Министерство образования и науки Российской Федерации

Санкт-Петербургский политехнический университет Петра Великого

—

Институт компьютерных наук и технологий

**Кафедра «Информационная безопасность компьютерных систем»**

**Курсовая работа**

**Математический подход к раскрытию шифров**

по дисциплине «Методы программирования»

|  |  |  |
| --- | --- | --- |
| Выполнили  студенты гр. 13508/11 | подпись | Н.А. Супротивный |
|  | подпись | Г.С. Кубрин |
|  | подпись | Д.А. Куликов |
| Руководитель  старший преподаватель | подпись | П.В. Семьянов |

Санкт-Петербург

2016

**СОДЕРЖАНИЕ**

[Цель работы 3](#_Toc451737746)

[Задачи 3](#_Toc451737747)

[Ход работы 4](#_Toc451737748)

[Введение 4](#_Toc451737749)

[Определение длины ключевого слова 5](#_Toc451737750)

[Подбор ключа 7](#_Toc451737751)

[Определение перестановок 9](#_Toc451737752)

[Лексический анализ 10](#_Toc451737753)

[Программная реализация 13](#_Toc451737754)

[Результаты работы 15](#_Toc451737755)

[Вывод 16](#_Toc451737756)

[Список литературы 16](#_Toc451737757)

[Приложения 17](#_Toc451737758)

[Приложение 1. Квадрат Виженера. Пример. 17](#_Toc451737759)

[Приложение 2. Метод Касиски. 18](#_Toc451737760)

[Приложение 3. Тест Фридмана 20](#_Toc451737761)

[Приложение 4. Длина ключа 21](#_Toc451737762)

[Приложение 5. Определение перестановок 22](#_Toc451737763)

**Цель работы**

Изучить алгоритмы криптоанализа шифра Виженера и реализовать их на языке *С*.

**З**адачи

1. Написать программный инструмент криптоанализа, который в качестве входных данных принимает шифрованное сообщение и, в предположении, что оно зашифровано с помощью шифра Виженера, позволяет расшифровать его. Программа должна находить в процессе работы ключевое слово и квадрат Виженера.
2. С помощью разработанного инструмента расшифровать представленный в русском переводе книги Чарльза Уэзерелл «Этюды программистов» [1] шифротекст.

**Ход работы**

**Введение**

В простейшем общем классе подстановочных шифров для построения правила шифрования используется некоторый смешанный алфавит, например, перестановка обычного алфавита.

Перед нами же стоит задача вскрытия шифра Виженера, а именно.

Таблица шифрования символов (см. приложение 1) строится на основе некоторого смешанного алфавита, который для каждой следующей буквы смещается на единицу влево, а первая буква отправляется в конец, таким образом получается 32 смешанных алфавитов, каждому из которых соответствует одна из букв русского языка.

Далее выбирается некоторое ключевое слово, которое записывается под исходным текстом, и каждая буква исходного текста шифруется при помощи алфавита, соответствующего той букве ключевого слова, которая стоит под данной буквой исходного текста.

Такой способ шифрования не подлежит раскрытию при помощи простого подсчета частот букв, поскольку одна и та же буква исходного текста шифруется по-разному в зависимости от выпавшей на нее буквы ключевого слова.

Далее текст разбивается на группы по 5 букв, это скрывает исходную структуру текста, разбитого на слова, что сильно усложняет синтаксическую обработку.

Таким образом, к расшифровке предлагается текст такого вида:

ЖНФЖП ЕЕЫШВ ЛПЖАТ ГФБЦМ КЖЬЗА ЮЪИВУ ЩЖРСЮ Б**ЬЬК**Ь ЫЕСУУ ЦТЮБШ УНЖНЦ ЭЭШЮЗ УЬЕКН АУЕЫЩ ШЖРЬЙ ЛЮП**КН ДЙЯГЭ** ЪНЫГЖ ОУШИШ УФГВР ШМАГВ ВУВОС ЗХЧИУ ГНЛАЯ **ЬЬК**ИЯ РЦЖРЫ АХЪВИ ЖГЭЯЦ СЪУЫФ ЯРМЗФ ЧФЬЩС ЬФШВЕ ОМКТИ МБЭВЪ КФХЙЦ ХНЬЮЬ МФЛБИ МРУЛМ ЯЗФЧЪ ЪЧЗНК ЗНИВЛ НШГЛЩ ИЛЗНФ ФУЖ**КН ДЙЯГЭ** ЕУЮЛЛ ЮЖНЯИ ЕМДЙШ ГЯУГВ ЦФЩВЮ МФАГЯ ВХМЭВ ВФПГФ ФЖККГ ЦМЛЫБ ШМПУЕ ШЖЛЯЮ ЯРЧВЪ ЖУПВМ КЛЫЭС ЭЧИРЫ ГЫЩЗЗ ЗКЖЛЕ ШВРЪЧ ЪААЖЗ ДХЪФС БРНМЪ КЫБЪФ УНЦЮБ ТЖУНЯ ЕШИМУ КФВГВ ГЧМЭВ ЗРВМЪ ЪЕЕТО ЯЦБЖГ ВИЖМД КЗЗПА ФЯВНР ЫГЮЩЭ ЯЫЦШЪ ЧНГВЫ АХЪВЛ НШАПВ ЧОЬОЙ КЮАШО КЗЛЩУ ШЯРНЗ ГХЛТЮ ЖЫШШГ ППЬЫШ АЬФМА ФЕЙЗА ЙПЛУЭ ЖЛЗИЗ НЖККР ЦЯДЧК НДЙЯГ ЭБФЬА ВБЭКЗ ФКЫТВ ЛЕЪЭЯ ЛЭЩЗН ФХГЧК ТКЫЮЗ ЗЪУЖА ПВЧОЬ ОЙКЕС ЛЗАЮЪ ИВУНЫ ПКЗВЯ ЪГОСЩ ЛБЬГМ ЯВЗГЬ КШЪГЙ ЕНПСМ ЭВГОГ ЧСОРГ ЩОЦМВ ДГЩКЧ ЮЗВЗК ЦЧЯРЧ ВЪЖФЫ ЕЛЖАЪ УССХР УОЬЫЕ ЙГЫОТ УЕАГЖ ГЫСЩИ ЯРВТЮ ДЖНЛГ ЦМЗЬЪ ЯИЦТР ЕМИКЦ ЩВЦОР ЛХМХЖ ВРЬПУ ГВЯРЬ ПМЯЖЖ РЧПШЪ ЧУВГЧ СЕЕГЦ ЬПЗДМ ОЬОЧЗ КВУФЯ УПОХЪ ГЪЭЯЖ ВЖФ

**Текст 1**

Определение длины ключевого слова

Главный недостаток шифра Виженера состоит в том, что его ключ повторяется.

Благодаря этому можно заметить повторяющиеся группы букв КНДЙЯГЭ, ЬЬК (Текст 1). Это может свидетельствовать тому, что эти буквы зашифрованы одним и тем же смешанным алфавитом, соответствующим одной и той же букве ключевого слова, а следовательно, если совпадение не случайное, то расстояние между ними кратно длине ключевого слова (этого не наблюдается, если ключ случайный, его длина равна длине сообщения, и он использовался единожды, тогда шифр Виженера теоретически будет невзламываемым, для такого шифра доказана абсолютная криптостойкость).

**1.1** Первым повторяющиеся группы символов использовал Фридрих Касиски, поэтому этот метод нахождения длины ключа и назван его именем.

Для осуществления этого метода была написана функция (реализация функции см. Приложение.1.), которая ищет совпадающие группы символов, если длина группы больше трех, тогда в динамический массив заносится длина совпавших групп и расстояние между ними.

Чарльз Уэзерелл предлагает в качестве длин ключа брать то число, которое делит 90% или более из этих расстояний, но чтобы отдать приоритет совпадениям с большими длинами, для каждой длины ключа считается коэффициент h:

Где hi коэффициент для длины ключа i.

Lj – длины совпадений, расстояния которых кратны i.

Массив h сортируется по убыванию, а соответственно по убыванию вероятности длины ключа, т.е. чем больше hi, тем выше вероятность что длина ключевого слова i.

**1.2** Следующий шаг к определению длины ключа – тест Фридмана. (см. Приложение.2.)

Этот тест основывается на индексе совпадения, который измеряет частоты повторения символов. Т.е. зная ожидаемый индекс совпадения для естественного языка (для русского языка 0.0553), можно ожидать, что зашифрованный текст на русском языке будет обладать таким же индексом, потому что данная характеристика не меняется, если текст зашифрован с использованием моноалфавитного шифра, такого, как шифр Цезаря.

Так как буквы текста, соответствующие одинаковым буквам ключевого слова, зашифрованы одним и тем же смешанным алфавитом, то этот метод сработает и в нашем случае.

Для этого надо разбить текст на группы, где алфавит каждой группы соответствует алфавиту напротив буквы ключевого в таблице Виженера, а позиции рассматриваемых букв кратны предполагаемой длине ключа.



Где  – количество появлений *i*-й буквы в группе , а *N* – общее число рассматриваемых букв в группе.

Таким образом количество групп равно предполагаемой длине ключа.

ИС рассчитывается для каждой группы. Диапазон рассматриваемых значений ключа 20, но его можно расширить.

После вычисления ИС для каждой группы, соответствующим всем длинам ключа, вычисляется ошибка для данных длин, равная:



Далее создается массив значений, длины ключей в котором отсортированы по возрастанию ошибки, соответственно по убыванию вероятности подлинности длины ключевого слова.

