Федеральное государственное бюджетное образовательное учреждение высшего образования
«ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ ДВАЖДЫ ГЕРОЯ СОВЕТСКОГО СОЮЗА, ЛЕТЧИКА-КОСМОНАВТА А.А. ЛЕОНОВА»

ИНСТИТУТ ИНФОКОММУНИКАЦИОННЫХ СИСТЕМ И ТЕХНОЛОГИЙ

КАФЕДРА ИНФОРМАЦИОННОЙ БЕЗОПАСНОСТИ

ОТЧЁТ ПО УЧЕБНО-ЛАБОРАТОРНОЙ ПРАКТИКЕ

Выполнил:
студент группы ИБО-ТС-22
Титов К.А.

Проверил:
заведующий кафедрой ИБ
Сазонов С.Ю.

Королёв 2024 г.

Содержание

1. Лабораторная работа №1. КРИПТОАНАЛИЗ ШИФРА ПРОСТОЙ ЗАМЕНЫ

Цель работы: Выполнить криптоанализ текста, зашифрованного шифром простой замены с применением неперебoрного метода вскрытия шифротекста.

Ход работы:

1.1. Исходный код программы

Ниже приведён код программы по шифрованию и дешифрованию на основе шифра простой замены на языке программирования Python 3.10 (листинг 1). Программа позволяет вводить данные с клавиатуры и файлов (тестовые данные - листинг 2).

```
import random
import sys
from typing import Optional, Tuple, Union

RU_ALPHABET = "абвгдеёжзийклмнопрстуфхцчшщъыьэюя"
ENCRYPT_ACTION = "--enc"
DECRYPT_ACTION = "--dec"


# Считывает текст из файла
def read_from_file_enc(filename: str) -> Optional[str]:
  try:
    with open(filename, "r") as f:
      buf = f.read().split('\n')[0].lower()
      return buf
  except Exception as e:
    print(f'Ошибка: {str(e)}')
    return None

def read_from_file_dec(filename: str) -> Optional[Tuple[str, str]]:
  try:
    with open(filename, "r") as f:
      buf = f.read().split('\n')
      s, k = buf[0], buf[1]
      return (s, k)
  except Exception as e:
    print(f'Ошибка: {str(e)}')
    return None


# Считывает текст из терминала
def read_from_console_enc() -> str:
  print("Введите текст (на русском языке), который требуется зашифровать/
```

```
расшифровать:")
  return input().lower()

def read_from_console_dec() -> Tuple[str, str]:
  print("Введите текст (на русском языке), который требуется зашифровать/
расшифровать:")
  s = input().lower()
  k = input("Введите ключ шифрования: ")
  return (s, k)


# Шифрует текст предоставленным ключом
def encrypt(text: str) -> Union[str, str]:
  key = []
  ru_sample = list(RU_ALPHABET)
  while len(ru_sample) != 0:
    letter = random.choice(ru_sample)
    key.append(letter)
    ru_sample.remove(letter)
  key = ''.join(key)

  encrypted = []
  for symbol in text:
    pos = RU_ALPHABET.find(symbol)
    if pos == -1:
      encrypted.append(symbol)
      continue
    new_sym = key[pos]
    encrypted.append(new_sym)
  return ''.join(encrypted), key


# Расшифровывает текст предоставленным ключом
def decrypt(encrypted: str, key: str) -> str:
  decrypted = []
  for symbol in encrypted:
    pos = key.find(symbol)
    if pos == -1:
      decrypted.append(symbol)
      continue
    old_sym = RU_ALPHABET[pos]
    decrypted.append(old_sym)
  return ''.join(decrypted)


# Уточняет, что нужно сделать
def decide_action() -> str:
  def print_prompt() -> None:
    print('Выберите действие: ')
```

```
    print('1. Зашифровать текст с помощью ключа')
    print('2. Расшифровать текст с помощью ключа')

  def read_opt(allowed: list) -> Optional[int]:
    try:
      opt = int(input("Введите номер [1-2]: "))
      if opt not in allowed:
        print(f'Ошибка: опция #{opt} недоступна!')
        return None
      return opt
    except Exception as e:
      print(f'Ошибка: {str(e)}')
      return None

  print_prompt()
  opt = read_opt([1, 2])
  while opt is None:
    print_prompt()
    opt = read_opt([1, 2])

  match opt:
    case 1:
      return ENCRYPT_ACTION
    case 2:
      return DECRYPT_ACTION
    case _:
      return None


# Читает в программу текст и ключ
def read(filename: Optional[str], action: str) -> Optional[Union[Tuple[str, str],
str]]:
  if filename is None:
    if action == ENCRYPT_ACTION:
      return read_from_console_enc()
    else:
      return read_from_console_dec()
  else:
    if action == ENCRYPT_ACTION:
      return read_from_file_enc(filename)
    else:
      return read_from_file_dec(filename)


# Начало программы
if __name__ == '__main__':
  # Если программу запустить с аргументами (именем файла с текстом и ключом), ввод
будет считан из файла
  filename = None
```

```
action = None
for arg in sys.argv[1:]:
  if arg == ENCRYPT_ACTION:
    action = ENCRYPT_ACTION
  elif arg == DECRYPT_ACTION:
    action = DECRYPT_ACTION
  else:
    filename = arg

if action is None:
  action = decide_action()

if action == ENCRYPT_ACTION:
  s = read(filename, action)
  encrypted, key = encrypt(s)
  print(f'Результат: "{encrypted}"')
  print(f'Ключ: "{key}"')
elif action == DECRYPT_ACTION:
  s, k = read(filename, action)
  decrypted = decrypt(s, k)
  print(f'Результат: "{decrypted}"')
```

