1. Министерство образования и науки Российской Федерации
2. Санкт-Петербургский Политехнический Университет Петра Великого
3. —
4. **Институт компьютерных наук и кибербезопасности**

**Курсовая работа**

1. «Дешифровка простого XOR»
2. по дисциплине «Структуры данных»
3. Выполнили
4. студенты гр. 5131001/30003 Шевчук Н.Е.
5. Мальцев М.А.

1. Преподаватель Семьянов П.В.
3. Санкт-Петербург

2024

**ОГЛАВЛЕНИЕ**

[Введение 3](#_Toc168643736)

[1. Цель 4](#_Toc168643737)

[2. Задачи 4](#_Toc168643738)

[3. Теоретическая справка 5](#_Toc168643739)

[3.1. Шифрование повторяющимся XOR 5](#_Toc168643740)

[3.2. Взлом повторяющегося ключа XOR 5](#_Toc168643741)

[3.2.1. Вычисление длины ключа 7](#_Toc168643742)

[3.2.2. Частотный анализ 7](#_Toc168643743)

[3.2.3. Итоговая расшифровка текста и проверка 8](#_Toc168643744)

[4. Ход работы 8](#_Toc168643745)

[4.1. Программа кодер 8](#_Toc168643746)

[4.2. Программа декодер 9](#_Toc168643747)

[5. Результаты 11](#_Toc168643748)

[6. Вывод 14](#_Toc168643749)

[7. Список источников 14](#_Toc168643750)

[8. Приложение 15](#_Toc168643751)

Введение

При изучении криптографии нельзя не заметить, что операция “исключающие или” (XOR) используется довольно часто, особенно в шифрах. XOR - это простая побитовая операция, которая позволяет криптографам создавать надежные системы шифрования и, следовательно, является фундаментальным компонентом практически всех современных шифров. Давайте углубимся в детали и посмотрим, что делает XOR таким важным.

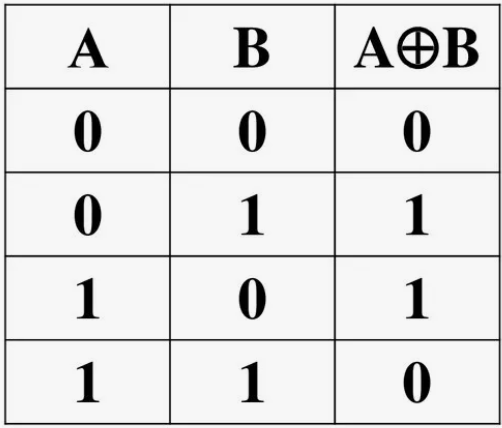
XOR, или “исключающее или”, работает с двоичными данными. Он возвращает значение true, если оба его входных значения противоположны (один false и один true), в противном случае он возвращает значение false (см. рис. 1). Существуют различные варианты обозначения XOR: ⊕, ^.

Рисунок 1. Таблица истинности операции XOR

Благодаря свойствам XOR один из входов может использоваться в качестве ключа для передачи данных на другой вход. Например, если A – одиночный бит ключа шифрования, XOR с битом данных из B «переключает» бит в другое состояние, если A – 1. Повторное применение побитовой операции XOR с ключом и зашифрованным сообщением расшифровывает его:

𝑎 = (𝑎 ⊕ 𝑏) ⊕ 𝑏

Отсюда вытекает основное применение простого XOR в шифровании данных:

*encrypted* = *message* ⊕ *key*

*message* = *encrypted* ⊕ *key*

На самом деле алгоритм простого XOR – не более чем помеха для криптоаналитика. Реальной защиты такой алгоритм не обеспечивает.

