Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное  
учреждение высшего образования  
"Владимирский государственный университет имени Александра  
Григорьевича и Николая Григорьевича Столетовых"

(ВлГУ)

**КУРСОВАЯ РАБОТА**

Дисциплина: Методы и средства криптографической защиты информации.

|  |  |  |
| --- | --- | --- |
| Исполнитель темы |  | ст. гр. ИСБ-121 Розанов Л. А. |
|  | (подпись, дата) |  |
| Научный руководитель |  | Александров А.В. |
|  | (подпись, дата) |  |

Владимир 2022

**СОДЕРЖАНИЕ**

[1 ВЗЛОМ ПРОСТОЙ ПОДСТАНОВКИ 3](#_Toc122985853)

[2 ГАММИРОВАНИЕ(ШИФР ВИЖЕНЕРА) 6](#_Toc122985854)

[3 МНОГОРАУНДОВЫЙ БЛОЧНЫЙ ШИФР 8](#_Toc122985855)

[режим кодовой книги 10](#_Toc122985856)

[ПРИЛОЖЕНИЕ А 11](#_Toc122985857)

[ПРИЛОЖЕНИЕ Б 16](#_Toc122985858)

[ПРИЛОЖЕНИЕ в 21](#_Toc122985859)

[ЛИТЕРАТУРА 24](#_Toc122985860)

[БЛАГОДАРНОСТЬ 25](#_Toc122985861)

# 1 ВЗЛОМ ПРОСТОЙ ПОДСТАНОВКИ

* 1. **Определение простой подстановки.**

шифр, при котором каждый символ открытого текста заменяется на некоторый, фиксированный при данном ключе символ того же алфавита.

В шифрах простой замены замена производится только над одним-единственным символом. Для наглядной демонстрации шифра простой замены достаточно выписать под заданным алфавитом тот же алфавит, но в другом порядке или, например, со смещением. Записанный таким образом алфавит называют алфавитом замены.

Главный недостаток этого метода шифрования — это то, что последние буквы алфавита (которые имеют низкие коэффициенты при частотном анализе) имеют тенденцию оставаться в конце. Более защищённый способ построить алфавит замены состоит в том, чтобы выполнить колоночное перемещение (перемещение столбцов) в алфавите, используя ключевое слово, но это нечасто делается.

* 1. **Методология частотного анализа**

Частотный анализ — один из методов криптоанализа, основывающийся на предположении о существовании нетривиального статистического распределения отдельных символов и их последовательностей, как в открытом тексте, так и в шифротексте, которое, с точностью до замены символов, будет сохраняться в процессе шифрования и дешифрования.

Описание

Используется тот факт, что вероятность появления отдельных букв, а также их порядок в словах и фразах естественного языка подчиняются статистическим закономерностям: например, пара стоящих рядом букв «ся» в русском языке более вероятна, чем «цы», а «оь» в русском языке не встречается вовсе (зато часто встречается, например, в чеченском). Анализируя достаточно длинный текст, зашифрованный методом замены, можно по частотностям появления символов произвести обратную замену и восстановить исходный текст.

Вероятности встречаемости букв русского языка

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| символ | вер-ть | символ | вер-ть | символ | вер-ть |
| пробел | 0.175 | К | 0.028 | Ч | 0.012 |
| О | 0.089 | М | 0.026 | Й | 0.010 |
| Е | 0.072 | Д | 0.025 | Х | 0.009 |
| А | 0.062 | П | 0.023 | Ж | 0.007 |
| И | 0.062 | У | 0.021 | Ю | 0.006 |
| Н | 0.053 | Я | 0.018 | Ш | 0.006 |
| Т | 0.053 | Ы | 0.016 | Ц | 0.004 |
| С | 0.045 | З | 0.016 | Щ | 0.003 |
| Р | 0.040 | Ь | 0.014 | Э | 0.003 |
| В | 0.038 | Б | 0.014 | Ф | 0.002 |
| Л | 0.035 | Г | 0.013 |  |  |