Листинг 1. Файл lab_01.py

```
Безумие - это когда ты делаешь одно и то же, ожидая совсем другого результата.
```

Листинг 2. Файл lab_01_encrypt.txt

1.2. Частотный анализ шифрованного текста

Для проведения работы был зашифрован следующий текст:

Князь Багратион, выехав на самый высокий пункт нашего правого фланга, стал спускаться книзу, где слышалась перекатная стрельба и ничего не видно было от порохового дыма. Чем ближе они спускались к лощине, тем менее им становилось видно, но тем чувствительнее становилась близость самого настоящего поля сражения. Им стали встречаться раненые. Одного, с окровавленною головой, без шапки, тащили двое солдат под руки. Он хрипел и плевал. Пуля попала, видно, в рот или в горло. Другой, встретившийся им, бодро шел один, без ружья, громко охая и махая от свежей боли рукою, из которой кровь лилась, как из склянки, на его шинель. Лицо его казалось больше испуганным, чем страдающим. Он минуту тому назад был ранен. Переехав дорогу, они стали круто спускаться и на спуске увидели несколько человек, которые лежали; им встретилась толпа солдат, в числе которых были и раненые. Солдаты шли в гору, тяжело дыша, и, несмотря на вид генерала, громко разговаривали и махали руками. Впереди, в дыму, уже были видны ряды серых шинелей, и офицер, увидав Багратиона, с криком побежал за солдатами, шедшими толпой, требуя, чтоб они воротились. Багратион подъехал к рядам, по которым то там, то здесь быстро щелкали выстрелы, заглушая говор и командные крики. Весь воздух пропитан был пороховым дымом. Лица солдат все были закопчены порохом и оживлены. Иные забивали шомполами, другие подсыпали на полки, доставали заряды из сумок, третьи стреляли. Но в кого они стреляли, этого

не было видно от порохового дыма, не уносимого ветром. Довольно часто слышались приятные звуки жужжанья и свистения. Что это такое? – думал князь Андрей, подъезжая к этой толпе солдат. – Это не может быть цепь, потому что они в куче! Не может быть атака, потому что они не двигаются; не может быть каре: они не так стоят.