# Цель

Реализовать на языке C две программы – кодер и декодер. Кодер позволяет зашифровать английский текст, находящийся во входном файле *“source\_text.txt”*, с помощью ключа шифрования, введённого пользователем, и записать зашифрованный текст в выходной файл *“encrypted\_text.txt”.* Декодер в свою очередь получает на вход зашифрованный текст в файле *“encrypted\_text.txt”,* вычисляет длину исходного ключа и расшифровывает текст, записывая окончательный результат в файл *“decrypted.txt”.*

# Задачи

1. Изучить принцип работы операции простой XOR и возможность её применения в криптографии.
2. Рассмотреть шифрование длинных текстов с помощью повторяющегося ключа XOR.
3. Изучить алгоритм расшифровки простого XOR процедурой “подсчет совпадений”, описанной в книге Брюса Шнайера “Прикладная криптография”.
4. Реализовать программы “Кодер” и “Декодер” на языке C.
5. Протестировать написанные программы с различными текстами и ключами шифрования. Определить соотношение длин ключа и исходного текста при котором программа работает корректно.

# Теоретическая справка

### Шифрование повторяющимся XOR

Идея шифрования состоит в том, что выбирается какой-то секретный ключ, а затем повторяется до тех пор, пока не станет той же длины, что и открытый текст, который необходимо зашифровать, и для каждого символа строки (в том числе пробела) выполняется операция XOR с соответствующим символом в ключе. Повторяющийся ключ обрезается в конце, чтобы соответствовать длине открытого текста, из открытого текста вырезаются все символы, не соответствующие буквам английского алфавита. Итак, с ключом KEYXOR открытый текст рядом с повторяющимся ключом будет выглядеть так:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| M | E | E | T | M | E | A | T | M | I | D | N | I | G | H | T |
| K | E | Y | X | O | R | K | E | Y | X | O | R | K | E | Y | X |

### Взлом повторяющегося ключа XOR

В книге Брюса Шнайера изложен следующий алгоритм расшифровки:

1. Определим длину ключа с помощью процедуры, известной как *подсчет совпадений*. Применим операцию XOR к шифротексту, используя в качестве ключа сам шифротексту с различными смещениями, и подсчитаем совпадающие байты. Если величина смещения кратна длине ключа, то совпадет свыше 6% байтов. Если нет, то будут совпадать меньше чем 0,4% процента (считая, что обычный ASCII текст кодируется случайным ключом, для других типов открытых текстов числа будут другими). Это называется *показателем совпадений*. Минимальное смещение от одного значения, кратного длине ключа, к другому и есть длина ключа.
2. Сместим шифротекст на эту длину и проведем операцию XOR для смещенного и оригинального шифротекстов. Результатом операции будет удаление ключа и получение открытого текста, подвергнутого операции XOR с самим собой, смещенным на длину ключа. Так как в английском языке на один байт приходится 1.3 бита действительной информации, существующая значительная избыточность позволяет определить способ шифрования.

На практике реализация пункта 2 не приводила к расшифровке, поэтому был взят другой алгоритм. Рассмотрим более подробно полный алгоритм расшифровки текста:

1. Для каждого предположения о длине ключа сначала выполним циклический сдвиг шифротекста. Подсчитаем совпадения среди символов исходного и смещенного шифротекста. Для каждого из сдвигов вычислим процент (показатель) совпадений, равный отношению числу совпавших символов к общей длине шифротекста. Для сдвигов кратных длине ключа показатель совпадений должен превышать 6%. Таким образом, находя наибольший общий делитель всех сдвигов с большим показателем совпадений, вычислим исходную длину ключа *key\_length.*
2. Как только узнаем предполагаемую длину ключа, транспонируем исходный зашифрованный текст на фрагменты, где первый фрагмент состоит из первого байта каждого *key\_length* длинного блока текста, второй фрагмент состоит из второго байта каждого *key\_length* и т.д.
3. Для каждого из этих новых фрагментов выполним частотный анализ.
4. Соединим отдельные байты ключа, которые мы обнаружили на предыдущем шаге, вместе, чтобы получить вероятный ключ.
5. Расшифруем текст с помощью ключа и оценим его на читабельность.