* 1. **Детали взлома методом частотного анализа**

**ИСХОДНЫЙ ШИФР ТЕКСТ**

ПЙХБГХЫХЮХЭМНХБЙЩЦЮЩПНЕЗБНЙЦСЙЙГПЗБГЮНБЬХГКЮХБПХЦЗБГЩДСБМЗРЗ

ТБЛКШНБЛЗМЗШВМКДНБЫЮХТДХГЗДНБЫЩПЙХТМХПМЩАЩБЩЛНЩТЗБЖБЬЗЙГЩБЙ

ЫЮЗЭНПЗУБЙХЛЖБТХИЙГПНГХШВМЩБШНБЦЮЗЙНПКБЙГЗЮНММКХБЙСЫЩПКХБГЗЮ

ХШЦНБЦЩГЩЮКХБДКБГЗЦБПКЙЩЦЩБФХМНДБНШНБГЩШВЦЩБЩЮХЩШБТЮХПМЩЙГНБ

ЫЮНТЗХГБНДБЫЮХШХЙГВБПБМЗЭНБАШЗРЗ

**Посимвольная статистика  
Символ - Б встречается 38**

**Символ - Х встречается 25**

**Символ - Н встречается 21**

**Символ - З встречается 20**

**Символ - Щ встречается 20**

**Символ - Г встречается 17**

**Символ - Й встречается 16**

**Символ - Ю встречается 14**

**Символ - П встречается 13**

**Символ - М встречается 12**

**Символ - Ш встречается 10**

**Символ - К встречается 9**

**Символ - Ц встречается 9**

**Символ - Д встречается 7**

**Символ - Т встречается 7**

**Символ - Ы встречается 7**

**Символ - В встречается 4**

**Символ - Л встречается 4**

**Символ - С встречается 3**

**Символ - Э встречается 3**

**Символ - А встречается 2**

**Символ - Ж встречается 2**

**Символ - Р встречается 2**

**Символ - Ь встречается 2**

**Символ - Е встречается 1**

**Символ - И встречается 1**

**Символ - У встречается 1**

**Символ - Ф встречается 1**

**ИЗНАЧАЛЬНЫЙ ТЕКСТ**

все теперешние сокровища искусства три четыре века тому наза

д были банальными предметами повседневного обиода и часто с

прашиваю себи действительно ли красивы старинные суповые тар

елки которые мы так высоко ценим или только ореол древности

придает им прелесть в наши глаза

# **2 ГАММИРОВАНИЕ(ШИФР ВИЖЕНЕРА)**

**2.1 Описание шифра гаммирования**

Гамми́рование — метод симметричного шифрования, заключающийся в последовательности, состоящей из случайных чисел, на открытый текст. Последовательность случайных чисел называется гамма-последовательностью и используется для зашифровывания и расшифровывания данных. Суммирование обычно выполняется в каком-либо конечном поле

**2.2 Тест Казиска и атака Фридмана**

Метод Кази́ски — метод криптоанализа полиалфавитных шифров, таких как шифр Виженера. Основан на факте того, что повторяющиеся части открытого текста, зашифрованные одним и тем же ключевым словом, приводят к идентичным сегментам шифрованного текста. Разработан независимо криптоаналитиками Фридрихом Касиски и Чарльзом Бэббиджем.

Если повторяющаяся подстрока в открытом тексте зашифровывается одной и той же подстрокой в ключевом слове, тогда шифрованный текст содержит повторяющуюся подстроку, а расстояние между двумя вхождениями кратно длине ключевого слова.Расстояние между двумя повторяющимися подстроками в зашифрованном тексте g. Ключевое слово длиной k повторяется, чтобы заполнить длину зашифрованного текста, расстояние g кратно длине ключевого слова k. Таким образом, если мы видим две повторяющиеся подстроки с расстоянием g, то один из делителей g может быть длиной ключевого слова. Например, если расстояние равно g = 18, поскольку делители g равны 2, 3, 6, 9 и 18, один из них может быть длиной неизвестного ключевого слова.

Атака Фридмана — один из методов криптоанализа шифра Виженера. Описание было опубликовано Уильямом Фридманом в 1920 году. Метод основывается на вычислении вероятности того, что два случайных элемента текста совпадут. Эту вероятность называют индексом совпадений. Уильям Фридман показал, что значения индекса совпадений существенно отличаются для текстов различной природы. Это позволяет сначала определить длину ключа шифра, а затем найти и сам ключ. Появление метода индекса совпадений открыло новые возможности в криптоанализе шифра Виженера. По сравнению с распространённым в то время методом Касиски, новый метод был менее трудоёмким, требовал меньшей длины текста, был более пригоден для автоматизации и менее подвержен ошибкам. Индекс совпадений являлся более эффективным и допускал анализ шифров с длинными ключами.

**2.3 Детали взлома шифра Виженера**

**Исходный текст:**

цзалтсущхварзтбпхбяк\_хбызлушнвбэтуоцздфцтсашцмупкигузсакхиу\_цхщцзъещйюуьезфкпгчцжржцивяюхгуьзсхынъщшхсдэгаубнобнннфктсещшюэкквхцрйфф\_лэкоиубифупцоърхввщллхшыхпкцхуюмцлжжвбшзсмючяскхгвыиеьцщбухзфцщнмуыифюывхбфзсшухсющсвькэсящмрбфзтбьъияу

Так как мы знаем что длина ключа примерно от 3 до 6 букв я с помощью программы высчитал 10 самых частых биграмм и триграмм и уже на их основе подобрал ключ.

Код реализации представлен в Приложении А

Ключ:звук

**Расшифрованный текст:**

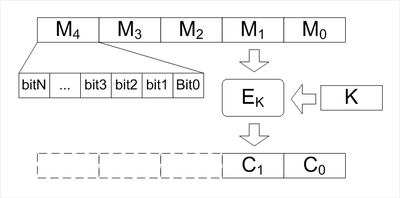
**ОДНАКО\_ОН\_НЕ\_ПОДНЯЛ\_ШТОР\_И\_НЕ\_ОТКРЫЛ\_БАЛКОННОЙ\_ДВЕРИ\_ОН\_НЕ\_ХОТЕЛ\_ЧОБЫ\_СДА\_ЗАГЛЯНУЛА\_ЛУНА\_С\_ОБРЕЧЕННОСТЬЮ\_ЧЕЛОВЕКА\_КООРЫЙ\_В\_БЛИЖАЙШИЙ\_ЖЕ\_ЧАС\_ДОЛЖЕН\_ПОГИБНУТЬ\_ОТ\_УДУШЬЯ\_ОН\_ОЩУПЬЮ\_НАПРАИЛСЯ\_К\_СВОЕЙ\_РАСКРЫТОЙ\_ОДИНОКОЙ\_И\_ХОЛОДНОЙ\_ПОСТЕЛИ**

