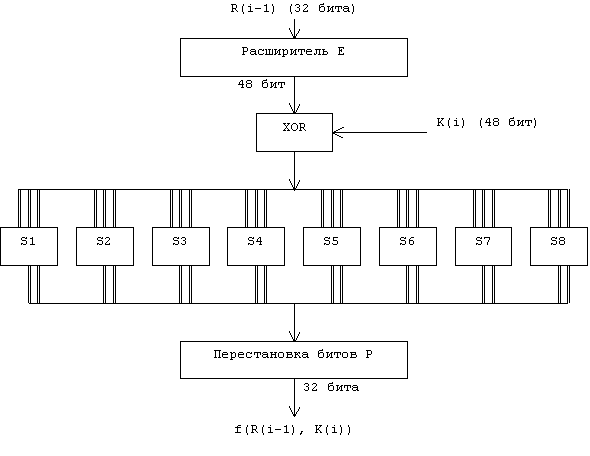
L(i-1)=R(i) xor f(L(i),K(i)), i=1,2, ... ,16.

На 16-й итерации получают последовательности L(0) и R(0), которые конкатенируют в 64-битовую последовательность L(0)R(0).

Затем позиции битов этой последовательности переставляют в соответствии с матрицей IP. Результат такой перестановки - исходная 64-битовая последовательность.

Теперь рассмотрим функцию шифрования f(R(i-1),K(i)). Схематически она показана на рис. 3.

Рис.3. Вычисление функции f(R(i-1),K(i))



Для вычисления значения функции f используются следующие функции-матрицы:

* Е - расширение 32-битовой последовательности до 48-битовой,
* S1, S2, ... , S8 - преобразование 6-битового блока в 4-битовый,
* Р - перестановка бит в 32-битовой последовательности.

Результат функции Е(R(i-1)) есть 48-битовая последовательность, которая складывается по модулю 2 (операция xor) с 48-битовым ключом К(i). Получается 48-битовая последовательность, которая разбивается на восемь 6-битовых блоков B(1)B(2)B(3)B(4)B(5)B(6)B(7)B(8). То есть:

E(R(i-1)) xor K(i) = B(1)B(2)...B(8).

Функции S1, S2, ... , S8 определяются по специальной таблице. Пусть на вход функции-матрицы Sj поступает 6-битовый блок B(j) = b1b2b3b4b5b6, тогда двухбитовое число b1b6 указывает номер строки матрицы, а b2b3b4b5 - номер столбца. Результатом Sj(B(j)) будет 4-битовый элемент, расположенный на пересечении указанных строки и столбца таблицы. Применив операцию выбора к каждому из 6-битовых блоков B(1), B(2), ..., B(8), получаем 32-битовую последовательность S1(B(1))S2(B(2))S3(B(3))...S8(B(8)).

Наконец, для получения результата функции шифрования надо переставить биты этой последовательности. Для этого применяется функция перестановки P.

Таким образом,

f(R(i-1), K(i)) = P(S1(B(1)),...S8(B(8)))

Чтобы завершить описание алгоритма шифрования данных, осталось привести алгоритм получения 48-битовых ключей К(i), i=1...16. На каждой итерации используется новое значение ключа K(i), которое вычисляется из начального ключа K. K представляет собой 64-битовый блок с восемью битами контроля по четности, расположенными в позициях 8,16,24,32,40,48,56,64.

Для удаления контрольных битов и перестановки остальных используется функция G первоначальной подготовки ключа.

Результат преобразования G(K) разбивается на два 28-битовых блока C(0) и D(0).

После определения C(0) и D(0) рекурсивно определяются C(i) и D(i), i=1...16. Для этого применяют циклический сдвиг влево на один или два бита в зависимости от номера итерации, определённого в соответствующей таблице.

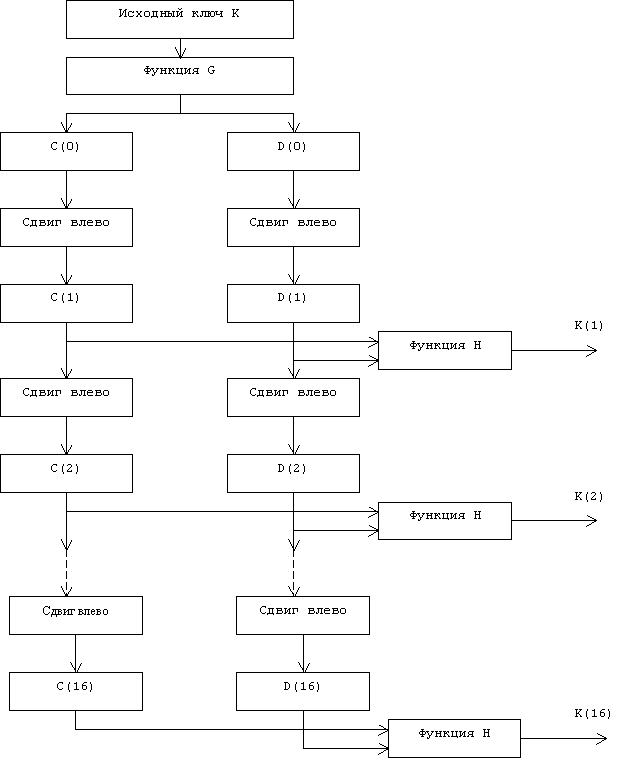
Полученное значение вновь "перемешивается" в соответствии с матрицей H.

Таким образом:

K(i) = H(C(i)D(i))

Блок-схема алгоритма вычисления ключа приведена на рис.4.

Рис.4. Блок-схема алгоритма вычисления ключа K(i)



Восстановление исходного текста осуществляется по этому алгоритму, но вначале используется ключ K(15), затем - K(14) и так далее.

Режим работы алгоритма DES – DES-CFB

Цифровая обратная связь (Cipher Feedback).

В этом режиме размер блока может отличаться от 64. Исходный файл M считывается последовательными t-битовыми блоками (t <= 64): M = M(1)M(2)...M(n) (остаток дописывается нулями или пробелами).

64-битовый сдвиговый регистр (входной блок) вначале содержит вектор инициализации IV, выравненный по правому краю. Для каждого сеанса шифрования используется новый IV.

Для всех i = 1...n блок шифртекста C(i) определяется следующим образом:

C(i) = M(i) xor P(i-1),

где P(i-1) - старшие t битов операции DES(С(i-1)), причем C(0)=IV.

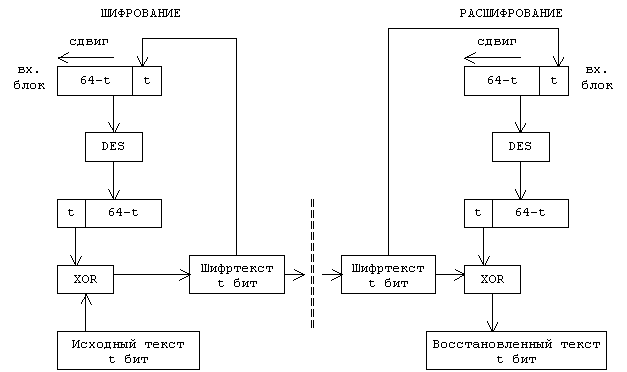
Обновление сдвигового регистра осуществляется путем удаления его старших t битов и дописывания справа C(i).