Листинг 3. Файл lab_01_text.txt

Получен был следующий результат:

```
хйлят ымъгмрдвй, фжэзмф йм пмчжо фжпвхдо ацйхр ймеэъв агмфвъв нимйъм, прми пацпхмртпл хйдяц, ъсэ пижемимпт аэгэхмрймл пргэитым д йдьэъв йэ фдсйв ыжив вр авгвзвфвъв сжчм. ьэч ыидёэ вйд пацпхмидпт х ивудйэ, рэч чэйээ дч прмйвфдивпт фдсйв, йв рэч ьцфпрфдрэитйээ прмйвфдимпт ыидявпрт пмчвъв ймпрвлуэъв авил пгмёэйдл. дч прмид фпргэьмртпл гмйэйжэ. всйвъв, п вхгвфмфиэййвш ъвивфво, ыэя емахд, рмудид сфвэ пвисмр авс гцхд. вй згдаэи д аиэфми. ацил авамим, фдсйв, ф гвр дид ф ъвгив. сгцъво, фпргэрдфедопл дч, ывсгв еэи всдй, ыэя гцётл, ъгвчхв взмл д чмзмл вр пфэёэо ывид гцхвш, дя хврвгво хгвфт идимпт, хмх дя пхилйхд, йм эъв едйэит. идбв эъв хмямивпт ывитеэ дпацъмййжч, ьэч пргмсмшудч. вй чдйцрц рвчц ймямс ыжи гмйэй. аэгээзмф свгвъц, вйд прмид хгцрв пацпхмртпл д йм пацпхэ цфдсэид йэпхвитхв ьэивфэх, хврвгжэ иэёмид; дч фпргэрдимпт рвиам пвисмр, ф ьдпиэ хврвгжз ыжид д гмйэйжэ. пвисмрж еид ф ъвгц, рлёэив сжем, д, йэпчвргл йм фдс ъэйэгмим, ъгвчхв гмяъвфмгдфмид д чмзмид гцхмчд. фаэгэсд, ф сжчц, цёэ ыжид фдсйж глсж пэгжз едйэиэо, д вндбэг, цфдсмф ымъгмрдвйм, п хгдхвч авыэёми ям пвисмрмчд, еэседчд рвиаво, ргэыцл, ьрвы вйд фвгврдидпт. ымъгмрдвй авсюэзми х глсмч, ав хврвгжч рв рмч, рв ясэпт ыжпргв уэихмид фжпргэиж, ямъицемл ъвфвг д хвчмйсйжэ хгдхд. фэпт фвясцз агвадрмй ыжи авгвзвфжч сжчвч. идбм пвисмр фпэ ыжид ямхваьэйж авгвзвч д вёдфиэйж. дйжэ ямыдфмид евчавимчд, сгцъдэ авспжамид йм авихд, свпрмфмид ямглсж дя пцчвх, ргэртд пргэилид. йв ф хвъв вйд пргэилид, крвъв йэ ыжив фдсйв вр авгвзвфвъв сжчм, йэ цйвпдчвъв фэргвч. свфвитйв ьмпрв пижемидпт агдлрйжэ яфцхд ёцёёмйтл д пфдпрэйдл. ьрв крв рмхвэ? – сцчми хйлят мйсгэо, авсюэяёмл х крво рвиаэ пвисмр. – крв йэ чвёэр ыжрт бэат, аврвчц ьрв вйд ф хцьэ! йэ чвёэр ыжрт мрмхм, аврвчц ьрв вйд йэ сфдъмшрпл; йэ чвёэр ыжрт хмгэ: вйд йэ рмх првлр.
```