**1.3** Далее массивы с отсортированными по вероятности длинами ключевого слова подаются третей функции, которая формирует результирующий массив длин, сортируя по возрастанию ключи в зависимости от их номера в тестах Касиски и Фридмана. Этот массив и используется в дальнейшей работе (Приложение. 3.)

Подбор ключа

Поиск ключа осуществлялся с помощью вероятностного метода, предложенного переводчиками Этюдов, и словарного перебора. В связке с эффективным поиском по словарю, речь о котором пойдёт ниже, в разделе Лексический анализ, такой подход даёт достаточно точный результат при незначительном приросте времени работы.

Предположим, мы нашли длину ключа описанным выше способом, пусть она равна *k*. Следующим шагом необходимо разбить шифротекст на *k* групп таких, что все буквы в группе соответствуют одной букве ключевого слова: в *i*-ю группу попадут буквы *i, i + k, i + 2k,…* Подсчитаем для каждой группы количество каждой буквы.

На основе этих данных несложно подсчитать вероятности выпадения букв в каждой группе. Это значит, что эти данные стоит сравнить со статистическими: сделать частотный анализ. В работе использовалось опубликованное [2] Национальным корпусом русского языка распределение буквенных частот в русскоязычных текстах.

Рассмотрим биномиальное распределение вероятностей появления фактического числа букв , если обозначает в *k-*й группе обозначает букву исходного текста.

.

Из него с помощью формулы Байеса получим вероятности того, что буквав *k*-й группе означает букву  исходного текста:

,

где *n –* число букв алфавита.

Эта информация нам даёт возможность найти самое главное – сдвиги между группами. Так как каждая группа шифруется одним алфавитом, соответствующим букве ключа в таблице Виженера, что есть моноалфавитное шифрование, то алфавиты каждой группы сдвинуты на некоторое число позиций в таблице, равное разности букв в ключевом слове (имеется в виду разность позиций букв в алфавите языка).

Для нахождения сдвигов получена вероятность того, что между *k*-й и *l*-й группами имеется сдвиг *r*:

.

Вероятность фактического распределения числа появлений букв при условии, что имеет место сдвиг *r*:



Для текстов порядка 1000 символов, такие как шифротекст из Этюдов программистов, часто дают хорошие вероятности сдвигов, то есть такие, что фактические значения находятся на 6 – 7 месте в отсортированном массиве вероятностей сдвигов для интересуемых *k* и *l*. Однако задача перебора таблицы вероятностей не такая быстрая и простая задача даже при таких показателях.

Можно применить некоторые эвристики при переборе, например, воспользоваться уравнением для отброса некорректных сдвигов, либо использовать лингвистический анализ текста, частично расшифровывая текст на некотором промежуточном этапе, либо применить генетический алгоритм [3], критерием качества для которого будет лучшая вероятность подобранной комбинации букв ключевого слова.

Мы же используем перебор словарных ключей необходимой длины, подсчитываем вероятность того, что этот ключ фактический и возвращаем выборку наиболее вероятных.

Определение перестановок

Для формирования таблицы соответствий вычислим для всех букв вероятности  того, что буква  в зашифрованном тексте соответствует букве  в первой группе, - во второй и так далее.



где *d* – число групп.

Далее найдем отношение этих вероятностей, к сумме вероятностей того, что буква yj соответствует некоторой другой букве  для всех букв  русского алфавита. Естественно, чем больше получится тем больше вероятность, что именно xi в первой группесоответствует зашифрованного текста.



Для большинства букв максимальные значения  дают верное соответствие, но часто при выборе максимальных значений получается так, что для одной и той же буквы возможны 2 и более варианта, в которых она оптимальна, поэтому таблица соответствий строится следующим образом:

1. Выбирается максимальное значение  из всех возможных , .
2. В соответствие  в зашифрованном тексте ставится xi в первой группе,  во второй и т.д.
3. Буквы  и  вычеркиваются из рассмотрения как буквы исходного и зашифрованного текстов соответственно.
4. Возврат в пункт 1.

Таким образом, если для буквы и есть 2 или более вариантов с большим значением p(xi|yj), то эта буква задействуется с максимальным значением, либо ее оптимальное соотношение было отброшено для более выгодного соответствия других букв.

(Реализация Приложение 4.)

Лексический анализ

**2.1 Представления словаря и задача поиска**

Для реализации алгоритмов из этого раздела использовался готовый файл формата .txt содержащий большое количество слов русского языка. Основная проблема, возникающая при работе со словарем, который представляет собой текстовый файл большой емкости, это низкая скорость поиска. Поскольку планировалось использовать поиск по словарю во многих алгоритмах, была поставлена задача увеличения скорости поиска слова в словаре.

Было рассмотрено несколько решений, все они основывались на использовании различных структур данных для хранения словаря. В итоге было решено использовать хэш-таблицы, потому что они способны дать максимально возможную скорость поиска.