Восстановление зашифрованных данных также не представляет труда: P(i-1) и C(i) вычисляются аналогичным образом и

M(i) = C(i) xor P(i-1).

Блок-схема режима CFB приведена на рис.5.

Рис.5. Работа алгоритма DES в режиме ECB



Демонстрация работы программы

Для демонстрации, согласно требованиям, будут зашифрованы следующие сообщения:

1. Нижегородский Государственный Технический Университет
2. Докукин Дмитрий Владимирович

Ключ будет сгенерирован автоматически, при помощи приложения. Он хранится, как текстовый документ.

Зашифрованное сообщение получается в формате бинарного файла.

Пример 1

Шифрование сообщения: Нижегородский Государственный Технический Университет

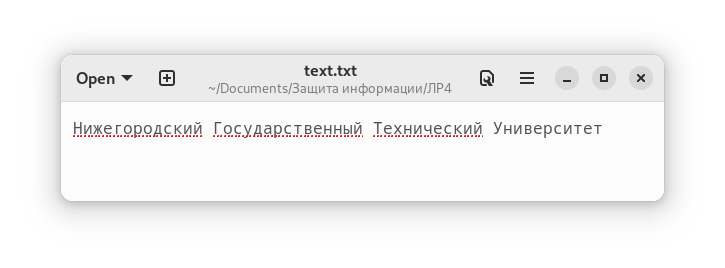
Команда

./des-cfb keygen

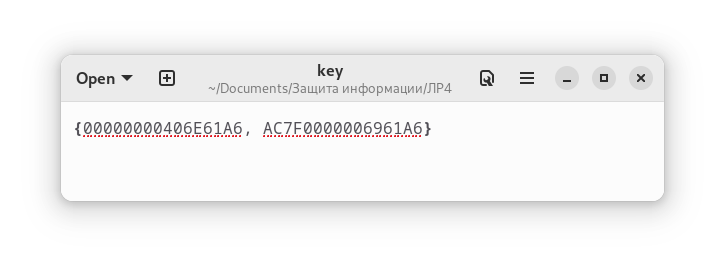
позволяет сгенерировать подходящий ключ для шифрования. Без указания названия файла ключа, по умолчанию будет создан файл «key». Помимо ключа, генерируется также и начальный вектор IV.

В файле text.txt записано сообщение. В файле key записан ключ. Шифртекст будет записан в файл encrypted

Содержимое файла text.txt:



Содержимое файла key:

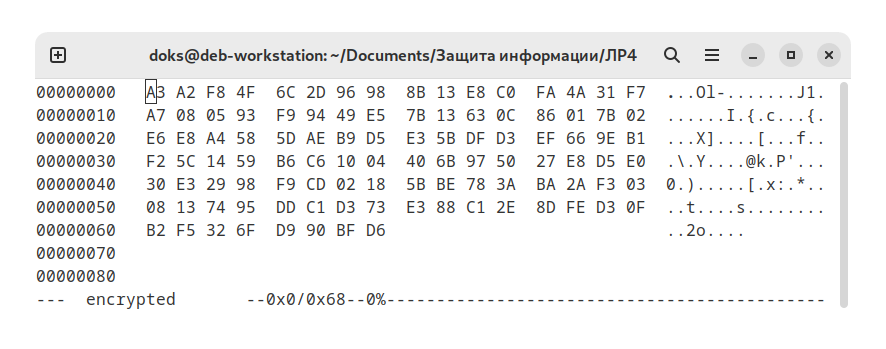


Помимо самого ключа, здесь также содержится начальный вектор IV.

Следующая команда вызывает шифрование указанного сообщения:

./des-cfb encrypt text.txt encrypted key=key

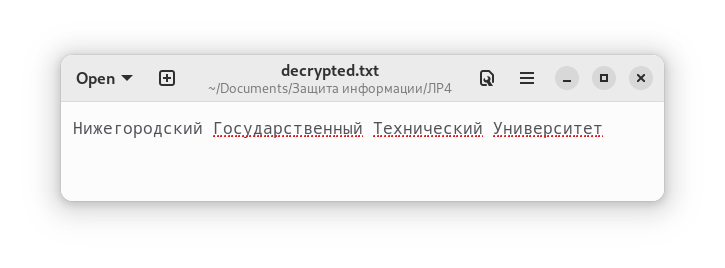
С помощью консольной утилиты hexedit можно посмотреть содержимое зашифрованного файла:



При расшифровке, указываю операцию расшифровки, файл с шифртекстом (encrypted),имя файла для расшифрованного сообщения (decrypted.txt) и файл с ключом (key):

./des-cfb decrypt encrypted decrypted.txt key=key

Содержимое файла decrypted.txt:

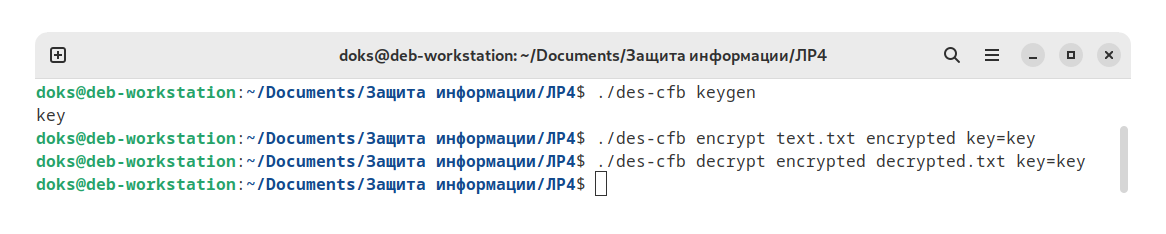


Расшифровка прошла успешно.

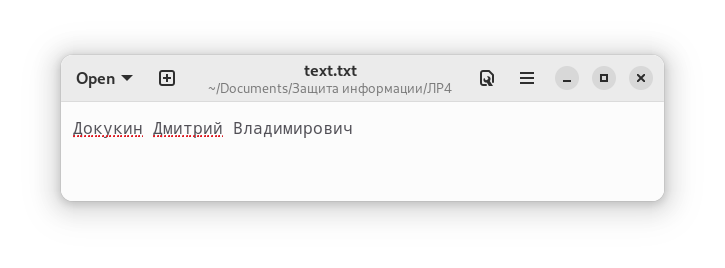
Пример 2

Шифрование и расшифровка сообщения: Докукин Дмитрий Владимирович

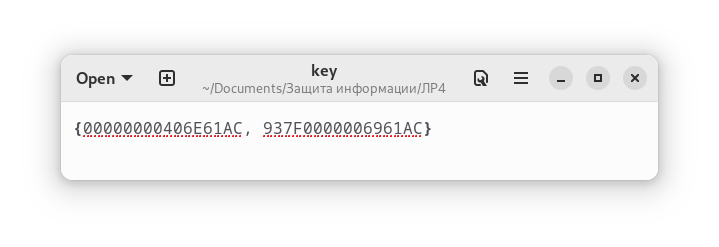
Все операции показаны на одном скриншоте:



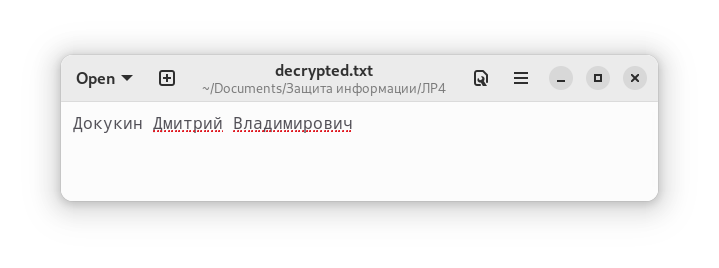
Содержимое файла text.txt:



Содержимое файла key:



Содержимое файла decrypted.txt:



Процессы прошли успешно.