# **3 МНОГОРАУНДОВЫЙ БЛОЧНЫЙ ШИФР**

**3.1 Краткое введение**

Бло́чный шифр — разновидность симметричного шифра[1], оперирующего группами бит фиксированной длины — блоками, характерный размер которых меняется в пределах 64‒256 бит. Если исходный текст (или его остаток) меньше размера блока, перед шифрованием его дополняют. Фактически, блочный шифр представляет собой подстановку на алфавите блоков, которая, как следствие, может быть моно- или полиалфавитной.[2] Блочный шифр является важной компонентой многих криптографических протоколов и широко используется для защиты данных, передаваемых по сети.

В отличие от шифроблокнота, где длина ключа равна длине сообщения, блочный шифр способен зашифровать одним ключом одно или несколько сообщений суммарной длиной больше, чем длина ключа. Передача малого по сравнению с сообщением ключа по зашифрованному каналу — задача значительно более простая и быстрая, чем передача самого сообщения или ключа такой же длины, что делает возможным его повседневное использование. Однако, при этом шифр перестаёт быть невзламываемым. От поточных шифров работа блочного отличается обработкой бит группами, а не потоком. При этом блочные шифры медленнее поточных.[3] Симметричные системы обладают преимуществом над асимметричными в скорости шифрования, что позволяет им оставаться актуальными, несмотря на более слабый механизм передачи ключа (получатель должен знать секретный ключ, который необходимо передать по уже налаженному зашифрованному каналу. В то же время, в асимметричных шифрах открытый ключ, необходимый для шифрования, могут знать все, и нет необходимости в передаче ключа шифрования).



Общая работы схемы блочного шифра.

**3.2 Криптографические примитивы**

Криптографические примитивы — низкоуровневые криптографические алгоритмы, которые часто используются для построения криптографических протоколов. В узком смысле это операции и процедуры, определяющие требуемые свойства криптосистемы.

Криптографические примитивы должны обладать следующими свойствами:

* Уровень безопасности. Обычно определяется количеством операций, требуемых (с использованием лучших методов, известных в настоящее время), чтобы достигнуть намеченную цель. Обычно уровень безопасности определяется необходимой верхней границей объёма работы. Это иногда называют рабочим фактором.
* Функциональность. Примитивы должны быть объединяемы для достижения различных целей. Какие примитивы наиболее эффективны для данной цели, будет определяться основными свойствами примитивов.
* Методы работы. Примитивы, когда они применяются в различных направлениях и при разных вложениях, как правило, проявляют различные характеристики; таким образом, один примитив мог бы использоваться по-разному, в зависимости от необходимости.
* Производительность. Это относится к эффективности примитива в определённом режиме работы (например, алгоритм шифрования может оцениваться по количеству битов в секунду, которую он может зашифровать).
* Простота реализации. Это относится к трудности реализации примитива в практической реализации, например реализации примитива в программной или аппаратной среде.Относительная важность различных критериев в значительной степени зависит от применения и имеющихся ресурсов. Например, в среде, где вычислительная мощность ограничена, возможно, придётся отказаться от очень высокого уровня безопасности для повышения производительности системы в целом.

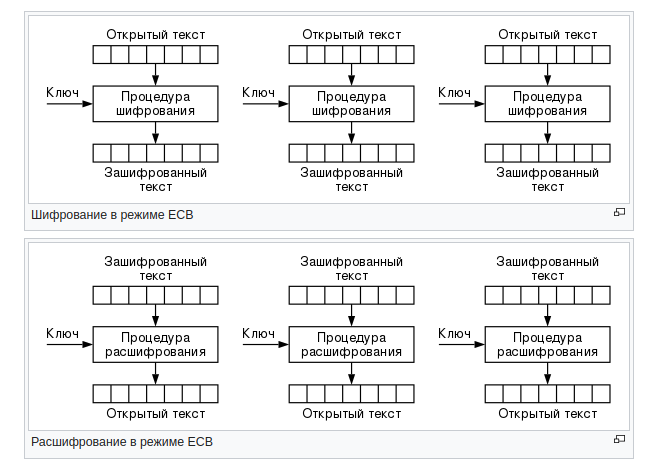
Код реализации криптографических примитивов представлен в Приложение Б

# **Режим кодовой книги**

Режим **электронной кодовой книги** ([англ.](https://ru.wikipedia.org/wiki/Английский_язык) *Electronic Codebook, ECB*) — один из вариантов использования симметричного [блочного шифра](https://ru.wikipedia.org/wiki/Блочный_шифр), при котором каждый блок [открытого текста](https://ru.wikipedia.org/wiki/Открытый_текст) заменяется блоком [шифротекста](https://ru.wikipedia.org/wiki/Шифротекст). В Г[ОСТ 28147—89](https://ru.wikipedia.org/wiki/ГОСТ_28147—89) называется **режимом простой замены**.