Использования хэш-таблиц в задаче поиска в текстовом файле является возможным благодаря операции сдвига курсора. Таким образом, можно предварительно отсортировать словарь необходимым образом и сохранить в отдельный файл некоторые сдвиги. Например, в алгоритмах из этого раздела использовались две модификации словаря с хэш-таблицами: отсортированный по алфавиту, и отсортированный по длине слова. В первом случае в хэш-таблице сохранялся символ алфавита и сдвиг, приводящий к первому слову в файле, начинающемуся с данного символа. Во втором случае в хэш таблице аналогичным образом сохранялись длины слов и соответствующие сдвиги.

Таким образом созданные хэш-таблицы не дают моментального соответствия для любого запроса (то есть реализованные хэш-функции не являются идеальными), но они значительно сокращают линейный перебор словаря. Также для еще большего сокращения перебора слов используется метод барьера. В первом случае барьером является слово, начинающееся на букву отличную от искомого слова. А во втором, слово с длинной отличной от длины исходного слова.

В итоговой реализации, как словарь, так и хеш-таблица хранятся в заранее созданных файлах, и в течение выполнения поиска не происходит их полного извлечения в оперативную память.

Алгоритм поиска слова в словаре в общем виде выглядит следующим образом. На вход подается слово. Сначала происходит сдвиг курсора согласно результату хэш-функции. Если слово не найдено, происходит линейный поиск по файлу, пока слово не найдется, или пока не встретится барьер.

**2.2 Поиск слов, частично соответствующих искомому**

Для реализации сбора статистики о тексте было необходимо собрать информации о количестве слов из словаря, совпадающих со словами из текста по одной, двум, и т.д. буквам. Алгоритм, выполняющий поиск слов, частично соответствующих искомому, работает следующим образом. С помощью хэш-таблицы происходит переход по словарю, отсортированному по длине слов, к месту, где начинаются слова с длиной, равной длине исходного слова. Далее в цикле происходит обход словаря до тех пор, пока не встретится слово с длиной, не соответствующей искомому. На каждой итерации внешнего цикла запускается внутренний цикл, подсчитывающий количество совпадающих символов в текущем слове из словаря с исходным искомым. Далее происходит проверка на превышение количества совпадений символов в текущей паре слов с максимум. Если текущее количество совпадений превышает максимум, то оно становится новым максимумом и массив найденных слов отчищается. Если количество текущих совпадений совпадает с максимумом, то словарное слово записывается в массив найденных слов. После окончания внешнего цикла на выход отправляется массив слов, имеющих максимальное количество символов, совпадающих с соответствующими символами искомого слова.

**2.3 Проверка расшифрованного текста на правильность**

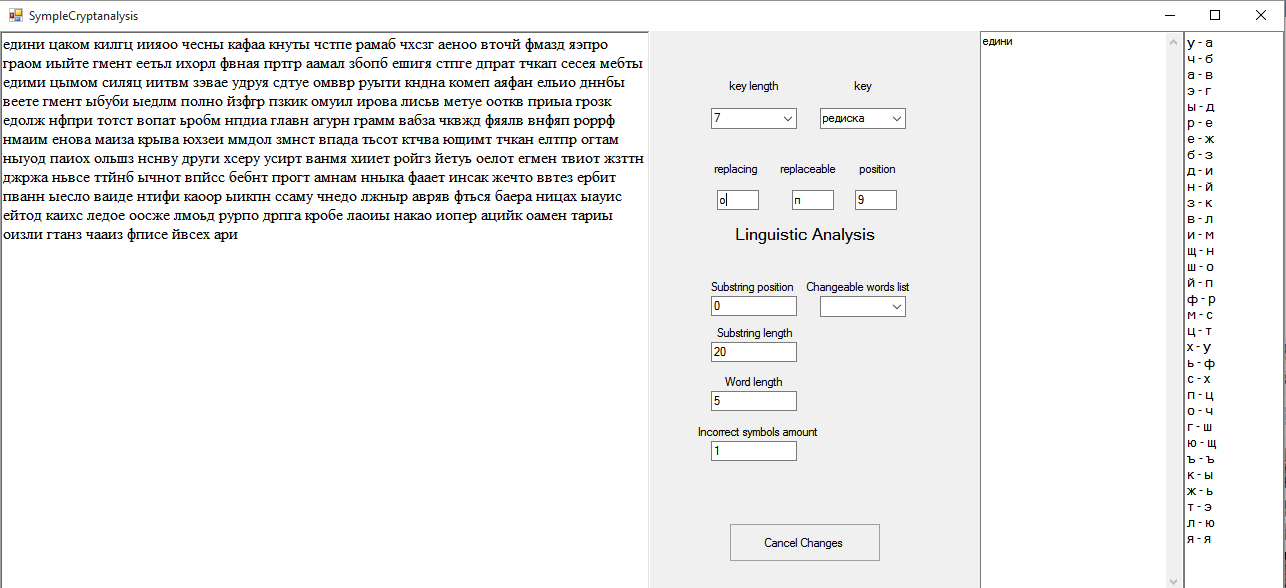
В ходе работы над проектом возникла необходимость в алгоритме, способно оценить, правильно ли расшифрован текст, для повышения степени автоматизации процесса расшифровки. В качестве критерия оценки текста в данном алгоритме было использовано отношение количество верных (найденных в словаре) слов длиной более трех символов к общему количеству символов в тексте. Для оценки такого значения критерия, при котором можно считать текст расшифрованным, критерий был рассчитан для не зашифрованных текстов разной длины и стилистики. В качестве критерия был взято усредненное значение. Алгоритм проверки текста возвращает значение типа bool и работает следующим образом.