Программа

В программе представлены три класса и два пространство имён:

InputHandler – отвечает за обработку ввода. Проверяет входные данные на корректность, хранит в себе имена файлов, а также имеет методы извлечения ключа из файла.

DES – отвечает за шифрование и расшифровку. Хранит в себе значение ключа. Имеет методы шифрования, расшифрования и генерации ключа.

FileManager – позволяет считывать данные из файлов и сохранять результаты шифрования и расшифрования.

Экземпляры данных классов объявляются в файле main.cpp, где происходит последовательный вызов нужных методов для выполнения требуемой операции.

DES\_functions – пространство имён, содержащее в себе реализацию процедур шифрования.

KeyHandling – пространство имён, содержащее функции импорта, экспорта и обработки ключей.

Листинг

Файл DES.h

#ifndef DES\_H

#define DES\_H

#include <iostream>

#include <vector>

#include <array>

#include "DES\_exception.h"

#include "KeyPair.h"

class DES

{

public:

enum Mode

{

NONE,

ECB,

CFB

};

private:

Mode m\_mode;

std::array<char, 8> m\_key;

std::array<char, 8> m\_iv;

public:

DES(DES::Mode mode) : m\_mode(mode) {}

std::vector<char> encrypt(std::vector<char> data);

std::vector<char> encrypt\_cfb(std::vector<char> data);

std::vector<char> decrypt(std::vector<char> data);

std::vector<char> decrypt\_cfb(std::vector<char> data);

auto genKey() -> std::array<char, 7>;

auto genIV() -> std::array<char, 8>;

auto getKey() -> std::array<char, 8>;

auto getKeyString(char separator = 0) -> std::string;

auto getKeyPair() -> KeyPair;

auto setKey(std::array<char, 8> key) -> void;

auto setKeyPair(KeyPair& pair) -> DES&;

auto extractKey(std::array<char, 8> eKey) -> std::array<char, 7>;

//std::ostream& printKey(std::ostream& out);

};

#endif // DES\_H

Файл DES\_functions.h

#ifndef DES\_FUNCTIONS\_H

#define DES\_FUNCTIONS\_H

#include <iostream>

#include <array>

#include <vector>

namespace DES\_functions

{

std::array<char, 8> initialPermutation(std::array<char, 8> T);

std::ostream& printBinary(std::ostream& out, char letter);

std::array<char, 8> permutation(std::array<char, 8> T, std::array<int, 64> IP); // !

std::array<char, 8> encryption(std::array<char, 8> TP, std::array<char, 8> key);

std::array<char, 7> prepareKey(std::array<char, 8> key); // !

std::array<char, 7> shiftCD(std::array<char, 7> CD, int shift); // !

std::array<char, 6> getK(std::array<char, 7> CDS); // !

std::array<char, 4> getL(std::array<char, 8> T); // !

std::array<char, 4> getR(std::array<char, 8> T); // !

std::array<char, 4> f(std::array<char, 4> RorL, std::array<char, 6> K); // !

std::array<char, 6> expand(std::array<char, 4> RorL); // !

std::array<char, 6> xor48(std::array<char, 6> RorLE, std::array<char, 6> K); // !

std::array<char, 8> getB(std::array<char, 6> RorLEK); // !

std::array<char, 4> shrink(std::array<char, 8> B); // !

std::array<char, 4> transposition(std::array<char, 4> BS); // !

std::array<char, 8> combineLR(std::array<char, 4> L, std::array<char, 4> R); // !

std::array<char, 8> decryption(std::array<char, 8> revTP, std::array<char, 8> K);

std::array<char, 7> reverseShiftCD(std::array<char, 7> CD, int shift); // !

std::vector<char> encrypt(std::vector<char>& data, std::array<char, 8> key);

std::vector<char> decrypt(std::vector<char>& data, std::array<char, 8> key);

int getRandomNumber(int min, int max);

std::array<char, 7> genKey();

std::array<char, 8> expandKey(std::array<char, 7> key);

std::array<char, 7> extractKey(std::array<char, 8> key);

}

#endif // DES\_FUNCTIONS\_H

Файл InputHandler.h

#ifndef INPUTHANDLER\_H

#define INPUTHANDLER\_H

#include <string>

#include <fstream>

#include <array>

class InputHandler

{

public:

enum class Status

{

NONE, // None

S, // Only a Source provided

SD, // Source and Destination provided

KG, // Keygen requested

KGD, // Keygen requested with Destination file

INVALID // Invalid input

};

enum class Command

{

HELP,

KEYGEN,

ENCRYPT,

DECRYPT,

NONE

};

private:

// Members

std::string m\_sourceName;

std::string m\_destinationName;

std::string m\_keyPathName;

//std::array<char, 8> m\_key;

Command m\_command;

bool m\_keyFlag;

// Private methods

auto isArgKey(char\* str) -> bool;

auto checkArgs(int argc, char\* argv[]) -> Status;

auto getKeyPath(char\* arg) -> std::string;

//auto extractKey(const std::string keyPath) -> std::string;

//auto convertKey(const std::string keyString) -> std::array<char, 8>;

public:

InputHandler():

m\_command(Command::NONE),

m\_keyFlag(false) {}

bool process(int argc, char\* argv[]);

void printErrorMessage();

// Getters

auto getSource() const -> std::string;

auto getDestination() const -> std::string;

auto getKeyPath() const -> std::string;

auto getCommand() const -> Command;

//auto getKey() -> std::array<char, 8>;

// Key Managment

bool isKeySet();

~InputHandler() {}

};

#endif // INPUTHANDLER\_H

Файл FileManager.h

#ifndef FILEMANAGER\_H

#define FILEMANAGER\_H

#include <vector>

#include <string>

#include <fstream>

#include <exception>

class FileManager

{

private:

std::ifstream m\_src;

std::ofstream m\_dest;

public:

FileManager(std::string dest);

FileManager(std::string src, std::string dest);

std::vector<char> getData(size\_t byteSize = 0);

void removeZeros(std::vector<char>& data);

void setData(std::vector<char> data);

void setData(std::string data);

void setData(char\* data, size\_t size);

~FileManager()

{

if (m\_src.is\_open())

m\_src.close();

if (m\_dest.is\_open())

m\_dest.close();