Шифрование может быть описано следующим образом:

где — номера блоков,и — блоки зашифрованного и открытого текстов соответственно, а— функция [блочного шифрования](https://ru.wikipedia.org/wiki/Блочное_шифрование). Расшифровка аналогична:

Данный алгоритм был реализован в программе, исходной код, которой вы можете видеть в **приложении В**

Программа в качестве входных данных принимает файл , количество циклов шифрования и ключ.

Далее с помощью криптографических примитивов описанных выше , программа шифрует файл.

Помимо реализации шифрования ,мною был реализован также процесс дешифрирования.

Программа реализована на языке Java

NIST тесты многораундового шифрования со сцеплением блоков и без:

|  |  |  |
| --- | --- | --- |
| file name | со сцеплением | без сцепления |
| dovlatov\_chemodan.doc | 16/16 | 4/16 |
| image1.jpg | 16/16 | 14/16 |
| image2.jpg | 16/16 | 10/16 |
| image3.jpg | 16/16 | 15/16 |
| image4.jpg | 15/16 | 14/16 |
| image5.jpg | 16/16 | 15/16 |
| Tolstoy\_Voina.txt | 16/16 | 9/16 |
| track1.wa | 16/16 | 13/16 |
| track2.wa | 16/16 | 8/16 |
| track3.wa | 16/16 | 10/16 |
| track4.wa | 15/16 | 12/16 |
| track5.wa | 15/16 | 8/16 |
| track6.fla | 16/16 | 15/16 |

По итогу сцепление блоков в разы улучшает результат шифрования. Тесты проводились на 9 раундах, последовательность: Простая подстановка->Гаммирование->Перестановка->Гаммирование->Простая подстановка->Гаммирование->Перестановка->Гаммирование->Простая подстановка.

# ПРИЛОЖЕНИЕ А

**КОД:**

Для вычисления биграмм и триграмм:

import nltk

from nltk import ngrams

from collections import Counter

text = ""

bigrams = ngrams(text, 2)

trigrams = ngrams(text, 3)

bigrams\_count = Counter(bigrams)

trigrams\_count = Counter(trigrams)

top\_10\_bigrams = bigrams\_count.most\_common(10)

top\_10\_trigrams = trigrams\_count.most\_common(10)

print("10 самых частых биграмм:")

for bigram, count in top\_10\_bigrams:

print(' '.join(bigram), count)

print("\n10 самых частых триграмм:")

for trigram, count in top\_10\_trigrams:

print(' '.join(trigram), count)

Для подстановки на основе ключа:

public static void main(String[] args) throws IOException {  
  
 *matrix* = new HashMap<>();  
 BufferedReader reader = new BufferedReader(new FileReader("matrix.txt"));  
 *symbols* = reader.readLine().split(";");  
 String[] s;  
 while (reader.ready()){  
 s = reader.readLine().split(";");  
 *matrix*.put(s[0], new HashMap<>());  
 for (int i = 0; i < *symbols*.length; i++) {  
 *matrix*.get(s[0]).put(*symbols*[i],s[i+1]);  
 }  
 }  
 reader.close();  
  
 Ini config = new Ini(new File("config.ini"));  
  
 String text = config.get("parameters", "text");  
 char[] key = config.get("parameters", "key").toCharArray();  
 StringJoiner itog = new StringJoiner("");  
 text = text.toUpperCase();  
 String now;  
 for (int i = 0; i < text.length(); i++) {  
 now = (""+key[i%4]).toUpperCase();  
 if (*matrix*.containsKey(now)){  
 for (String val:*matrix*.get(now).keySet()  
 ) {  
 if (*matrix*.get(now).get(val).equals(""+text.charAt(i))){  
 itog.add(val);  
 }  
 }  
 }  
 }  
  
 System.*out*.println((itog+"").replaceAll("Е/Ё", "Е"));  
  
}

Файл matrix.txt:

А;Б;В;Г;Д;Е/Ё;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;\_  
А;Б;В;Г;Д;Е/Ё;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;\_;А  
Б;В;Г;Д;Е/Ё;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;\_;А;Б  
В;Г;Д;Е/Ё;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;\_;А;Б;В  
Г;Д;Е/Ё;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;\_;А;Б;В;Г  
Д;Е/Ё;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;\_;А;Б;В;Г;Д  
Е/Ё;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;\_;А;Б;В;Г;Д;Е/Ё  
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;\_;А;Б;В;Г;Д;Е/Ё;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;\_;А;Б;В;Г;Д;Е/Ё;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;\_;А;Б;В;Г;Д;Е/Ё;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;\_;А;Б;В;Г;Д;Е/Ё;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;\_;А;Б;В;Г;Д;Е/Ё;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;\_;А;Б;В;Г;Д;Е/Ё;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;\_;А;Б;В;Г;Д;Е/Ё;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;\_;А;Б;В;Г;Д;Е/Ё;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;\_;А;Б;В;Г;Д;Е/Ё;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;\_;А;Б;В;Г;Д;Е/Ё;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;\_;А;Б;В;Г;Д;Е/Ё;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;\_;А;Б;В;Г;Д;Е/Ё;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;\_;А;Б;В;Г;Д;Е/Ё;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;\_;А;Б;В;Г;Д;Е/Ё;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;\_;А;Б;В;Г;Д;Е/Ё;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;\_;А;Б;В;Г;Д;Е/Ё;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;\_;А;Б;В;Г;Д;Е/Ё;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;\_;А;Б;В;Г;Д;Е/Ё;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;\_;А;Б;В;Г;Д;Е/Ё;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;\_;А;Б;В;Г;Д;Е/Ё;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;\_;А;Б;В;Г;Д;Е/Ё;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;\_;А;Б;В;Г;Д;Е/Ё;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;\_;А;Б;В;Г;Д;Е/Ё;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;\_;А;Б;В;Г;Д;Е/Ё;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;\_;А;Б;В;Г;Д;Е/Ё;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;\_;А;Б;В;Г;Д;Е/Ё;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  
\_;А;Б;В;Г;Д;Е/Ё;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;\_