На вход подается проверяемый текст. В цикле текст проходится посимвольно, и на каждой итерации цикла происходит поиск подстрок длины более чем три символа, начинающихся с текущей позиции, в словаре. Если подстрока найдена, то происходит приращение счетчика. Когда посимвольный обход текста заканчивается, значения счетчика делится на количество символов и сравнивается с заранее рассчитанным значением критерия. На основе сравнения делается вывод о правильности текста.

Для реализации поиска словарного слова в этом алгоритме использовался словарь, отсортированный по алфавиту.

Программная реализация

представляет из себя готовое решение для криптоанализа, с помощью алгоритмов. Пользователю предлагается поэтапная работа с шифротекстом с возможностью выбора на каждом этапе ключевых параметров шифра из наиболее вероятных, изменения других различными способами. Неоднозначность решения – не большая проблема в большинстве случаев, так как криптоаналитик может перебрать все параметры, откатившись до любого этапа и начав всё сначала.



Интерфейс многофункционален, но при этом работа не затрудняется, так как набор элементов управления минимально необходимый. Пользователю предоставляется работа с текстовым полем для самого шифротекста, полями для вывода таблицы соответствий, результатов работы лексического анализа, но всё взаимодействие преимущественно осуществляется с помощью центральной панели, на которой можно найти выпадающие списки, которые при первом же с ними взаимодействии включаются в работу.

Структура исходного кода разделена на четыре раздела, каждый из которых хранится в отдельном файле и доступен в своём пространстве имён: *VigenereAnalysis, LinguisticAnalysis, WordProcessing, FrequencyAnalysis.* Форма (интерфейс) описана в разделе *SympleCryptanalysis.* Пользователь запускает цепочку взаимодействий и передачи аргументов из формы во все остальные разделы.

Ознакомится с исходным кодом проекта можно на GitHub <https://github.com/OneTwoDevGroup/SympleCryptanalysis/>

Раздел *VigenereAnalysis* содержит основные функции статистического анализа текста: тест Фридмана, метод Касиски, поиск ключа, формирование расшифрованного текста. *LinguisticAnalysis*: поиск по словарю, поиск частично совпадающих слов из текста в словаре, замена выбранных слов в тексте. *FrequencyAnalysis* используется для подсчёта количества букв в тексте, но содержит также функции частотного анализа, от которых пришлось отказаться.

*WordProcessing* ответственна за две важных составляющих проекта, которые представлены в виде двух классов. Первый отвечает за описание и работу с языковыми параметрами шифротекста. В каждом разделе создаётся экземпляр этого класса с указанием на использованный язык, что позволяет использовать данные о хранении алфавита в памяти, статистические данные, функции работы с отдельными буквами. Второй в свою очередь реализует контроль версий шифротекста и таблицы соответствий для возможности отката изменений. Как следствие этот класс – связующее звено во взаимодействии с разделами.

**Результаты работы**

1. Для предложенного к расшифровке текста значения h полученные по методу Касиски имеют следующий вид:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ВL: | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
| HhL: | 909 | 16 | 438 | 183 | 16 | 909 | 312 | 0 | 183 | 0 | 0 | 0 | 909 | 16 | 271 | 0 | 0 | 126 |

Как видно, максимальными являются коэффициенты, соответствующие длинам 2, 7 и 14.Причем можно заметить, что они минимум в 2 раза больше остальных значений, что позволяет сделать вывод об эффективности этого метода.

Далее проводится тест Фридмана на соответствие индексов совпадения групп символов индексу совпадения русского языка.

Самым лучшим вариантом является ключ длиной 7 букв. Для такой длины имеем следующие индексы совпадения групп:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| G0 | G1 | G2 | G3 | G4 | G5 | G6 |
| 0.0638 | 0.0536 | 0.0522 | 0.0420 | 0.0422 | 0.0458 | 0.0419 |

Тест Фридмана однозначно подтвердил, что действительной длиной ключа является 7. Так как для остальных длин, средний уровень ИС около 0.030-0.035.