}

};

class FileManagerException : public std::exception

{

private:

std::string m\_error;

public:

FileManagerException(std::string msg) : m\_error(msg) {}

const char\* what() const noexcept { return m\_error.c\_str(); }

};

#endif // FILEMANAGER\_H

Файл KeyHandling.h

#ifndef KEYHANDLING\_H

#define KEYHANDLING\_H

#include <string>

#include <fstream>

#include <exception>

#include <array>

#include "KeyPair.h"

namespace KeyHandling

{

KeyPair importKey(const char\* path);

KeyPair importKey(const std::string& path);

KeyPair importKey(const std::string&& path);

void exportKey(const std::string& path, const KeyPair& key);

void exportKey(const std::string& path, const KeyPair&& key);

class Exception : public std::exception

{

private:

std::string m\_error;

public:

Exception(std::string msg) : m\_error(msg) {}

const char\* what() const noexcept { return m\_error.c\_str(); }

};

namespace \_\_private\_\_

{

bool isCharNumber(char ch);

bool isProbablyKey(std::ifstream& file);

std::array<char, 8> convertKey(const std::string keyString);

std::string getKeyString(std::array<char, 8> key, char separator = 0);

}

}

#endif // KEYHANDLING\_H

Файл Constants.h

#ifndef CONSTANTS\_H

#define CONSTANTS\_H

#include <array>

namespace Constants

{

const std::array<int, 64> IP =

{

58, 50, 42, 34, 26, 18, 10, 2,

60, 52, 44, 36, 28, 20, 12, 4,

62, 54, 46, 38, 30, 22, 14, 6,

64, 56, 48, 40, 32, 24, 16, 8,

57, 49, 41, 33, 25, 17, 9, 1,

59, 51, 43, 35, 27, 19, 11, 3,

61, 53, 45, 37, 29, 21, 13, 5,

63, 55, 47, 39, 31, 23, 15, 7

};

const std::array<int, 64> IP\_reverse =

{

40, 8, 48, 16, 56, 24, 64, 32,

39, 7, 47, 15, 55, 23, 63, 31,

38, 6, 46, 14, 54, 22, 62, 30,

37, 5, 45, 13, 53, 21, 61, 29,

36, 4, 44, 12, 52, 20, 60, 28,

35, 3, 43, 11, 51, 19, 59, 27,

34, 2, 42, 10, 50, 18, 58, 26,

33, 1, 41, 9, 49, 17, 57, 25

};

const int E[48] =

{

32, 1, 2, 3, 4, 5,

4, 5, 6, 7, 8, 9,

8, 9, 10, 11, 12, 13,

12, 13, 14, 15, 16, 17,

16, 17, 18, 19, 20, 21,

20, 21, 22, 23, 24, 25,

24, 25, 26, 27, 28, 29,

28, 29, 30, 31, 32, 1

};

const int S[8][4][16] =

{

{

{ 14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7 },

{ 0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8 },

{ 4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0 },

{ 15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13 }

}, // 1

{

{ 15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10 },

{ 3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5 },

{ 0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15 },

{ 13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9 }

}, // 2

{

{ 10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4, 2, 8 },

{ 13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1 },

{ 13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7 },

{ 1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12 }

}, // 3

{

{ 7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15 },

{ 13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 9 },

{ 10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4 },

{ 3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14 }

}, // 4

{

{ 2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9 },

{ 14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6 },

{ 4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14 },

{ 11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3 }

}, // 5

{

{ 12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5, 11 },

{ 10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8 },

{ 9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6 },

{ 4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8, 13 }

}, // 6

{

{ 4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1 },

{ 13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6 },

{ 1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2 },

{ 6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12 }

}, // 7

{

{ 13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7 },

{ 1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14, 9, 2 },

{ 7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8 },

{ 2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11 }

} // 8

};

const int P[32] =

{

16, 7, 20, 21,

29, 12, 28, 17,

1, 15, 23, 26,

5, 18, 31, 10,

2, 8, 24, 14,

32, 27, 3, 9,

19, 13, 30, 6,

22, 11, 4, 25

};

const int G[56] =

{

57, 49, 41, 33, 25, 17, 9,

1, 58, 50, 42, 34, 26, 18,

10, 2, 59, 51, 43, 35, 27,

19, 11, 3, 60, 52, 44, 36,

63, 55, 47, 39, 31, 23, 15,

7, 62, 54, 46, 38, 30, 22,

14, 6, 61, 53, 45, 37, 29,

21, 13, 5, 28, 20, 12, 4

};

const int shift[16] =

{ 1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1 };

const int H[48] =

{

14, 17, 11, 24, 1, 5,

3, 28, 15, 6, 21, 10,

23, 19, 12, 4, 26, 8,

16, 7, 27, 20, 13, 2,

41, 52, 31, 37, 47, 55,

30, 40, 51, 45, 33, 48,

44, 49, 39, 56, 34, 53,

46, 42, 50, 36, 29, 32

};

}

#endif // CONSTANTS\_H

Файл main.cpp

#include "DES.h"

#include "InputHandler.h"

#include "FileManager.h"

#include "KeyHandling.h"

#include <iostream>

int main(int argc, char\* argv[])

{

namespace KH = KeyHandling;

InputHandler ih;

if (!ih.process(argc, argv))

{

ih.printErrorMessage();

return 0;

}

DES des(DES::CFB);

switch(ih.getCommand())

{

case InputHandler::Command::KEYGEN:

{

try {

KeyPair pair = des.getKeyPair();

std::cout << ih.getKeyPath() << '\n';

KH::exportKey(ih.getKeyPath(), pair);

}

catch (std::exception& exc)

{

std::cerr << "Exception occured (" << exc.what() << ")\n";

}

break;

}

case InputHandler::Command::ENCRYPT:

{

if (!ih.isKeySet())

{

try

{

KeyPair pair = des.getKeyPair();

KH::exportKey(ih.getKeyPath(), pair);

FileManager fmD(ih.getSource(), ih.getDestination());

auto data = des.encrypt\_cfb(fmD.getData());

fmD.setData(data);

}

catch (std::exception& exc)

{

std::cerr << "Exception occured (" << exc.what() << ")\n";

}

} else

{

try

{

KeyPair pair = KH::importKey(ih.getKeyPath());

des.setKeyPair(pair);

FileManager fmD(ih.getSource(), ih.getDestination());

auto data = des.encrypt\_cfb(fmD.getData());

fmD.setData(data);

}

catch (std::exception& exc)

{

std::cerr << "Exception occured (" << exc.what() << ")\n";

}

}

break;

}

case InputHandler::Command::DECRYPT:

{

try

{

KeyPair pair = KH::importKey(ih.getKeyPath());

des.setKeyPair(pair);

FileManager fmD(ih.getSource(), ih.getDestination());

auto data = des.decrypt\_cfb(fmD.getData());

fmD.removeZeros(data);

fmD.setData(data);

}

catch (std::exception& exc)

{

std::cerr << "Exception occured (" << exc.what() << ")\n";

}

break;

}

default:

break;

}

return 0;

}

Файл DES.cpp

#include "DES.h"