### 3.2.1. Вычисление длины ключа

Процент совпадений между двумя случайными символами ASCII, представленными в виде байтов, должен быть больше, чем процент совпадений между двумя совершенно случайными (равномерно распределенными) байтами. Т.е. рассматривая различные циклические сдвиги шифротекста, соответствующие предполагаемой длине ключа, мы ищем какое из мнимых разбиений является не случайным, а обладает некоторой закономерностью. Сдвиги, кратные исходной длине ключа, будут обладать процентов совпадений выше 6%. С помощью отдельных функций найдем наибольший общий делить всех таких сдвигов – это и будет исходная длина ключа.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| a | b | c | d | e |
| a | f | c | k | j |
| Процент совпадений – 0,04 | | | | |

### 3.2.2. Частотный анализ

Зная, что каждый символ ключа использовался для шифрования соответствующего по порядку символа исходного текста, делим зашифрованный текст на группы, соответствующие положению в ключе (тоже пример с midnight добавить). Каждый из получившихся фрагментов зашифрован определенным символом. Применим частотный анализ по отдельности к каждому фрагменту, чтобы определить исходные символы ключа.

Сочетание “ETAOINSHRDLU” представляет собой упорядоченный набор 12 наиболее часто используемых в английском языке букв. Перебираем символы ASCII, соответствующие английским буквам, и производим операцию XOR с каждой группой в отдельности. Находим символ, при котором результатом операции XOR будут только буквы английского алфавита. Также оцениваем, какие из получившихся букв находятся в “ETAOINSHRDLU”. Данный символ ASCII и будет соответствующим по порядку символом ключа (картинка с примером).

### 3.2.3. Итоговая расшифровка текста и проверка

Зная исходный ключ, необходимо повторно реализовать посимвольную операцию XOR с повторяющимся ключом к зашифрованному тексту.

Очевидно, что в выбранном алгоритме неправильное определение длины ключа однозначно ведет к неверной расшифровке текста. Т.к. вычисление длины ключа основано на проценте совпадений, который сравнивается с константной величиной (6%) и вычисляется по исходному шифротексту, правильность определения во многом зависит от соотношений длины ключа и открытого текста. Поэтому необходимо проверить результаты работы программы на нескольких текстах различной длины, в каждом из которых рассмотреть также ключи различной длины.

# Ход работы

### Программа кодер

В основной функции *main()* пользователь вводит произвольный ключ шифрования как строку. Эта строка передается входным параметром в процедуру *encryption\_key\_xor()*, в которой осуществляется считывание открытого текста из входного файла, удаление из него всех символов, несоответствующих заглавным или строчным буквам английского алфавита, и его посимвольное кодирование операцией простой XOR повторяющимся ключом. Повторение ключа осуществляется с помощью перемещения указателя на начало строки. Кроме того, данная программа записывает введённый ключ в текстовый файл для последующего анализа правильности его нахождения.

### Программа декодер

В основной функции *main()* из файла считывается зашифрованный текст и вычисляется его длина. Шифротекст записывается в динамический массив *encrypted\_text.* Рассмотрим основные функции, отвечающие за реализацию шагов алгоритма.

* *determine\_key\_length –* функция вычисления длины ключа. На вход принимает указатель на массив с шифротекстом и его длину, на выход возвращает предполагаемую длину ключа. С помощью цикла for перебираются различные сдвиги. Для каждого из сдвигов еще одним циклом вычисляется количество совпадений между исходным и смещенным шифротекстом. Циклический сдвиг обеспечивается за счет взятия остатка от деления на длину шифротекста:

if (encrypted\_text[j] == encrypted\_text [(j + i) % size])

Вычисляется процент совпадений, сдвиг записывается в отдельный массив *valid\_key\_lengths,* если он больше 6%. С помощью вспомогательным функций *gcd* (нахождение наибольшего общего делителя двух чисел) и *gcd\_multiple* (нахождение наибольшего общего делителя нескольких чисел) находим исходную длину ключа как наибольший общий делитель сдвигов, процент совпадений которых выше 0,06. Память под созданный массив кратных сдвигов освобождается по завершению функции.