Таким образом, эти два метода позволяют однозначно определить длину ключа даже для небольших текстов.

4. Описанный переводчиком способ нахождения перестановок и выбранный способ выбора наилучшего соответствия дал следующие результаты:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| y | а | б | в | г | д | е | ж | з | и | й | к | л | м | н | о | п | р | с | т | у | ф | х | ц | ч | ш | щ | ъ | ы | ь | э | ю | я |
| x | у | ч | а | э | ы | р | е | б | г | н | з | в | и | щ | ш | й | ф | м | ц | х | ь | с | п | о | д | ю | ъ | к | ж | т | л | я |

После дальнейшей расшифровки “вручную” была найдена истинная таблица соответствий:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| y | а | б | в | г | д | е | ж | з | и | й | к | л | м | н | о | п | р | с | т | у | ф | х | ц | ч | ш | щ | ъ | ы | ь | э | ю | я |
| x | п | х | а | э | ы | р | у | в | ч | н | ж | з | и | щ | ш | й | ф | м | ц | б | ь | с | у | о | г | ю | ъ | к | д | т | л | я |

Реализованной функцией правильно составлено 21 соответствие. Далее с помощью лексической обработки текст легко расшифровывается.

При большей длине текста (порядка 1500 символов) в некоторых случаях текст вовсе расшифровывается верно автоматически.

Вывод

Большинство методов классического криптоанализа основано на статистическом подходе, который дает неоднозначный результат, особенно для шифротекстов небольшого размера. Мы применили два подхода к решению этой проблемы.

Для лучшего результата пользователю дается контроль над процессом дешифрования, путем предоставления ему выбора из наиболее вероятных промежуточных результатов и некоторой отладочной информации.

Усиление математического аппарата позволяет увеличить точность результатов статистических методов криптоанализа. Простые подходы к расшифровке классических шифров, таких как шифр Цезаря, например, монограммный частотный анализ, могут быть неэффективны и не привести к правильному результату вовсе за адекватное время.

**Список литературы**

1. Уэзерелл, Ч. Этюды для программистов. Ч. Уэзерелл – Москва: Мир, 1982. – 288 с.
2. Ляшевская , О. Н. Новый частотный словарь русской лексики/ О. Н. Ляшевская, С. А. Шаров
3. Взлом шифра Виженера с помощью частотного криптоанализа [Электронный ресурс] / 2014 – Режим доступа: https://habrahabr.ru/post/221485/, свободный.

Приложения

Приложение 1. Квадрат Виженера. Пример.

АБВГДЕЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯ

А ХБМЙДЖЗИКЛфНОПЩЕРШСЯТУЦЧАЪЭГЫЬВЮ

Б БМЙДЖЗИКЛфНОПЩЕРШСЯТУЦЧАЪЭГЫЬВЮХ

В МЙДЖЗИКЛфНОПЩЕРШСЯТУЦЧАЪЭГЫЬВЮХБ

Г ЙДЖЗИКЛфНОПЩЕРШСЯТУЦЧАЪЭГЫЬВЮХБМ

Д ДЖЗИКЛфНОПЩЕРШСЯТУЦЧАЪЭГЫЬВЮХБМЙ

Е ЖЗИКЛфНОПЩЕРШСЯТУЦЧАЪЭГЫЬВЮХБМЙД

Ж ЗИКЛфНОПЩЕРШСЯТУЦЧАЪЭГЫЬВЮХБМЙДЖ

З ИКЛфНОПЩЕРШСЯТУЦЧАЪЭГЫЬВЮХБМЙДЖЗ

И КЛфНОПЩЕРШСЯТУЦЧАЪЭГЫЬВЮХБМЙДЖЗИ

Й ЛфНОПЩЕРШСЯТУЦЧАЪЭГЫЬВЮХБМЙДЖЗИК

К фНОПЩЕРШСЯТУЦЧАЪЭГЫЬВЮХБМЙДЖЗИКЛ

Л НОПЩЕРШСЯТУЦЧАЪЭГЫЬВЮХБМЙДЖЗИКЛф

М ОПЩЕРШСЯТУЦЧАЪЭГЫЬВЮХБМЙДЖЗИКЛфН

Н ПЩЕРШСЯТУЦЧАЪЭГЫЬВЮХБМЙДЖЗИКЛфНО

О ЩЕРШСЯТУЦЧАЪЭГЫЬВЮХБМЙДЖЗИКЛфНОП

П ЕРШСЯТУЦЧАЪЭГЫЬВЮХБМЙДЖЗИКЛфНОПЩ

Р РШСЯТУЦЧАЪЭГЫЬВЮХБМЙДЖЗИКЛфНОПЩЕ

С ШСЯТУЦЧАЪЭГЫЬВЮХБМЙДЖЗИКЛфНОПЩЕР

Т СЯТУЦЧАЪЭГЫЬВЮХБМЙДЖЗИКЛфНОПЩЕРШ

У ЯТУЦЧАЪЭГЫЬВЮХБМЙДЖЗИКЛфНОПЩЕРШС

Ф ТУЦЧАЪЭГЫЬВЮХБМЙДЖЗИКЛфНОПЩЕРШСЯ

Х УЦЧАЪЭГЫЬВЮХБМЙДЖЗИКЛфНОПЩЕРШСЯТ

Ц ЦЧАЪЭГЫЬВЮХБМЙДЖЗИКЛфНОПЩЕРШСЯТУ

Ч ЧАЪЭГЫЬВЮХБМЙДЖЗИКЛфНОПЩЕРШСЯТУЦ

Ш АЪЭГЫЬВЮХБМЙДЖЗИКЛфНОПЩЕРШСЯТУЦЧ

Щ ЪЭГЫЬВЮХБМЙДЖЗИКЛфНОПЩЕРШСЯТУЦЧА

Ъ ЭГЫЬВЮХБМЙДЖЗИКЛфНОПЩЕРШСЯТУЦЧАЪ

Ы ГЫЬВЮХБМЙДЖЗИКЛфНОПЩЕРШСЯТУЦЧАЪЭ

Ь ЫЬВЮХБМЙДЖЗИКЛфНОПЩЕРШСЯТУЦЧАЪЭГ

Э ЬВЮХБМЙДЖЗИКЛфНОПЩЕРШСЯТУЦЧАЪЭГЫ

Ю ВЮХБМЙДЖЗИКЛфНОПЩЕРШСЯТУЦЧАЪЭГЫЬ

Я ЮХБМЙДЖЗИКЛфНОПЩЕРШСЯТУЦЧАЪЭГЫЬВ

Приложение 2. Метод Касиски.

int\* KasiskiExamination(String^ text)

{

int size;

unsigned char alf[R] =

'а','б','в','г','д','е','ж','з','и','й','к','л','м','н','о','п',

'р','с','т','у','ф','х','ц','ч','ш','щ','э','ъ','ы','ь','ю','я'

};

// в lenght\_distane[][0] заносится длина совпадения,

// в lenght\_distance[][1] расстояние между совпадениями

int\*\* lenght\_distance;

int w = 0; // номер совпадения

float best;

bool flag = 1;

int i,j,l,g,z,s,k,f,maxind; //счетчики

unsigned char S; //строка с шифром, место поиска

S = (unsigned char\*)System::Runtime::InteropServices::Marshal::StringToHGlobalAnsi(text);

i = 0;

size = strlen((const char\*)S);

lenght\_distance = (int\*\*)malloc(sizeof(int\*));

for (i = 0; i < R; i++) // i-прогоняемся по всем символам алфавита

{

for (j = 0; j < size; j++) // ищем все символы Alf[i] в S[]

if (S[j] == alf[i])

{

s = 0; //счетчик пробелов

for (l = j + 1; l < size; l++) // l - ищем все символы Alf[i] в строке S[j+]

{

if ((S[l] < MAXLOW)||(S[l]>MAXHIGH)) //отбрасываем пробельные символы

{

s++;

}

if (S[l] == alf[i])

{

g = 0;

k = 0;

f = 0;

/\*g считаем сколько символов совпало\*/

while ( ((l + g+f) < size) && ((j + g+k) < size)

&& ((S[l + g+f] == S[j +g+k])) )

{

g++;

/\* отбрасываем пробельные символы\*/

while (

((l + g + f) < size) && ((j + g + k) < size) &&

((S[l + g + f] < MAXLOW) || (S[l + g + f]>MAXHIGH))

) f++;

/\*отбрасываем пробельные символы\*/

while (

((l + g + f) < size) && ((j + g + k) < size) &&

((S[j + g + k] < MAXLOW) || (S[j + g + k]>MAXHIGH))

) k++;

}

if (g > 3)

{

lenght\_distance = (int\*\*)realloc( lenght\_distance, sizeof(int\*)\*(w + 1) );

lenght\_distance[w] = (int\*)malloc( 2 \* (sizeof(int)) );

lenght\_distance[w][0] = g; /\*длина совпадения\*/

lenght\_distance[w][1] = l - j - s; /\*расстояние между совпадениями\*/

w++; //счетчик совпадений

}

}

}

}

}

float h[u];

for (j = 2; j <u; j++)

{

h[j] = 0;

for (i = 0; i < w; i++)

{

if ((lenght\_distance[i][1] % j) == 0)

//умножаем на длину совпадения

h[j] +=1.0\*length\_distance[i][0]\*length\_distance[i][0];

}

}

static int besti[u];

float max;

for (i = 0; i < u; i++)

{

max = -0.5;

for (j = 2; j < u;j++)

if (h[j]>max)

