МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования



НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ им. Р.Е.АЛЕКСЕЕВА

ОТЧЁТ

по лабораторной работе №1

по дисциплине

Защита информации

Вариант 5

РУКОВОДИТЕЛЬ:

Капранов С.Н.

СТУДЕНТ:

Докукин Д.В.

ГРУППА:

20-ПО

Нижний Новгород 2023

Задание

Лабораторная работа 1 – Шифры замены

Задание №1.5

Реализовать алгоритм шифрования данных «Шифр Трансфельда»\*

*\*Примечание: по всей видимости, в тексте задания ошибка и подразумевался шифр Гронсфельда, поскольку упоминания по поисковому запросу “Шифр Трансфельда” отсутствуют. Исходя из этого, в работе реализован шифр Гронсфельда.*

Введение

Текущее задание, как и все последующие, было реализовано мной на языке программирования C++. Хотя возможности языка с учётом доступных библиотек позволяют оперировать с русским языком, реализация такого взаимодействия накладывает дополнительные временные затраты, существенным образом не влияющие на картину понимания темы. По этой причине, данный шифр в моей реализации оперирует исключительно английским алфавитом в кодировке ASCII.

Описание алгоритма

Этот шифр сложной замены, называемый шифром Гронсфельда, представляет собой модификацию шифра Цезаря числовым ключом. Для этого под буквами исходного сообщения записывают цифры числового ключа. Если ключ короче сообщения, то его запись циклически повторяют. Шифртекст получают примерно, как в шифре Цезаря, но отсчитывают по алфавиту не третью букву (как это делается в шифре Цезаря), а выбирают ту букву, которая смещена по алфавиту на соответствующую цифру ключа. Например, применяя в качестве ключа группу из четырех начальных цифр числа e (основания натуральных логарифмов), а именно 2718, получаем для исходного сообщения «ВОСТОЧНЫЙ ЭКСПРЕСС» следующий шифртекст:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Сообщение |  | В | О | С | Т | О | Ч | Н | Ы | Й |  | Э | К | С | П | Р | Е | С | С |
| Ключ |  | 2 | 7 | 1 | 8 | 2 | 7 | 1 | 8 | 2 |  | 7 | 1 | 8 | 2 | 7 | 1 | 8 | 2 |
| Шифртекст |  | Д | Х | Т | Ь | Р | Ю | О | Г | Л |  | Д | Л | Щ | С | Ч | Ж | Щ | У |

Чтобы зашифровать первую букву сообщения В, используя первую цифру ключа 2 , нужно отсчитать вторую по порядку букву от В в алфавите:

|  |  |  |
| --- | --- | --- |
| В | Г | Д |
|  | 1 | 2 |

получается первая буква шифр-текста Д.

Демонстрация работы программы

Для демонстрации, согласно требованиям, будут зашифрованы следующие сообщения:

1. Nizhny Novgorod State Technical University
2. Dokukin Dmitry Vladimirovich

В качестве ключа использована последовательность:

12345678

Поскольку шифр в своей оригинальной форме преобразует только буквы алфавита, прочие знаки пунктуации, пробелы и переходы на новую строку не изменяются.

Зашифрованное сообщение получается в формате текстового документа.

Синтакс программы выглядит следующим образом:

./gronsfeld encrypt|decrypt <source\_file\_path> [<destination\_file\_path>] [-k=\"<key\_file\_path>\"]

Первым аргументом идет имя программы.

Далее, тип операции.

Третьим имя(путь) шифруемого/расшифровываемого файла.

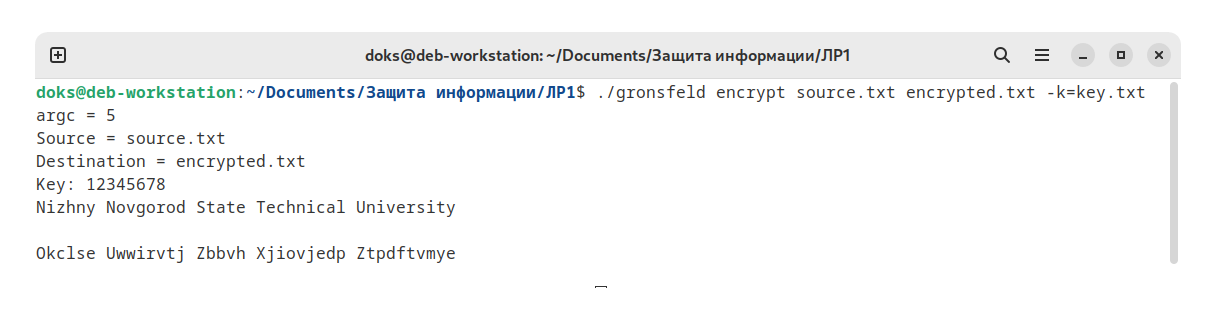
Четвёртый служит именем файла, куда будет записан шифртекст.