* *get\_key* – процедура, находящая исходный ключ. На вход принимает указатель на созданный под заполнение ключа массив *key,* указатель на массив исходного шифротекста, размер шифротекста и вычисленную длину ключа. Циклом for перебираем различные позиции в ключе. Для каждой из них создаем массив *block* – группа символов закодированных данной позицией ключа. Циклом перебираем различные символы английского алфавита и выполняем операцию XOR с одной буквой для всех символов группы, записывая результат в новый массив *x*. Проверяем, что результат операции XOR – только буквы английского алфавита, тогда можно рассматривать текущую букву как символ ключа. Кроме того, отдельным циклом проверяем, какое количество результатов операции XOR встречаются в сочетании "ETAOINSHRDLU”. Если их число больше, чем у предыдущих букв, то новая буква становится символом ключа, а максимальное количество “частых” букв в результате операции XOR обновляется. В конце функции память, выделенная под вспомогательные массивы, освобождается.
* *decrypt* – процедура, расшифровывающая текс по известному ключу. Посимвольно выполняет операцию XOR с шифротекстом и найденным повторяющимся ключом, записывая расшифрованный текст в новый файл. Повторение ключа осуществляется с помощью перемещения указателя на начало строки.
* *percent\_of\_key\_lenght –* процедура, проверяющая корректность расшифровки. На вход принимает найденный ключ и его вычисленную длину. Сравнивает их с исходным ключом, сохраненным в файле. Выводит на экран найденную длину ключа и количество совпадений символов исходного и вычисленного ключей.

# Результаты

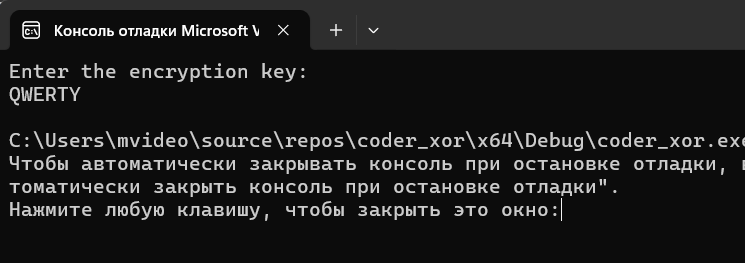
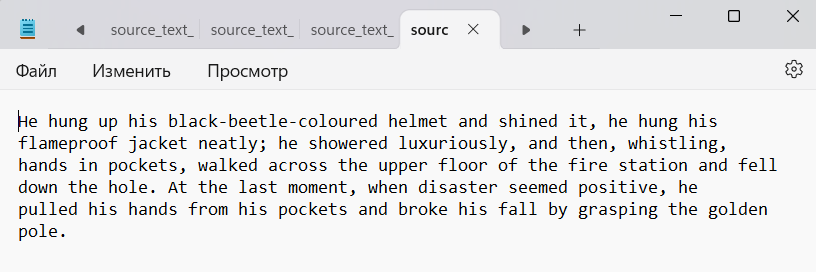
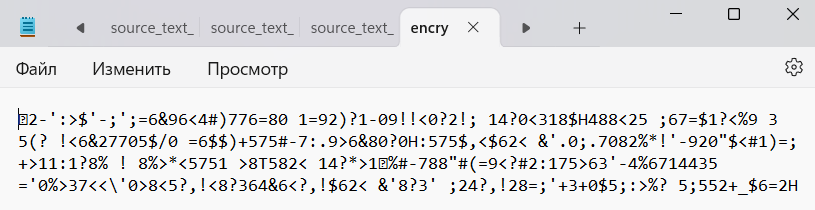
На рисунках ниже представлен результат работы программы:

Рисунок 4. Шифротекст, полученный программой “Кодер”