#include "DES\_functions.h"

#include "Constants.h"

#include <iostream>

#include <iomanip>

#include <sstream> // std::basic\_stringstream

std::array<char, 7> DES::genKey()

{

namespace DF = DES\_functions;

std::array<char, 7> key = DF::genKey();

m\_key = DF::expandKey(key);

return key;

}

std::array<char, 8> DES::getKey()

{

if (m\_key.size() == 0)

throw DESException("Key is not generated");

return m\_key;

}

void DES::setKey(std::array<char, 8> key)

{

m\_key = key;

}

std::vector<char> DES::encrypt(std::vector<char> data)

{

size\_t dSize = data.size();

if (data.size() == 0)

throw DESException("Data is empty");

namespace DF = DES\_functions;

std::vector<char> dataEnc; //= data;

dataEnc = DF::encrypt(data, m\_key);

return dataEnc; //dataEnc;

}

std::vector<char> DES::encrypt\_cfb(std::vector<char> data)

{

namespace DF = DES\_functions;

size\_t dSize = data.size();

if (dSize == 0)

return std::vector<char>();

std::vector<char> dataEnc;

std::array<char, 8> T;

size\_t boundary;

if (dSize % 8 == 0)

boundary = dSize / 8;

else

boundary = dSize / 8 + 1;

T = m\_iv;

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

{

// DES -------

T = DF::permutation(T, Constants::IP);

T = DF::encryption(T, m\_key);

T = DF::permutation(T, Constants::IP\_reverse);

// -----------

// C\_i = M\_i xor Z\_i, Z\_i = DES(C\_i-1), C\_0 = IV

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

{

if (i \* 8 + j < dSize)

T[j] = T[j] ^ data[i \* 8 + j];

else

T[j] = T[j] ^ 0u;

dataEnc.push\_back(T[j]);

}

}

return dataEnc;

}

std::vector<char> DES::decrypt\_cfb(std::vector<char> data)

{

size\_t dSize = data.size();

if (dSize == 0 || dSize % 8 != 0)

return std::vector<char>();

std::vector<char> dataDec;

std::array<char, 8> T;

namespace DF = DES\_functions;

T = m\_iv;

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

{

// DES -----------

T = DF::permutation(T, Constants::IP);

T = DF::encryption(T, m\_key); // GENIOUS !!!

//T = DF::decryption(T, m\_key);

T = DF::permutation(T, Constants::IP\_reverse);

// ---------------

// M\_i = C\_i xor Z\_i, Z\_i = DES(C\_i-1), C\_0 = IV

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

{

dataDec.push\_back(T[j] ^ data[i \* 8 + j]);

T[j] = data[i \* 8 + j];

}

}

return dataDec;

}

std::vector<char> DES::decrypt(std::vector<char> data) //

{

size\_t dSize = data.size();

if (dSize == 0)

throw DESException("Data is empty");

if (dSize % 8 != 0)

throw DESException("Data size is not multiple of 8");

namespace DF = DES\_functions;

//std::array<char, 8> T{0};

std::vector<char> dataDec;

dataDec = DF::decrypt(data, m\_key);

return dataDec;

}

std::array<char, 7> DES::extractKey(std::array<char, 8> eKey)

{

namespace DF = DES\_functions;

return DF::extractKey(eKey);

}

std::string DES::getKeyString(char separator)

{

std::stringstream ss;

std::ios init(NULL);

init.copyfmt(ss);

ss << std::hex << std::uppercase;

ss << std::setfill('0');

for (auto el : m\_key)

{

ss << std::setw(2) << static\_cast<unsigned int>

(static\_cast<unsigned char>(el));

if (separator != 0)

ss << separator;

}

ss.copyfmt(init);

return ss.str();

}

std::array<char, 8> DES::genIV()

{

namespace DF = DES\_functions;

std::array<char, 8> iv{};

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

{

iv[i] |= DF::getRandomNumber(0x0, 0xFF);

}

m\_iv = iv;

return iv;

}

KeyPair DES::getKeyPair()

{

if (m\_key.size() == 0)

genKey();

if (m\_iv.size() == 0)

genIV();

return KeyPair(m\_key, m\_iv);

}

DES& DES::setKeyPair(KeyPair& pair)

{

m\_key = pair.get\_key();

m\_iv = pair.get\_iv();

return \*this;

}

Файл DES\_functions.cpp

#include <array>

#include <iostream>

#include <random> // C++11, std::random\_device, std::mt19937

#include "Constants.h"

#include "DES\_functions.h"

namespace DES\_functions

{

std::ostream& printBinary(std::ostream& out, char letter)

{

for (int i = 7; i >= 0; --i)

out << static\_cast<int>((letter >> i) & 1);

out << ' ';

return out;

}

std::array<char, 8> initialPermutation(std::array<char, 8> T)

{

using namespace Constants;

std::array<char, 8> T0;

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

{

T0[i] = 0;

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

{

int byte = (IP[i \* 8 + j] - 1) / 8;

int bit = (IP[i \* 8 + j] - 1) - (byte \* 8);

T0[i] |= ((T[byte] << bit) & 0x80) >> j;

}

}

return T0;

}

std::array<char, 8> permutation(std::array<char, 8> T, std::array<int, 64> IP)

{

std::array<char, 8> TP;

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

{

TP[i] = 0;

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

{

int byte = (IP[i \* 8 + j] - 1) / 8;

int bit = (IP[i \* 8 + j] - 1) - (byte \* 8);

TP[i] |= ((T[byte] << bit) & 0x80) >> j;

}

}

return TP;

}

std::array<char, 4> getL(std::array<char, 8> T)

{

std::array<char, 4> L;

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

L[i] = T[i];

return L;

}

std::array<char, 4> getR(std::array<char, 8> T)

{

std::array<char, 4> R;

for (int i = 4, j = 0; i < 8; ++i, ++j)

R[j] = T[i];

return R;

}

std::array<char, 6> expand(std::array<char, 4> RorL)

{

using Constants::E;

std::array<char, 6> RorLE;

for (int i = 0; i < 6; ++i)

{

RorLE[i] = 0;

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

{

int byte = (E[i \* 8 + j] - 1) / 8;

int bit = (E[i \* 8 + j] - 1) - (byte \* 8);

RorLE[i] |= ((RorL[byte] << bit) & 0x80) >> j;

}

}

return RorLE;

}

std::array<char, 6> xor48(std::array<char, 6> RorLE, std::array<char, 6> K)

{

std::array<char, 6> RorLEK;

for (int i = 0; i < 6; ++i)

RorLEK[i] = RorLE[i] ^ K[i];

return RorLEK;

}

std::array<char, 8> getB(std::array<char, 6> RorLEK)

{

std::array<char, 8> B;

int ind = 0, bitShift = 8;

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

{

B[i] = 0;

for (int j = 5; j >= 0; --j)

{

B[i] |= ((RorLEK[ind] >> --bitShift) & 0x1) << j;

if (bitShift == 0)

{

++ind;

bitShift = 8;

}

}

}

return B;

}

std::array<char, 4> shrink(std::array<char, 8> B)

{

std::array<char, 4> BS{};

using Constants::S;