# ПРИЛОЖЕНИЕ Б

**ПОДСТАНОВКА:**

Encoder:

public static void encodeSimpleSubstitution(String whatCodeFile, String toCodeFile, int[] asci){  
 try(FileInputStream whatCode = new FileInputStream(whatCodeFile);  
 FileOutputStream toCode = new FileOutputStream(toCodeFile)){  
 int bufferSize = 64000;  
 byte[] bufferRead = new byte[bufferSize];  
 byte[] bufferWrite = new byte[bufferSize];  
 while (whatCode.available() > 0){  
 if (whatCode.available() < bufferSize) bufferSize = whatCode.available();  
 whatCode.read(bufferRead, 0, bufferSize);  
 if (bufferSize != 64000) {  
 bufferWrite = new byte[bufferSize];  
 }  
 for (int i = 0; i < bufferSize; i++) {  
 bufferWrite[i] = (byte) asci[bufferRead[i] & 0b11111111];  
 //System.out.print(bufferWrite[i] + " ");  
 }  
  
 toCode.write(bufferWrite);  
 toCode.flush();  
 }  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
  
}

Decoder:

public static void decodeSimpleSubstitution(String whatDecodeFileName, String toCodeFileName, int[] asci){  
 try(FileInputStream whatDecodeFile = new FileInputStream(whatDecodeFileName);  
 FileOutputStream toCodeFile = new FileOutputStream(toCodeFileName)){  
 int[] asciDecode = new int[256];  
 for (int i = 0; i < 256; i++) {  
 asciDecode[asci[i]] = i;  
 }  
  
 int bufferSize = 64000;  
 byte[] bufferRead = new byte[bufferSize];  
 byte[] bufferWrite = new byte[bufferSize];  
 while (whatDecodeFile.available() > 0){  
 if (whatDecodeFile.available() < bufferSize) bufferSize = whatDecodeFile.available();  
 whatDecodeFile.read(bufferRead, 0, bufferSize);  
 if (bufferSize != 64000) {  
 bufferWrite = new byte[bufferSize];  
 }  
 for (int i = 0; i < bufferSize; i++) {  
 bufferWrite[i] = (byte)asciDecode[bufferRead[i] & 0b11111111];  
 }  
 toCodeFile.write(bufferWrite);  
 toCodeFile.flush();  
 }  
  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
}

**ПЕРЕСТАНОВКА**

Encoder:

public static int encodePermutation(String whatCodeFile, String toCodeFile, int[] key){  
 int whatCodeFileSize = 0;  
 try(FileInputStream whatCode = new FileInputStream(whatCodeFile);  
 FileOutputStream toCode = new FileOutputStream(toCodeFile)){  
 int bufferSize = 64000;  
 if (bufferSize % key.length != 0) bufferSize += bufferSize % key.length;  
 whatCodeFileSize = whatCode.available();  
  
 byte[] bufferRead = new byte[bufferSize];  
 byte[] bufferWrite = new byte[bufferSize];  
  
 while (whatCode.available() > 0){  
 if (whatCode.available() < bufferSize) {  
 bufferSize = whatCode.available();  
 bufferSize = bufferSize + key.length - (bufferSize % key.length);  
 bufferWrite = new byte[bufferSize];  
 }  
 whatCode.read(bufferRead, 0, bufferSize);  
  
 for (int i = 0; i < bufferSize / key.length; i++) {  
 for (int j = 0; j < key.length; j++) {  
 bufferWrite[i\*key.length + key[j]] = bufferRead[i\*key.length + j];  
 }  
 }  
 toCode.write(bufferWrite);  
 toCode.flush();  
 }  
  
 } catch (FileNotFoundException e) {  
 e.printStackTrace();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 return whatCodeFileSize;  
  