Листинг 4. Зашифрованный текст

Проведём частотный анализ при помощи следующей программы:

```
from collections import Counter
import lab_01

s = lab_01.read_from_file_enc('lab_01_text.txt')
encrypted, key = lab_01.encrypt(s)

print(f'Результат: "{encrypted}"')
print(f'Ключ: "{key}"')

ru_letters = ""
for symbol in encrypted:
  if symbol in lab_01.RU_ALPHABET:
    ru_letters += symbol

cntr = Counter(ru_letters)
```

```
size = len(ru_letters)

print('Частотный анализ:')
for ch, freq in cntr.most_common():
  p = freq / size
  print(f'"{ch}":\t{p}')
```

Листинг 5. Программа криптоанализа lab_01_freq_analysis.py

Результаты:

```
"в":    0.11818181818181818
"м":    0.07832167832167833
"д":    0.07832167832167833
"э":    0.07132867132867132
"р":    0.06153846153846154
"и":    0.05944055944055944
"й":    0.05524475524475524
"п":    0.04965034965034965
"г":    0.045454545454545456
"ф":    0.03986013986013986
"х":    0.03496503496503497
"ч":    0.032167832167832165
"с":    0.032167832167832165
"ж":    0.028671328671328673
"а":    0.027972027972027972
"ц":    0.025174825174825177
"ъ":    0.022377622377622378
"т":    0.021678321678321677
"л":    0.02097902097902098
"ы":    0.01818181818181818
"я":    0.014685314685314685
"ё":    0.011188811188811189
"е":    0.01048951048951049
"з":    0.009790209790209791
"ь":    0.009790209790209791
"о":    0.007692307692307693
"у":    0.0034965034965034965
"ш":    0.002797202797202797
"б":    0.002797202797202797
"к":    0.002797202797202797
"н":    0.0013986013986013986
"ю":    0.0013986013986013986
```

Видно, что букве «в» шифротекста соответствует наивысшая частота, что похоже на букву «о». У букв «м», «д», «э» также похожи частоты на буквы «е», «ф» и «и». Также вероятно, что одна из них - это буква «а».

Приступим к подбору мелких частиц и предлогов. Буква «д» достаточно часто встречается в качестве предлога или союза, поэтому резонно предположить, что это буква «и» оригинального текста.

Часто встречается словесная единица «дч», причём

$$p(\text{'ч'}) \approx p(\text{'л'} \mid \text{'к'} \mid \text{'м'})$$

Предположим, что это местоимение «им».

Частоты букв «рэч» позволяют предположить, что это слово «там». Однако это не проходит проверку словом «чэйээ», поэтому делаем вывод, что «э» соответствует «е». Отсюда «чэйээ» - это «менее», тогда «й» - это «н».

Достаточно часто встречается слово «крв», и поскольку «в» предположительно равно «о», «крв» - это «это».

Расшифруем первое большое слово: «йэпхвитхв» («не__о___о» ≈ «несколько»). На второе - «ымъгмрдвй» («_____ион») - пока не хватает смысла.

Как насчёт «пмчжо»? «с_м__». «всйвъв» - «о_но_о», причём «с» ($p \approx 0.032$) похоже на «д» ($p \approx 0.025$) и ещё не встречалось. Тогда «всйвъв» - это «одного».

Отсюда «ымъгмрдвй» - это «__г___ион». Всё ещё не хватает.

Берём словосочетание «дч фпргэрдимпт рвиам пвисмр»: «им _ст_е_ил_сь тол__ солд_т». Отсюда «м» - это «а», «а» - «п», и словосочетание расшифровывается как «им встретилась толпа солдат».

«ымъгмрдвй» - «_агратион», т.е. - Багратион; «ы» - это «б».

Возьмём достаточно большой кусок текста: «фаэгэсд, ф сжчц, цёэ ыжид фдсйж глсж пэгжз едйэиэо, д вндбэг» - «впереди, в дыму, уже были видны ряды серых шинелей».

Итоговый ключ дешифровки: «мыфъсэщёядохичйвагпрцнзбьеующкшл».

Таким образом, благодаря частотному анализу была значительно ускорена дешифровка текста без необходимости полного перебора.