Рисунок 3. Исходный текст

Рисунок 2. Ввод ключа в программе “Кодер”

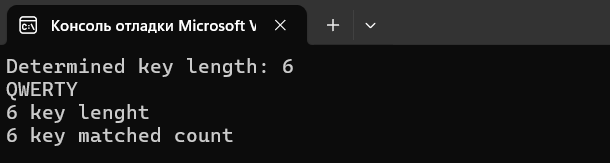
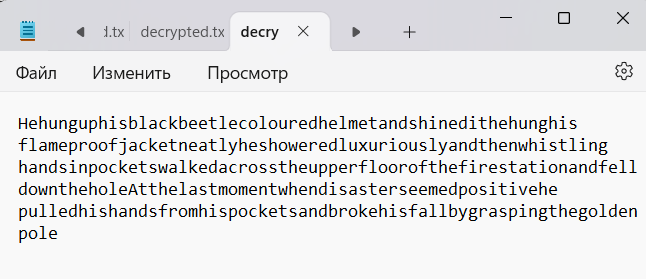


Рисунок 6. Расшифрованный программой “Декодер” текст

Рисунок 5. Результат программы “Декодер”

Как сказано выше, т.к. расшифровка в первую очередь зависит от правильности вычисления длины ключа, которая в свою очередь зависит от процента совпадений (различного для разных текстов), то результат работы программы будет зависеть от соотношения длины ключа к длине текста (будем рассматривать, какой процент составляет длина ключа от длины текста). Т.к. итоговый расшифрованный текст определяется с помощью посимвольной операции XOR, то процент совпадений букв расшифрованного и исходного текста будет равен проценту совпадения символов исходного и вычисленного программой ключа. На диаграммах ниже представлена зависимость правильности вычисления длины ключа и его символов от этого процентного соотношения длин для текстов в 100, 500 и 2000 символов (таблицы, по которым построен график представлены в приложении):