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

{

for (int j = 0; j < 2; ++j)

{

int ind = i \* 2 + j;

int row = ( ((B[ind] >> 4) & 0x2) | (B[ind] & 0x1) ); // Строка

int column = ((B[ind] >> 1) & 0xF); // столбец

BS[i] |= S[ind][row][column] << (4 - 4 \* j);

}

}

return BS;

}

std::array<char, 4> transposition(std::array<char, 4> BS)

{

std::array<char, 4> BST;

using Constants::P;

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

{

BST[i] = 0;

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

{

int byte = (P[i \* 8 + j] - 1) / 8;

int bit = (P[i \* 8 + j] - 1) - (byte \* 8);

BST[i] |= ((BS[byte] << bit) & 0x80) >> j;

}

}

return BST;

}

std::array<char, 4> f(std::array<char, 4> RorL, std::array<char, 6> K)

{

std::array<char, 6> RorLE = expand(RorL);

std::array<char, 6> RorLEK = xor48(RorLE, K);

std::array<char, 8> B = getB(RorLEK);

std::array<char, 4> BS = shrink(B);

std::array<char, 4> BST = transposition(BS);

return BST;

}

std::array<char, 8> combineLR(std::array<char, 4> L, std::array<char, 4> R)

{

std::array<char, 8> C;

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

{

C[i] = L[i];

C[4 + i] = R[i];

}

return C;

}

std::array<char, 7> prepareKey(std::array<char, 8> key)

{

using Constants::G;

std::array<char, 7> CD{};

for (int i = 0; i < 7; ++i)

{

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

{

int byte = (G[i \* 8 + j] - 1) / 8;

int bit = (G[i \* 8 + j] - 1) % 8; //- (byte \* 8);

CD[i] |= ((key[byte] << bit) & 0x80) >> j;

}

}

return CD;

}

std::array<char, 7> shiftCD(std::array<char, 7> CD, int shift)

{

std::array<char, 7> CDS{};

int counter = 0;

int CDSbyte, CDSbit, CDbyte, CDbit;

for (int i = 0; i < 28; ++i)

{

CDSbyte = i / 8;

CDSbit = i % 8;

if (i + shift < 28)

{

CDbyte = (i + shift) / 8;

CDbit = (i + shift) % 8;

} else

{

CDbyte = (i + shift - 28) / 8;

CDbit = (i + shift - 28) % 8;

}

CDS[CDSbyte] |= ((CD[CDbyte] << CDbit) & 0x80) >> CDSbit;

}

for (int i = 28; i < 56; ++i)

{

CDSbyte = i / 8;

CDSbit = i % 8;

if (i + shift < 56)

{

CDbyte = (i + shift) / 8;

CDbit = (i + shift) % 8;

} else

{

CDbyte = (i + shift - 28) / 8;

CDbit = (i + shift - 28) % 8;

}

CDS[CDSbyte] |= ((CD[CDbyte] << CDbit) & 0x80) >> CDSbit;

}

return CDS;

}

std::array<char, 6> getK(std::array<char, 7> CDS)

{

using Constants::H;

std::array<char, 6> K{};

for (int i = 0; i < 6; ++i)

{

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

{

int byte = (H[i \* 8 + j] - 1) / 8;

int bit = (H[i \* 8 + j] - 1) % 8;

K[i] |= ((CDS[byte] << bit) & 0x80) >> j;

}

}

return K;

}

std::array<char, 8> encryption(std::array<char, 8> TP, std::array<char, 8> key)

{

using Constants::shift;

std::array<char, 7> CiDi = prepareKey(key);

std::array<char, 6> K;

std::array<char, 4> Lprev, L = getL(TP);

std::array<char, 4> R = getR(TP);

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

{

CiDi = shiftCD(CiDi, shift[i]);

K = getK(CiDi);

Lprev = L;

L = R;

R = f(R, K);

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

R[j] = Lprev[j] ^ R[j];

} // Проверить, что получается на выходе!

std::array<char, 8> C = combineLR(L, R);

return C;

}

std::array<char, 8> decryption(std::array<char, 8> revTP, std::array<char, 8> key)

{

using Constants::shift;

std::array<char, 7> CiDi = prepareKey(key);

std::array<char, 6> K;

std::array<char, 4> L = getL(revTP);

std::array<char, 4> Rprev, R = getR(revTP);

/\*

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

{

CiDi = shiftCD(CiDi, shift[i]);

} \*/ // Шаг логически не обоснован

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

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

{

K = getK(CiDi); // Сравнить генерируемые здесь ключи с ключами из encryption() !

CiDi = reverseShiftCD(CiDi, shift[i]); // Возможно, это неверно

//CiDi = shiftCD(CiDi, shift[i]); // Тогда, это верный вариант

// K = getK(CiDi); // Сравнить генерируемые здесь ключи с ключами из encryption() !

Rprev = R;

R = L;

L = f(L, K);

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

L[j] = Rprev[j] ^ L[j];

}

std::array<char, 8> C = combineLR(L, R);

return C;

}

std::array<char, 7> reverseShiftCD(std::array<char, 7> CD, int shift)

{

std::array<char, 7> CDS{};

int counter = 0;

int CDSbyte, CDSbit, CDbyte, CDbit;

for (int i = 0; i < 28; ++i)

{

CDSbyte = i / 8;

CDSbit = i % 8;

if (i - shift >= 0)

{

CDbyte = (i - shift) / 8;

CDbit = (i - shift) % 8;

} else

{

CDbyte = (i - shift + 28) / 8;

CDbit = (i - shift + 28) % 8;

}

CDS[CDSbyte] |= ((CD[CDbyte] << CDbit) & 0x80) >> CDSbit;

}

for (int i = 28; i < 56; ++i)

{

CDSbyte = i / 8;

CDSbit = i % 8;

if (i - shift >= 28)

{

CDbyte = (i - shift) / 8;

CDbit = (i - shift) % 8;

} else

{

CDbyte = (i - shift + 28) / 8;

CDbit = (i - shift + 28) % 8;

}

CDS[CDSbyte] |= ((CD[CDbyte] << CDbit) & 0x80) >> CDSbit;

}

return CDS;

}

std::vector<char> encrypt(std::vector<char>& data, std::array<char, 8> key)

{

size\_t dSize = data.size();

if (dSize == 0)

return std::vector<char>();

std::vector<char> dataEnc;

std::array<char, 8> T;

size\_t boundary;

if (dSize % 8 == 0)

boundary = dSize / 8;

else

boundary = dSize / 8 + 1;

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

{

T = {};

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

{

if (i \* 8 + j < dSize)

T[j] = data[i \* 8 + j];

else

T[j] = 0;

}

// DES-ECB encryption

T = permutation(T, Constants::IP);

T = encryption(T, key);

T = permutation(T, Constants::IP\_reverse);

/////////////////

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

dataEnc.push\_back(T[j]);

}

return dataEnc;

}

std::vector<char> decrypt(std::vector<char>& data, std::array<char, 8> key)

{

size\_t dSize = data.size();

if (dSize == 0 || dSize % 8 != 0)

return std::vector<char>();

std::vector<char> dataDec;

std::array<char, 8> T;