Пятый аргумент указывает на файл с ключом.

Пример 1

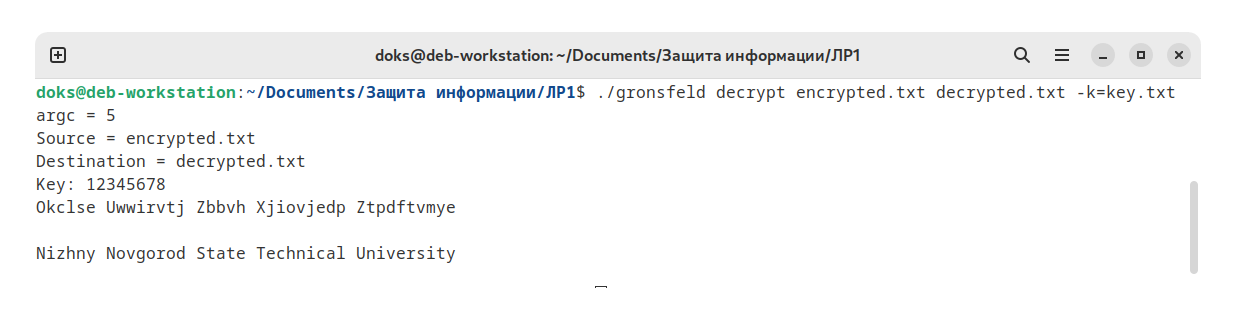
Шифрование сообщения: Nizhny Novgorod State Technical University

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



Программа показывает дополнительную информацию о входных и выходных данных. В данном случае, можно наблюдать зашифрованную форму сообщения.

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

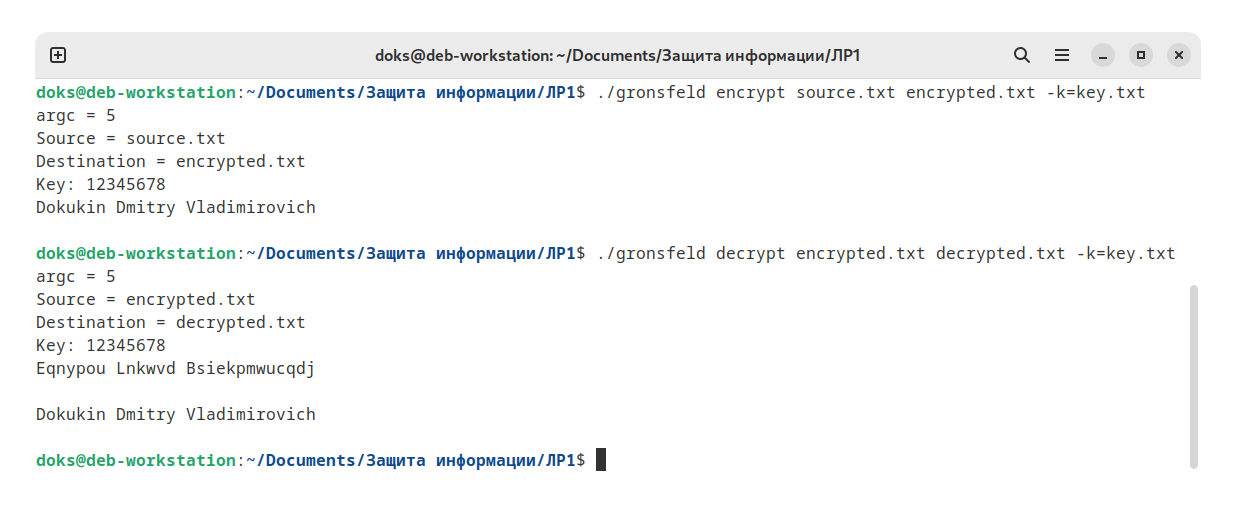


Можно наблюдать, что расшифровка прошла успешно.