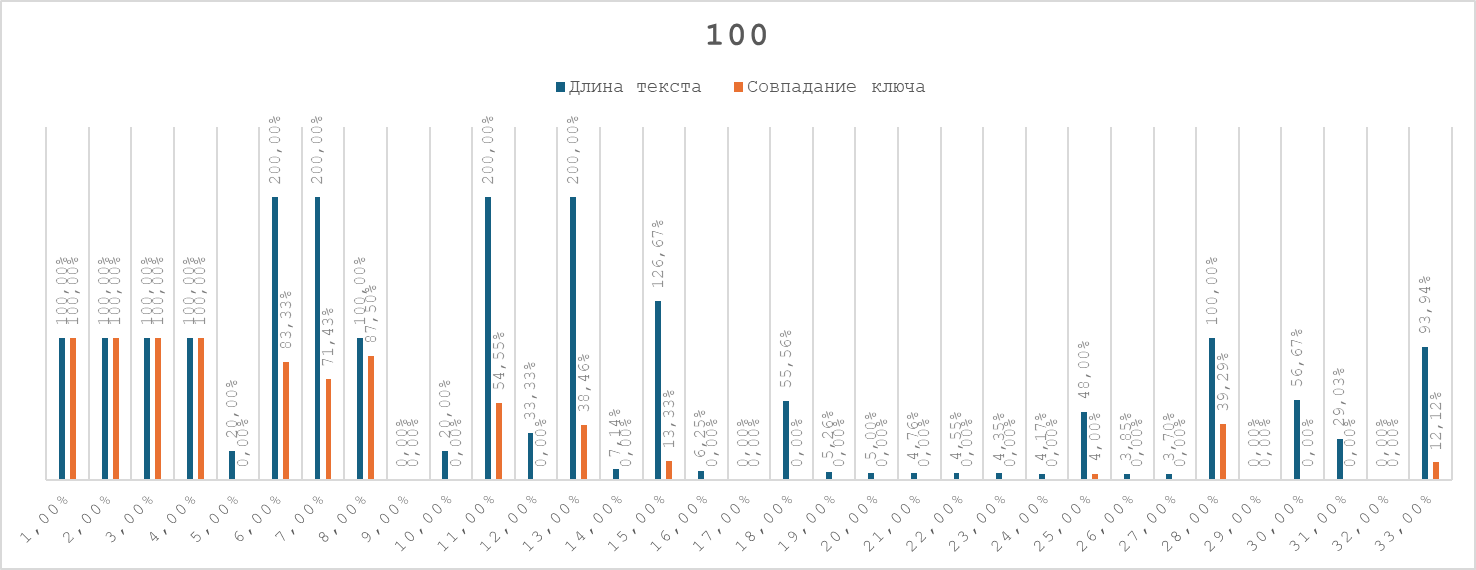
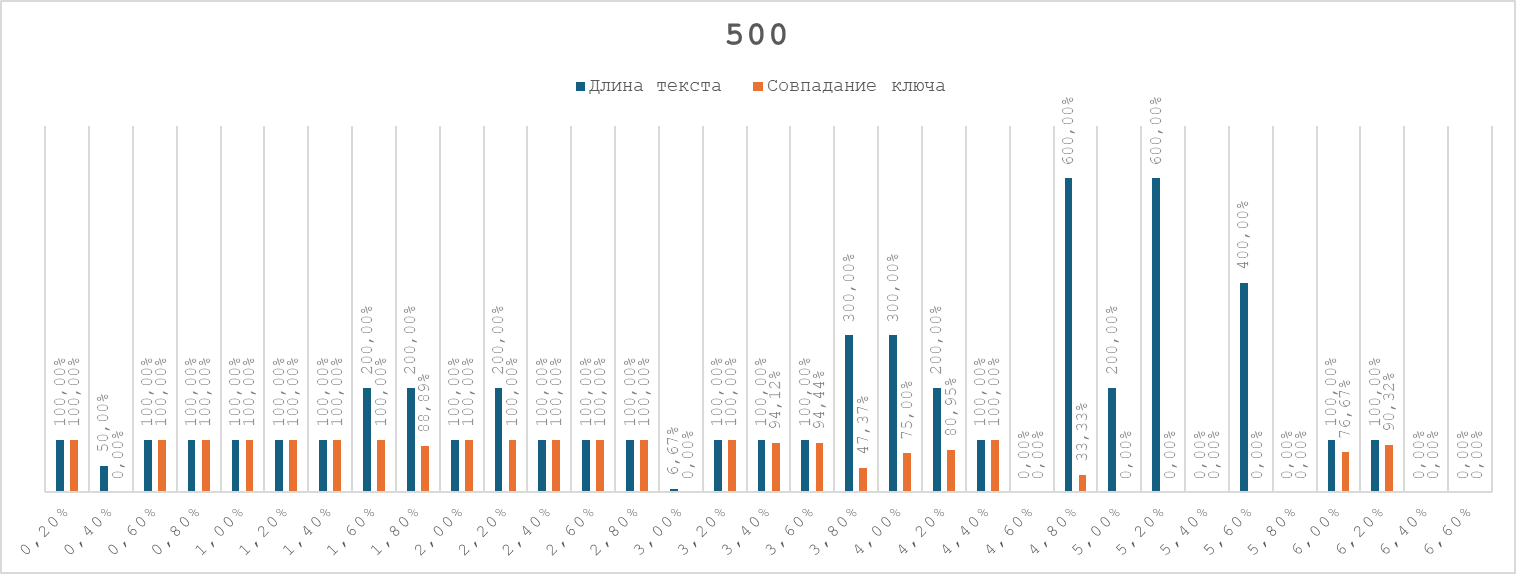