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

{

T = {};

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

{

//if (i \* 8 + j < dSize)

T[j] = data[i \* 8 + j];

//else

// T[j] = 0;

}

// DES-ECB decryption

//T = permutation(T, Constants::IP\_reverse);

T = permutation(T, Constants::IP);

T = decryption(T, key);

//T = permutation(T, Constants::IP);

T = permutation(T, Constants::IP\_reverse);

/////////////////

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

dataDec.push\_back(T[j]);

}

return dataDec;

}

int getRandomNumber(int min, int max)

{ // Вихрь Мерсенна

std::random\_device rd;

std::mt19937 getRand(rd());

double fraction = 1.0 / (static\_cast<double>(getRand.max()) + 1.0);

return static\_cast<int>(getRand() \* fraction \* (max - min + 1) + min);

}

std::array<char, 7> genKey()

{

std::array<char, 7> key{};

for (int i = 0; i < 7; ++i)

{

key[i] |= getRandomNumber(0x0, 0xFF);

}

return key;

}

std::array<char, 8> expandKey(std::array<char, 7> key)

{

std::array<char, 8> eKey{};

int keyByte = 0;

int keyBit = 0;

int ones = 0;

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

{

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

{

if (keyBit == 8)

{

++keyByte;

keyBit = 0;

}

if (j < 7)

{

unsigned char bit = (key[keyByte] << keyBit++) & 0x80;

if (bit != 0)

{

++ones;

eKey[i] |= bit >> j;

}

} else

{

if (ones % 2 == 0)

eKey[i] |= 0x1;

ones = 0;

}

}

}

return eKey;

}

std::array<char, 7> extractKey(std::array<char, 8> eKey)

{

std::array<char, 7> key{};

int eKeyByte = 0;

int eKeyBit = 0;

for (int i = 0; i < 7; ++i)

{

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

{

if (eKeyBit == 7)

{

++eKeyByte;

eKeyBit = 0;

}

key[i] |= (((eKey[eKeyByte] << (eKeyBit++)) & 0x80) >> j);

}

}

return key;

}

} // DES\_functions

Файл InputHandler.cpp

#include "InputHandler.h"

#include <cstring>

#include <iostream>

#include <limits> // std::numeric\_limits<>::max()

#include <iomanip> // std::setfill, std::setw(2), std::uppercase

bool InputHandler::isArgKey(char\* str)

{

char key[] = "key=";

if (strncmp(str, key, sizeof(key) - 1) == 0)

return true;

else

return false;

}

InputHandler::Status InputHandler::checkArgs(int argc, char\* argv[])

{

if (argc == 1)

return Status::NONE;

if (argc > 5)

return Status::INVALID;

if (strcmp(argv[1], "help") == 0)

m\_command = Command::HELP;

if (strcmp(argv[1], "keygen") == 0)

m\_command = Command::KEYGEN;

else if (strcmp(argv[1], "encrypt") == 0)

m\_command = Command::ENCRYPT;

else if (strcmp(argv[1], "decrypt") == 0)

m\_command = Command::DECRYPT;

else

return Status::INVALID;

Status result {Status::NONE};

switch(m\_command)

{

case Command::KEYGEN:

{

if (argc == 2)

result = Status::KG;

else if (argc == 3)

result = Status::KGD;

else

result = Status::INVALID;

break;

}

case Command::ENCRYPT:

{

switch(argc)

{

case 3:

result = Status::S;

break;

case 4:

if (isArgKey(argv[argc - 1]))

{

result = Status::S;

m\_keyFlag = true;

} else

result = Status::SD;

break;

case 5:

if (isArgKey(argv[argc - 1]))

{

result = Status::SD;

m\_keyFlag = true;

} else

result = Status::INVALID;

break;

default:

result = Status::INVALID;

break;

}

break;

}

case Command::DECRYPT:

{

if (argc < 4 || !isArgKey(argv[argc - 1]))

{

result = Status::INVALID;

break;

}

m\_keyFlag = true;

switch(argc)

{

case 4:

result = Status::S;

break;

case 5:

result = Status::SD;

break;

default:

result = Status::INVALID;

break;

}

break;

}

default:

break;

}

return result;

}

std::string InputHandler::getKeyPath(char\* arg)

{

std::string path;

for (int i = 4; i < strlen(arg); ++i)

{

path += \*(arg + i);

}

return path;

}

bool InputHandler::process(int argc, char\* argv[])

{

using std::string, std::cout, std::cin;

Status result = checkArgs(argc, argv);

if (result == Status::INVALID || result == Status::NONE)

{

return false;

}

switch(m\_command)

{

case Command::KEYGEN:

{

switch(result)

{

case Status::KG:

m\_keyPathName = "key";

break;

case Status::KGD:

m\_destinationName = argv[2];

break;

default:

break;

}

return true;

}

case Command::ENCRYPT:

{

m\_sourceName = argv[2];

switch(result)

{

case Status::SD:

m\_destinationName = argv[3];

break;

case Status::S:

m\_destinationName = "encrypted";

break;

default:

break;

}

break;

}

case Command::DECRYPT:

{

m\_sourceName = argv[2];

switch(result)

{

case Status::S:

m\_destinationName = "decrypted";

break;

case Status::SD:

m\_destinationName = argv[3];

break;

default:

break;

}

break;

}

default:

break;

}

if (m\_keyFlag)

{

m\_keyPathName = getKeyPath(argv[argc - 1]);

}

return true;

}

void InputHandler::printErrorMessage()

{

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

char keygen[] = "./des-cfb keygen [<key\_file\_path>]\n" \

"\tIf key file path is not set,\n" \

"\tfile named \"key\" will be created at the same directory.\n";

char encrypt[] = "./des-cfb encrypt <source\_file\_path> [<destination\_file\_path>] \

[key=\"<key\_file\_path>\"]\n" \

"\tIf key option is not set, a key file will be generated automatically.\n" \

"\tIf destination file name is not set,\n" \

"\tfile named \"encrypted\" will be created at the same directory.\n";

char decrypt[] = "./des-cfb decrypt <encrypted\_file\_path> [<decrypted\_file\_path>] \

key=<key\_file\_path>\n" \

"\tIf decrypted file name is not set,\n" \

"\tfile named \"decrypted\" will be created at the same directory.\n";

char help[] = "./des-cfb help\n";

switch(m\_command)

{

case Command::KEYGEN:

std::cout << keygen << '\n';

break;

case Command::ENCRYPT:

std::cout << encrypt << '\n';

break;

case Command::DECRYPT:

std::cout << decrypt << '\n';

break;

default:

std::cout << keygen << '\n' << encrypt << '\n' << decrypt << '\n'

<< help << '\n';

break;

}

}

std::string InputHandler::getSource() const

{

return m\_sourceName;

}

std::string InputHandler::getDestination() const

{

return m\_destinationName;

}

std::string InputHandler::getKeyPath() const

{

return m\_keyPathName;

}

InputHandler::Command InputHandler::getCommand() const

{

return m\_command;

}

bool InputHandler::isKeySet()