}

Decoder:

public static void decodePermutation(String whatDecodeFile, String toDecodeFile,  
 int[] key, int origFileSize){  
 int whatCodeFileSize = 0;  
 try(FileInputStream whatDecode = new FileInputStream(whatDecodeFile);  
 FileOutputStream toDecode = new FileOutputStream(toDecodeFile)){  
 int bufferSize = 64000 + key.length - (64000 % key.length);  
  
 int[] keyDecode = new int[key.length];  
 for (int i = 0; i < key.length; i++) {  
 keyDecode[key[i]] = i;  
 }  
 int diffOfSize = whatDecode.available() - origFileSize;  
  
 byte[] bufferRead = new byte[bufferSize];  
 byte[] bufferWrite = new byte[bufferSize];  
  
 while (whatDecode.available() > 0){  
 if (whatDecode.available() < bufferSize) {  
 bufferSize = whatDecode.available();  
 bufferWrite = new byte[bufferSize];  
 }  
 whatDecode.read(bufferRead, 0, bufferSize);  
  
 for (int i = 0; i < bufferSize / key.length; i++) {  
 for (int j = 0; j < key.length; j++) {  
 bufferWrite[i\*key.length + keyDecode[j]] = bufferRead[i\*key.length + j];  
 }  
 }  
 if (bufferSize < 64000){  
 byte[] lastBufferWrite = new byte[bufferSize-diffOfSize];  
 for (int i = 0; i < lastBufferWrite.length; i++) {  
 lastBufferWrite[i] = bufferWrite[i];  
 }  
 toDecode.write(lastBufferWrite);  
 toDecode.flush();  
 }else {  
 toDecode.write(bufferWrite);  
 toDecode.flush();  
 }  
 }  
  
 } catch (FileNotFoundException e) {  
 e.printStackTrace();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
  
}

**ГАММИРОВАНИЕ**

Encoder:

public static void encodeGamming(String whatCodeFileName, String toCodeFileName, int[] key){  
 try (FileInputStream whatCodeFile = new FileInputStream(whatCodeFileName);  
 FileOutputStream toCodeFile = new FileOutputStream(toCodeFileName)){  
  
 int bufferSize = 64000;  
 byte[] bufferRead = new byte[bufferSize];  
 byte[] bufferWrite = new byte[bufferSize];  
  
 while (whatCodeFile.available() > 0){  
 if (whatCodeFile.available() < bufferSize) {  
 bufferSize = whatCodeFile.available();  
 bufferWrite = new byte[bufferSize];  
 }  
  
 whatCodeFile.read(bufferRead, 0, bufferSize);  
  
 for (int i = 0; i < bufferSize; i++) {  
 bufferWrite[i] = (byte) (bufferRead[i] + key[i % key.length]);  
 }  
  
 toCodeFile.write(bufferWrite);  
 }  
  
 } catch (FileNotFoundException e) {  
 e.printStackTrace();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
}

Decoder:

public static void decodeGamming(String whatDecodeFileName, String toDecodeFileName, int[] key){  
 try (FileInputStream whatDecodeFile = new FileInputStream(whatDecodeFileName);  
 FileOutputStream toDecodeFile = new FileOutputStream(toDecodeFileName)){  
  
 int bufferSize = 64000;  
 byte[] bufferRead = new byte[bufferSize];  
 byte[] bufferWrite = new byte[bufferSize];  
  
 while (whatDecodeFile.available() > 0){  
 if (whatDecodeFile.available() < bufferSize) {  
 bufferSize = whatDecodeFile.available();  
 bufferWrite = new byte[bufferSize];  
 }  
  
 whatDecodeFile.read(bufferRead, 0, bufferSize);  
  
 for (int i = 0; i < bufferSize; i++) {  
 bufferWrite[i] = (byte) (bufferRead[i] - key[i % key.length]);  
 }  
  
 toDecodeFile.write(bufferWrite);  
 }  
  
 } catch (FileNotFoundException e) {  
 e.printStackTrace();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
}

# ПРИЛОЖЕНИЕ в

Основной класс:

public class MultilevelEncryption {  
 private int ending;  
 private CntOfRounds strategy;  
 private final int blockSize = 32;  
  
 private int[] permutationKey;  
 private int[] substitutionKey;  
 private int[] gummingKey;  
  
 private byte[] previousEncodeBlock;  
 private byte[] nowEncodeBlock;  
  
 public MultilevelEncryption(CntOfRounds strategy){  
 this.strategy = strategy;  
 }  
  
 public void encode(String whatCodeFileName, String toCodeFileName){  
 initKeys();  
  
 try(FileInputStream whatCodeFile = new FileInputStream(whatCodeFileName);  
 FileOutputStream toCodeFile = new FileOutputStream(toCodeFileName)){  
  
 int bufferSize = 64000;  
 byte[] bufferRead = new byte[bufferSize];  
 byte[] bufferWrite = new byte[bufferSize];  
 byte[] block;  
  
 while(whatCodeFile.available() > 0){  
 if (whatCodeFile.available() < bufferSize){  
 bufferSize = whatCodeFile.available();  
 ending = blockSize - (bufferSize % blockSize);  
 bufferSize += ending;  
 bufferWrite = new byte[bufferSize];  
 }  
 whatCodeFile.read(bufferRead, 0, bufferSize);  
  
 for (int i = blockSize - 1; i < bufferSize; i+=blockSize) {  
 if (previousEncodeBlock == null){  
 block = choseEncodeMethod(  
 Arrays.*copyOfRange*(bufferRead, i-blockSize+1, i+1));  
 previousEncodeBlock = block;  
 }else{  
 block = Arrays.*copyOfRange*(bufferRead, i-blockSize+1, i+1);  
 block = xorBlocks(block, previousEncodeBlock);  
 block = choseEncodeMethod(block);  
 previousEncodeBlock = block;  
 }  
 System.*arraycopy*(block, 0, bufferWrite, i-blockSize+1, blockSize);  
 }  
  