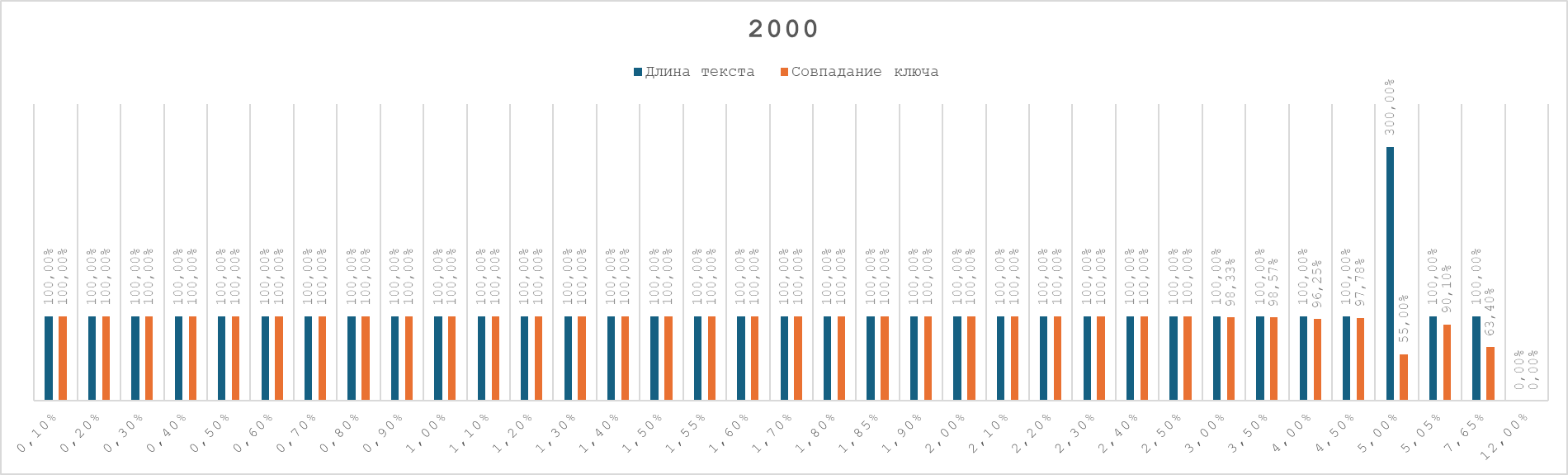