Пример 2

Шифрование и расшифровка сообщения: Dokukin Dmitry Vladimirovich

Обе операции показаны на одном скриншоте.



Как можно видеть, процессы прошли успешно.

Программа

В программе представлены два класса:

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

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

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

Листинг

Файл Gronsfeld.h

#define GRONSFELD\_H

#include <vector>

#include <string>

#include <fstream>

class Gronsfeld

{

private:

std::vector<unsigned int> m\_key;

public:

Gronsfeld(std::vector<unsigned int> key) : m\_key(key) {}

void encrypt(char\* msg, size\_t msgSize);

void decrypt(char\* msg, size\_t msgSize);

};

#endif // GRONSFELD\_H

Файл InputHandler.h

#ifndef INPUTHANDLER\_H

#define INPUTHANDLER\_H

#include <string>

#include <fstream>

#include <vector>

class InputHandler

{

public:

enum Status

{

NONE, // None

S, // Only the Source was entered

SD, // Source and Destination were entered

SK, // Source and Key were entered

SDK, // Source, Destination and key were entered

INVALID // Invalid input

};

enum Command

{

ENCRYPT,

DECRYPT

};

private:

std::string m\_sourceName;

std::string m\_destinationName;

std::vector<unsigned int> m\_key;

Status m\_status;

Command m\_command;

bool isKey(char\* str);

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

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

std::vector<unsigned int> extractKey(const std::string keyPath);

std::vector<unsigned int> convertKey(const std::string keyString);

public:

InputHandler(): m\_status(Status::NONE) {}

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

void printMessage();

std::string getSource() const;

std::string getDestination() const;

std::vector<unsigned int> getKey() const;

Command getCommand() const;

~InputHandler() {}

};

#endif // INPUTHANDLER\_H

Файл main.cpp

#include <iostream>

#include "InputHandler.h"

#include "Gronsfeld.h"

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

{

InputHandler ih;

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

{

ih.printMessage();

return 0;

}

std::ifstream source;

std::ofstream destination;

source.open(ih.getSource(), std::ios\_base::in);

if (!source.is\_open())

{

std::cout << "Error opening source\n";

return 0;

}

destination.open(ih.getDestination(), std::ios\_base::out);

if (!destination.is\_open())

{

std::cout << "Error creating dest\n";

return 0;

}

source.seekg(0, source.end);

size\_t sSize = source.tellg();

source.seekg(0, source.beg);

char\* buff = new char[sSize + 1];

source.read(buff, sSize);

source.close();

buff[sSize] = '\0';

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

Gronsfeld gf(ih.getKey());

if (ih.getCommand() == InputHandler::Command::ENCRYPT)

gf.encrypt(buff, sSize);

else if (ih.getCommand() == InputHandler::Command::DECRYPT)

gf.decrypt(buff, sSize);

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

destination.write(buff, sSize);

destination.close();

delete[] buff;

return 0;

}

Файл Gronsfeld.cpp

#include "Gronsfeld.h"

void Gronsfeld::encrypt(char\* msg, size\_t msgSize)

{

char capitalFirst = 'A';

char capitalLast = 'Z';

char lowercaseFirst = 'a';

char lowercaseLast = 'z';

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

{

if (msg[i] >= capitalFirst && msg[i] <= capitalLast)

{

if (msg[i] + m\_key[j] > capitalLast)

msg[i] = msg[i] + m\_key[j] - capitalLast

+ capitalFirst - 1;

else

msg[i] = msg[i] + m\_key[j];

} else

if (msg[i] >= lowercaseFirst && msg[i] <= lowercaseLast)

{

if (msg[i] + m\_key[j] > lowercaseLast)

msg[i] = msg[i] + m\_key[j] - lowercaseLast

+ lowercaseFirst - 1;

else

msg[i] = msg[i] + m\_key[j];

} else

continue;

if (++j == m\_key.size())

j = 0;