 toCodeFile.write(bufferWrite);  
 toCodeFile.flush();  
 }  
 } catch (FileNotFoundException e) {  
 e.printStackTrace();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 private byte[] choseEncodeMethod(byte[] block){  
 byte[] resultBlock = new byte[blockSize];  
 switch (strategy){  
 case *FIVE\_PE*:  
 resultBlock = encodeFiveRndPe(block);  
 break;  
 case *FIVE\_SU*:  
 resultBlock = encodeFiveRndSu(block);  
 break;  
 case *NINE\_PE*:  
 resultBlock = encodeNineRndPe(block);  
 break;  
 case *NINE\_SU*:  
 resultBlock = encodeNineRndSu(block);  
 break;  
 }  
 return resultBlock;  
 }  
  
 public void decode(String whatDecodeFileName, String toDecodeFileName){  
 previousEncodeBlock = null;  
  
 try(FileInputStream whatDecodeFile = new FileInputStream(whatDecodeFileName);  
 FileOutputStream toDecodeFile = new FileOutputStream(toDecodeFileName)){  
  
 int bufferSize = 64000;  
 byte[] bufferRead = new byte[bufferSize];  
 byte[] bufferWrite = new byte[bufferSize];  
 byte[] block;  
 boolean flag = true;  
  
 while(whatDecodeFile.available() > 0){  
 if (whatDecodeFile.available() < bufferSize){  
 bufferSize = whatDecodeFile.available();  
 bufferWrite = new byte[bufferSize];  
 flag = false;  
 }  
 whatDecodeFile.read(bufferRead, 0, bufferSize);  
  
 for (int i = blockSize - 1; i < bufferSize; i+=blockSize) {  
 if (previousEncodeBlock == null){  
 previousEncodeBlock = Arrays.*copyOfRange*(bufferRead, i-blockSize+1, i+1);  
 block = choseDecodeMethod(  
 previousEncodeBlock);  
 }else{  
 nowEncodeBlock = Arrays.*copyOfRange*(bufferRead, i-blockSize+1, i+1);  
 block = choseDecodeMethod(nowEncodeBlock);  
 block = xorBlocks(block, previousEncodeBlock);  
 previousEncodeBlock = nowEncodeBlock;  
 }  
 System.*arraycopy*(block, 0, bufferWrite, i-blockSize+1, blockSize);  
 }  
 if (!flag) {  
 toDecodeFile.write(  
 Arrays.*copyOfRange*(bufferWrite, 0, bufferSize - ending+1)  
 );  
 toDecodeFile.flush();  
 }else {  
 toDecodeFile.write(bufferWrite);  
 toDecodeFile.flush();  
 }  
 }  
 } catch (FileNotFoundException e) {  
 e.printStackTrace();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 private byte[] choseDecodeMethod(byte[] block){  
 byte[] resultBlock = new byte[blockSize];  
 switch (strategy){  
 case *FIVE\_PE*:  
 resultBlock = decodeFiveRndPe(block);  
 break;  
 case *FIVE\_SU*:  
 resultBlock = decodeFiveRndSu(block);  
 break;  
 case *NINE\_PE*:  
 resultBlock = decodeNineRndPe(block);  
 break;  
 case *NINE\_SU*:  
 resultBlock = decodeNineRndSu(block);  
 break;  
 }  
 return resultBlock;  
 }  
  
 private byte[] xorBlocks(byte[] firstBlock, byte[] secondBlock){  
 byte[] result = new byte[firstBlock.length];  
 for (int i = 0; i < result.length; i++) {  
 result[i] = (byte) (firstBlock[i] ^ secondBlock[i]);  
 }  
  
 return result;  
 }  
  
 private byte[] encodeFiveRndPe(byte[] block){  
 block = Encoder.*encodePermutation*(block, permutationKey);  
 block = Encoder.*encodeGamming*(block, gummingKey);  
 block = Encoder.*encodeSimpleSubstitution*(block, substitutionKey);  
 block = Encoder.*encodeGamming*(block, gummingKey);  
 block = Encoder.*encodePermutation*(block, permutationKey);  
 return block;  
 }  
 private byte[] encodeFiveRndSu(byte[] block){  
 block = Encoder.*encodeSimpleSubstitution*(block, substitutionKey);  
 block = Encoder.*encodeGamming*(block, gummingKey);  
 block = Encoder.*encodePermutation*(block, permutationKey);  
 block = Encoder.*encodeGamming*(block, gummingKey);  
 block = Encoder.*encodeSimpleSubstitution*(block, substitutionKey);  
 return block;  
 }  
 private byte[] encodeNineRndPe(byte[] block){  
 block = Encoder.*encodePermutation*(block, permutationKey);  
 block = Encoder.*encodeGamming*(block, gummingKey);  
 block = Encoder.*encodeSimpleSubstitution*(block, substitutionKey);  
 block = Encoder.*encodeGamming*(block, gummingKey);  
 block = Encoder.*encodePermutation*(block, permutationKey);  
 block = Encoder.*encodeGamming*(block, gummingKey);  
 block = Encoder.*encodeSimpleSubstitution*(block, substitutionKey);  
 block = Encoder.*encodeGamming*(block, gummingKey);  
 block = Encoder.*encodePermutation*(block, permutationKey);  
 return block;  
 }  
 private byte[] encodeNineRndSu(byte[] block){  
 block = Encoder.*encodeSimpleSubstitution*(block, substitutionKey);  
 block = Encoder.*encodeGamming*(block, gummingKey);  
 block = Encoder.*encodePermutation*(block, permutationKey);  
 block = Encoder.*encodeGamming*(block, gummingKey);  
 block = Encoder.*encodeSimpleSubstitution*(block, substitutionKey);  
 block = Encoder.*encodeGamming*(block, gummingKey);  
 block = Encoder.*encodePermutation*(block, permutationKey);  
 block = Encoder.*encodeGamming*(block, gummingKey);  
 block = Encoder.*encodeSimpleSubstitution*(block, substitutionKey);  
 return block;  
 }  
  