{

return m\_keyFlag;

}

Файл FileManager.cpp:

#include "FileManager.h"

#include <array>

#include <iomanip>

FileManager::FileManager(std::string dest) try

{

m\_dest.exceptions(std::ofstream::failbit | std::ofstream::badbit);

m\_dest.open(dest, std::ios\_base::out | std::ios\_base::binary);

}

catch(std::ios\_base::failure& exc)

{

throw(FileManagerException("Error opening destination file"));

}

FileManager::FileManager(std::string src, std::string dest) try

{

m\_src.exceptions(std::ifstream::failbit | std::ifstream::badbit);

m\_dest.exceptions(std::ofstream::failbit | std::ofstream::badbit);

m\_src.open(src, std::ios\_base::in | std::ios\_base::binary);

m\_dest.open(dest, std::ios\_base::out | std::ios\_base::binary);

}

catch(std::ios\_base::failure& exc)

{

throw(FileManagerException("Error opening source and dest files"));

}

std::vector<char> FileManager::getData(size\_t byteSize)

{

try

{

if (!m\_src.is\_open())

throw std::ios\_base::failure("Source is not set");

size\_t size;

if (byteSize > 0)

{

size = byteSize;

}

else

{

m\_src.seekg(0, m\_src.end);

size = m\_src.tellg();

m\_src.seekg(0, m\_src.beg);

}

char\* rowData = new char[size];

m\_src.read(rowData, size);

return std::vector<char>(rowData, rowData + size);

}

catch(const std::ios\_base::failure& e)

{

throw(FileManagerException("Error getting data"));

}

}

void FileManager::removeZeros(std::vector<char>& data)

{

char ch = data.back();

while (ch == 0 && data.size() != 0)

{

data.pop\_back();

ch = data.back();

}

}

void FileManager::setData(std::vector<char> data)

{

try

{

m\_dest.write(data.data(), data.size());

}

catch(const std::ios\_base::failure& e)

{

throw(FileManagerException("Error setting data"));

}

}

void FileManager::setData(std::string data)

{

try

{

m\_dest.write(data.data(), data.size());

}

catch(const std::ios\_base::failure& e)

{

throw(FileManagerException("Error setting data"));

}

}

void FileManager::setData(char\* data, size\_t size)

{

try

{

m\_dest.write(data, size);

}

catch(const std::ios\_base::failure& e)

{

throw(FileManagerException("Error setting data"));

}

}

Файл KeyHandling.cpp:

#include "KeyHandling.h"

#include <sstream>

#include <iomanip>

namespace KH = KeyHandling;

namespace KHP = KH::\_\_private\_\_;

std::array<char, 8> KHP::convertKey(const std::string keyString)

{

if (keyString.size() != 16)

return std::array<char, 8>();

std::array<char, 8> key{};

unsigned char a;

int ind = 0;

int shift = 4;

for (const auto ch : keyString)

{

a = 0;

if (ch >= 'A' && ch <= 'F')

{

a = (ch - 'A') + 10;

} else

if (ch >= 'a' && ch <= 'f')

{

a = (ch - 'a') + 10;

} else

if (ch >= '0' && ch <= '9')

{

a = ch - '0';

}

key[ind] |= (a << shift);

if (shift == 0)

{

++ind;

shift = 4;

} else

shift = 0;

}

return key;

}

#define IMPORT\_KEY\_FUNC(PATH) \

std::ifstream file; \

file.exceptions(std::ifstream::failbit | std::ifstream::badbit); \

std::string keybuf; \

char ch; \

KeyPair key; \

try \

{ \

file.open(PATH, std::ios\_base::in); \

if (!KHP::isProbablyKey(file)) \

throw(KeyHandling::Exception("Key checking fail")); \

file.seekg(0, file.beg); \

do \

{ \

ch = file.get(); \

if (KHP::isCharNumber(ch)) \

keybuf.push\_back(ch); \

else if (ch == ',') \

{ \

key.set\_key(KHP::convertKey(keybuf)); \

keybuf.clear(); \

} \

else if (ch == '}') \

{ \

key.set\_iv(KHP::convertKey(keybuf)); \

break; \

} \

} while (!file.eof()); \

file.close(); \

} \

catch(std::ifstream::failure e) \

{ \

throw(KeyHandling::Exception("Key importing fail")); \

} \

return key;

KeyPair KH::importKey(const char\* path)

{

IMPORT\_KEY\_FUNC(path)

}

KeyPair KH::importKey(const std::string& path)

{

IMPORT\_KEY\_FUNC(path)

}

KeyPair KH::importKey(const std::string&& path)

{

IMPORT\_KEY\_FUNC(path)

}

std::string KHP::getKeyString(std::array<char, 8> key, char separator)

{

std::stringstream ss;

std::ios init(NULL);

init.copyfmt(ss);

ss << std::hex << std::uppercase;

ss << std::setfill('0');

for (auto el : key)

{

ss << std::setw(2) << static\_cast<unsigned int>

(static\_cast<unsigned char>(el));

if (separator != 0)

ss << separator;

}

ss.copyfmt(init);

return ss.str();

}

#define EXPORT\_KEY\_FUNC(PATH, KEY) \

std::ofstream file; \

file.exceptions(std::ofstream::failbit | std::ofstream::badbit); \

try \

{ \

file.open(PATH, std::ios\_base::out); \

file.put('{'); \

std::string buf = KHP::getKeyString(KEY.get\_key()); \

file.write(buf.c\_str(), buf.size()); \

file.write(", ", 2); \

buf = KHP::getKeyString(KEY.get\_iv()); \

file.write(buf.c\_str(), buf.size()); \

file.put('}'); \

file.close(); \

} \

catch(std::ifstream::failure e) \

{ \

throw(KeyHandling::Exception("Key exporting fail")); \

}

void KH::exportKey(const std::string& path, const KeyPair& key)

{

EXPORT\_KEY\_FUNC(path, key)

}

void KH::exportKey(const std::string& path, const KeyPair&& key)

{

EXPORT\_KEY\_FUNC(path, key)

}

bool KHP::isCharNumber(char ch)

{

if (ch >= '0' && ch <= '9' ||

ch >= 'a' && ch <= 'f' ||

ch >= 'A' && ch <= 'F')

return true;

else

return false;

}

bool KHP::isProbablyKey(std::ifstream& file)

{

if (!file.is\_open())

return false;

file.exceptions(std::ifstream::failbit | std::ifstream::badbit);

file.seekg(0, file.end);

size\_t fsize = file.tellg();

if (fsize > 256)

return false;

file.seekg(0, file.beg);

char ch;

int braceCnt = 0;

int commaCnt = 0;

int digitCnt = 0;

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

{

ch = file.get();

if (braceCnt == 0)

{

if (ch == '{')

++braceCnt;

}

else

{

if (isCharNumber(ch))

++digitCnt;

else

if (ch == ',' && digitCnt == 16)

{

++commaCnt;

digitCnt = 0;

} else

if (ch == '}' && digitCnt == 16)

{

--braceCnt;

break;

}

}

}

if (commaCnt == 1 && braceCnt == 0)

return true;

return false;

}