}

}

void Gronsfeld::decrypt(char\* msg, size\_t msgSize)

{

char capitalFirst = 'A';

char capitalLast = 'Z';

char lowercaseFirst = 'a';

char lowercaseLast = 'z';

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

{

if (msg[i] >= capitalFirst && msg[i] <= capitalLast)

{

if (msg[i] - m\_key[j] < capitalFirst)

msg[i] = msg[i] - capitalFirst + capitalLast - m\_key[j] + 1;

else

msg[i] = msg[i] - m\_key[j];

} else

if (msg[i] >= lowercaseFirst && msg[i] <= lowercaseLast)

{

if (msg[i] - m\_key[j] < lowercaseFirst)

msg[i] = msg[i] - lowercaseFirst + lowercaseLast - m\_key[j] + 1;

else

msg[i] = msg[i] - m\_key[j];

} else

continue;

if (++j == m\_key.size())

j = 0;

}

}

Файл InputHandler.cpp

#include "InputHandler.h"

#include <cstring>

#include <iostream>

bool InputHandler::isKey(char\* str)

{

if ( (\*(str) == '-')

&& (\*(str + 1) == 'k')

&& (\*(str + 2) == '='))

{

return true;

} else

return false;

}

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

{

if (argc < 3 || argc > 5)

return Status::INVALID;

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};

for (int i = 2; i < argc - 1; ++i)

{

if (isKey(argv[i]))

{

result = Status::INVALID;

break;

}

}

if (result != INVALID && isKey(argv[argc - 1]))

{

if (argc == 4)

result = Status::SK;

if (argc == 5)

result = Status::SDK;

}

return result;

}

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

{

std::string path;

for (int i = 3; i < strlen(arg); ++i)

{

path += \*(arg + i);

}

return path;

}

std::vector<unsigned int> InputHandler::extractKey(const std::string keyPath)

{

std::ifstream keyFile;

std::vector<unsigned int> key;

keyFile.open(keyPath, std::ios\_base::in);

if (!keyFile.is\_open())

{

std::cout << "Invalid key path\n";

return key;

}

char ch = keyFile.get();

while (ch != EOF)

{

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

{

key.push\_back(ch - '0');

ch = keyFile.get();

} else if (ch == '\n' || ch == ' ')

{

ch = keyFile.get();

} else

{

key.clear();

break;

}

}

keyFile.close();

return key;

}

std::vector<unsigned int> InputHandler::convertKey(const std::string keyString)

{

std::vector<unsigned int> key;

for (const auto ch : keyString)

{

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

return std::vector<unsigned int>();

key.push\_back(ch - '0');

}

return key;

}

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

{

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

Status result = checkArgs(argc, argv);

if (result == Status::INVALID)

{

return false;

}

if ((result == Status::SK) || (result == Status::SDK))

{

string keyFile = getKeyPath(argv[argc - 1]);

m\_key = extractKey(keyFile);

}

else

{

cout << "Enter key: ";

std::string keyString;

cin >> keyString;

m\_key = convertKey(keyString);

}

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

{

return false;

}

m\_sourceName = argv[2];

if (result == Status::SD || result == Status::SDK)

m\_destinationName = argv[3];

else

{

switch(m\_command)

{

case Command::ENCRYPT:

m\_destinationName = "encrypted\_" + m\_sourceName;

break;

case Command::DECRYPT:

m\_destinationName = "decrypted\_" + m\_sourceName;

break;

}

}

cout << "argc = " << argc << '\n';

cout << "Source = " << m\_sourceName << '\n';

cout << "Destination = " << m\_destinationName << '\n';

cout << "Key: ";

for (auto el : m\_key)

cout << el;

cout << '\n';

return true;

}

void InputHandler::printMessage()

{

char syntax[] = "Syntax:\n" \

"./gronsfeld encrypt|decrypt <source\_file\_path> [<destination\_file\_path>] [-k=\"<key\_file\_path>\"]\n";

std::cout << syntax;

}

std::string InputHandler::getSource() const

{

return m\_sourceName;

}

std::string InputHandler::getDestination() const

{

return m\_destinationName;

}

std::vector<unsigned int> InputHandler::getKey() const

{

return m\_key;

}

InputHandler::Command InputHandler::getCommand() const

{

return m\_command;

}