Рисунок 7. Диаграммы зависимости правильности определения ключа от соотношения длин ключа и исходного текста

По построенным диаграммам видно, что расшифровка текста происходит корректно, если длина ключа составляет не более 4% длины текста.

# Вывод

В ходе курсовой работы были реализованы программы “Кодер” и “Декодер”, корректно осуществляющие соответственно зашифровку и расшифровку английского текста с помощью операции простой XOR. Получена зависимость корректности расшифровки в зависимости от соотношений длин ключа и исходного текста. По результатам тестов, программа корректно расшифровывает текст, если длина ключа не превышает 4% от длины текста.

# Список источников

1. Брюс Шнайер “Прикладная криптография”;
2. <https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D0%B4%D0%B5%D0%BA%D1%81_%D1%81%D0%BE%D0%B2%D0%BF%D0%B0%D0%B4%D0%B5%D0%BD%D0%B8%D0%B9>

# Приложение

Таблицы зависимости корректности определения длины и символов ключа от соотношения длин ключа и исходного текста: (будет)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 100 | | | 500 | | | 2000 | | |
| 1,0% | 100,0% | 100,0% | 0,2% | 100,0% | 100,0% | 0,1% | 100,0% | 100,0% |
| 2,0% | 100,0% | 100,0% | 0,4% | 50,0% | 0,0% | 0,2% | 100,0% | 100,0% |
| 3,0% | 100,0% | 100,0% | 0,6% | 100,0% | 100,0% | 0,3% | 100,0% | 100,0% |
| 4,0% | 100,0% | 100,0% | 0,8% | 100,0% | 100,0% | 0,4% | 100,0% | 100,0% |
| 5,0% | 20,0% | 0,0% | 1,0% | 100,0% | 100,0% | 0,5% | 100,0% | 100,0% |
| 6,0% | 200,0% | 83,3% | 1,2% | 100,0% | 100,0% | 0,6% | 100,0% | 100,0% |
| 7,0% | 200,0% | 71,4% | 1,4% | 100,0% | 100,0% | 0,7% | 100,0% | 100,0% |
| 8,0% | 100,0% | 87,5% | 1,6% | 200,0% | 100,0% | 0,8% | 100,0% | 100,0% |
| 9,0% | 0,0% | 0,0% | 1,8% | 200,0% | 88,9% | 0,9% | 100,0% | 100,0% |
| 10,0% | 20,0% | 0,0% | 2,0% | 100,0% | 100,0% | 1,0% | 100,0% | 100,0% |
| 11,0% | 200,0% | 54,5% | 2,2% | 200,0% | 100,0% | 1,1% | 100,0% | 100,0% |
| 12,0% | 33,3% | 0,0% | 2,4% | 100,0% | 100,0% | 1,2% | 100,0% | 100,0% |
| 13,0% | 200,0% | 38,5% | 2,6% | 100,0% | 100,0% | 1,3% | 100,0% | 100,0% |
| 14,0% | 7,1% | 0,0% | 2,8% | 100,0% | 100,0% | 1,4% | 100,0% | 100,0% |
| 15,0% | 126,7% | 13,3% | 3,0% | 6,7% | 0,0% | 1,5% | 100,0% | 100,0% |
| 16,0% | 6,3% | 0,0% | 3,2% | 100,0% | 100,0% | 1,6% | 100,0% | 100,0% |
| 17,0% | 0,0% | 0,0% | 3,4% | 100,0% | 94,1% | 1,6% | 100,0% | 100,0% |
| 18,0% | 55,6% | 0,0% | 3,6% | 100,0% | 94,4% | 1,7% | 100,0% | 100,0% |
| 19,0% | 5,3% | 0,0% | 3,8% | 300,0% | 47,4% | 1,8% | 100,0% | 100,0% |
| 20,0% | 5,0% | 0,0% | 4,0% | 300,0% | 75,0% | 1,9% | 100,0% | 100,0% |
| 21,0% | 4,8% | 0,0% | 4,2% | 200,0% | 81,0% | 1,9% | 100,0% | 100,0% |
| 22,0% | 4,5% | 0,0% | 4,4% | 100,0% | 100,0% | 2,0% | 100,0% | 100,0% |
| 23,0% | 4,3% | 0,0% | 4,6% | 0,0% | 0,0% | 2,1% | 100,0% | 100,0% |
| 24,0% | 4,2% | 0,0% | 4,8% | 600,0% | 33,3% | 2,2% | 100,0% | 100,0% |
| 25,0% | 48,0% | 4,0% | 5,0% | 200,0% | 0,0% | 2,3% | 100,0% | 100,0% |
| 26,0% | 3,8% | 0,0% | 5,2% | 600,0% | 0,0% | 2,4% | 100,0% | 100,0% |
| 27,0% | 3,7% | 0,0% | 5,4% | 0,0% | 0,0% | 2,5% | 100,0% | 100,0% |
| 28,0% | 100,0% | 39,3% | 5,6% | 400,0% | 0,0% | 3,0% | 100,0% | 98,3% |
| 29,0% | 0,0% | 0,0% | 5,8% | 0,0% | 0,0% | 3,5% | 100,0% | 98,6% |
| 30,0% | 56,7% | 0,0% | 6,0% | 100,0% | 76,7% | 4,0% | 100,0% | 96,3% |
| 31,0% | 29,0% | 0,0% | 6,2% | 100,0% | 90,3% | 4,5% | 100,0% | 97,8% |
| 32,0% | 0,0% | 0,0% | 6,4% | 0,0% | 0,0% | 5,0% | 300,0% | 55,0% |