 private byte[] decodeFiveRndPe(byte[] block){  
 block = Decoder.*decodePermutation*(block, permutationKey);  
 block = Decoder.*decodeGamming*(block, gummingKey);  
 block = Decoder.*decodeSimpleSubstitution*(block, substitutionKey);  
 block = Decoder.*decodeGamming*(block, gummingKey);  
 block = Decoder.*decodePermutation*(block, permutationKey);  
 return block;  
 };  
 private byte[] decodeFiveRndSu(byte[] block){  
 block = Decoder.*decodeSimpleSubstitution*(block, substitutionKey);  
 block = Decoder.*decodeGamming*(block, gummingKey);  
 block = Decoder.*decodePermutation*(block, permutationKey);  
 block = Decoder.*decodeGamming*(block, gummingKey);  
 block = Decoder.*decodeSimpleSubstitution*(block, substitutionKey);  
 return block;  
 };  
 private byte[] decodeNineRndPe(byte[] block){  
 block = Decoder.*decodePermutation*(block, permutationKey);  
 block = Decoder.*decodeGamming*(block, gummingKey);  
 block = Decoder.*decodeSimpleSubstitution*(block, substitutionKey);  
 block = Decoder.*decodeGamming*(block, gummingKey);  
 block = Decoder.*decodePermutation*(block, permutationKey);  
 block = Decoder.*decodeGamming*(block, gummingKey);  
 block = Decoder.*decodeSimpleSubstitution*(block, substitutionKey);  
 block = Decoder.*decodeGamming*(block, gummingKey);  
 block = Decoder.*decodePermutation*(block, permutationKey);  
 return block;  
 };  
 private byte[] decodeNineRndSu(byte[] block){  
 block = Decoder.*decodeSimpleSubstitution*(block, substitutionKey);  
 block = Decoder.*decodeGamming*(block, gummingKey);  
 block = Decoder.*decodePermutation*(block, permutationKey);  
 block = Decoder.*decodeGamming*(block, gummingKey);  
 block = Decoder.*decodeSimpleSubstitution*(block, substitutionKey);  
 block = Decoder.*decodeGamming*(block, gummingKey);  
 block = Decoder.*decodePermutation*(block, permutationKey);  
 block = Decoder.*decodeGamming*(block, gummingKey);  
 block = Decoder.*decodeSimpleSubstitution*(block, substitutionKey);  
 return block;  
 };  
  
 private void initKeys(){  
 substitutionKey = Encoder.*mixArray*(256);  
 permutationKey = Encoder.*mixArray*(blockSize);  
 gummingKey = Encoder.*rndArrayOfByte*(blockSize);  
 }  
}

Класс стратегий:

public enum CntOfRounds {  
 //permutation->gamming->simpleSubstitution->gamming->permutation  
 *FIVE\_PE*,  
 //simpleSubstitution->gamming->permutation->gamming->simpleSubstitution  
 *FIVE\_SU*,  
 //permutation->gamming->simpleSubstitution->gamming  
 //->permutation->gamming->simpleSubstitution->gamming->permutation  
 *NINE\_PE*,  
 //simpleSubstitution->gamming->permutation->gamming  
 //->simpleSubstitution->gamming->permutation->gamming->simpleSubstitution  
 *NINE\_SU*;  
}

Пример вызова:

public class Main {  
 public static void main(String[] args) {  
 MultilevelEncryption encryption =  
 new MultilevelEncryption(CntOfRounds.*NINE\_SU*);  
 encryption.encode("firstFile.txt", "secondFile.txt");  
 encryption.decode("second.exe", "third.exe");  
 }  
}

# ЛИТЕРАТУРА

1. <https://ru.wikipedia.org/wiki/Криптографические_примитивы>
2. <https://ru.wikipedia.org/wiki/Блочный_шифр>
3. <https://ru.wikipedia.org/wiki/Индекс_совпадений>
4. <https://habr.com/ru/company/yandex/blog/324866/>
5. <https://python-scripts.com/encryption-cryptography>
6. <https://ru.wikipedia.org/wiki/Метод_Касиски>
7. <https://studfile.net/preview/6211059/page:12/>