{

max = h[j];

maxind = j;

}

besti[i] = maxind;

h[maxind] = -1;

}

//ЗАЧИСТКА

for (i = 0; i < w; i++)

{

free(lenght\_distance[i]);

}

free(lenght\_distance);

return besti;

}

Приложение 3. Тест Фридмана

int\* Index (String^ text)

{

unsigned char alf[R] = { 'а','б','в','г','д','е','ж','з','и','й','к','л','м','н','о','п','р','с','т','у','ф','х','ц','ч','ш','щ','э','ъ','ы','ь','ю','я' };

unsigned char \*S = (unsigned char\*)(void\*)System::Runtime::InteropServices::Marshal::StringToHGlobalAnsi(text);

int i,j,l,k;

float IS[u][u];

int n = 0,f=0,g;

int size = strlen((const char\*)S);

for (i = 1; i < u; i++)

{

for (k = 0; k < i; k++)

{

IS[i][k] = 0.0;

for (j = 0; j < R - 1; j++)

{

n = 0;

f = 0;

l = 0;

g = i-k;

while(l < size) //Вычисление ИС

{

while ((g < i)&& (l < size))

{

while ((l < size) && ((S[l] <MAXLOW) || (S[l]>MAXHIGH)))

{

l++;

}

g++;

l++;

}

while ((l < size) && ((S[l] <MAXLOW) || (S[l]>MAXHIGH)))

{

l++;

}

g = 0;

if (((l < size)) && (S[l] == alf[j]))

{

n++;

f++;

}

else

{

n++;

}

}

IS[i][k] += f\*(f - 1);

}

IS[i][k] /= n\*(n - 1);

}

}

float average;

float max = FLT\_MAX;

float mindx;

static int besti2[u] = {0};

for (i = 0; i < u; i++) {

max = MAXINT64;

for (j = 2; j < u; j++)

{

mindx = 0;

for (k = 0; k < j; k++)

{

mindx += fabs(IS[j][k] -0.0550);

}

mindx /= k;

if ((mindx < max) && (IS[j][0]>-0.5))

{

max = mindx;

besti2[i] = j;

}

}

IS[besti2[i]][0] = -1;

}

return besti2;

}

Приложение 4. Длина ключа

int\* result(int\* besti1, int \*besti2)

{

int i, j, best,k,max1,max2;

float min;

static int result[u];

for (k = 0; k < u; k++)

{

min = LONG\_MAX;

best = 0;

max1 = 0;

max2 = 0;

for (i = 0; i < u; i++)

{

for (j = 0; j < u; j++)

{

if ((besti1[i] == besti2[j]) && (i + j < min))

{

if (besti1[i] && besti2[j])

{

min = i + j;

best = besti1[i];

max1 = i;

max2 = j;

}

}

}

}

result[k] = best;

besti1[max1] = 0;

besti2[max2] = 0;

}

return result;

}

Приложение 5. Определение перестановок

int \*freq(String^word, int lenghtkey)

{

int i, j, k, dist;

double new\_amount\_probolity[R][R] = { 0 };

for (i = 0; i < R; i++)

for (j = 0; j < R; j++)

new\_amount\_probolity[i][j] = 1;

double sum = 0;

for (i = 0; i < R; i++)

for (j = 0; j < R; j++)

{

for (k = 0; k < lenghtkey; k++)

{

dist = i + word[0] - word[k];

if (dist < 0)

dist += 32;

dist %= 32;

new\_amount\_probolity[i][j] \*= amount\_probolity[k][j][dist];

}

}

for (i = 0; i < R; i++)

for (j = 0; j < R; j++)

{

sum = 0;

for (k = 0; k <R ; k++)

{

sum += new\_amount\_probolity[k][j];

}

if (sum)

probolity[i][j] = new\_amount\_probolity[i][j] / sum;

else

probolity[i][j] = 0;

}

unsigned char ind[R][2];

for (i = 0; i < R; i++)

{

ind[i][0] = 255;

ind[i][1] = 255;

}

int z,g;

bool flag = 1;

double max = -1;

int save[2];

for (k = 0; k < R; k++)

{

max = -1;

for (i = 0; i < R; i++)

{

for (j = 0; j < R; j++)

{

if (max < probolity[i][j])

{

flag = 1;

for (z = 0; z < R; z++)

{

if ((ind[z][0] == i) || (ind[z][1] == j))

{

flag = 0;

}

}

if (flag)

{

max = probolity[i][j];

ind[k][0] = i;

ind[k][1] = j;

}

}

}

}

}

for (i = 0; i < R; i++)

{

ind[i][0] += 192;

ind[i][1] += 192;

}

int\*conformity = (int\*)malloc(sizeof(int)\*MAXALPHLEN);

for (i = 0; i < R; i++)

{

conformity[ind[i][1]-192] = ind[i][0]-192;

}

return conformity;

}