1.3. Контрольный вопрос

Что понимается под избыточностью сообщения?

Ответ: Избыточность сообщения — это наличие в сообщении дополнительной информации, которая не является необходимой для передачи его основного смысла. Она часто используется для повышения надежности передачи данных, для предотвращения потери информации из-за шума или ошибок, а также для обеспечения ясности и понимания.

2. Лабораторная работа №2. ШИФРЫ ПЕРЕСТАНОВКИ НА ПРИМЕРЕ ШИФРА КАРДАНО

Цель работы: Разработать алгоритмы шифрования и дешифрования сообщений с применением решётки Кардано.

Ход работы:

2.1. Исходный код программы

Ниже приведён код программы по шифрованию и дешифрованию на основе шифра Кардано на языке программирования Python 3.10 (листинг 6). Программа позволяет вводить данные с клавиатуры и файлов (тестовые данные - листинг 7, листинг 8).

```
import random
import sys
from typing import Optional, Tuple, Union

ENCRYPT_ACTION = "--enc"
DECRYPT_ACTION = "--dec"
```

```
# Считывает текст из файла
def read_from_file_enc(filename: str) -> Optional[str]:
  try:
    with open(filename, "r") as f:
      buf = f.read().split('\n')[0]
      return buf
  except Exception as e:
    print(f'Ошибка: {str(e)}')
    return None

def read_from_file_dec(filename: str) -> Optional[Tuple[str, str]]:
  try:
    with open(filename, "r") as f:
      buf = f.read().split('\n')
      s, k = buf[0], buf[1]
      return (s, k)
  except Exception as e:
    print(f'Ошибка: {str(e)}')
    return None


# Считывает текст из терминала
def read_from_console_enc() -> str:
  print("Введите текст (на русском языке), который требуется зашифровать/
расшифровать:")
  return input().lower()

def read_from_console_dec() -> Tuple[str, str]:
  print("Введите текст (на русском языке), который требуется зашифровать/
расшифровать:")
  s = input().lower()
  k = input("Введите ключ шифрования: ")
  return (s, k)


# Создаёт ключ и шифрует текст с его помощью
def encrypt(text: str) -> Tuple[str, str]:
  square_len = find_nearest_bigger_square(len(text))
  padded_text = text.ljust(square_len ** 2)
  key = make_cardano_lattice_key(square_len)
  encrypted = []
  for si in key:
    encrypted.append(padded_text[si])
  for si in rotate_key_90(key, square_len):
    encrypted.append(padded_text[si])
  for si in rotate_key_180(key, square_len):
    encrypted.append(padded_text[si])
  for si in rotate_key_270(key, square_len):
```

```
    encrypted.append(padded_text[si])
  return ''.join(encrypted), encode_key(key)


# Расшифровывает текст с помощью ключа
def decrypt(encrypted: str, key: str) -> str:
  size = int(len(encrypted) ** 0.5)
  key = decode_key(key)
  decrypted = [' '] * (size ** 2)

  quarter = len(encrypted) // 4
  rotations = [
    key,
    rotate_key_90(key, size),
    rotate_key_180(key, size),
    rotate_key_270(key, size)
  ]

  for i, rotation in enumerate(rotations):
    for j, pos in enumerate(rotation):
      decrypted[pos] = encrypted[i * quarter + j]

  return ''.join(decrypted)


# Ищет ближайший больший квадрат
def find_nearest_bigger_square(text_size: int) -> int:
  i = 1
  while i ** 2 < text_size:
    i += 1
  return i


# Кодирует ключ
def encode_key(key: list[int]) -> str:
  return '-'.join(map(str, key))
# Декодирует ключ
def decode_key(key: str) -> list[int]:
  return list(map(int, key.split('-')))


# Поворот на 90 градусов
def rotate_90(row: int, col: int, n: int) -> Tuple[int, int]:
  return col, n - 1 - row
# Поворот на 180 градусов
def rotate_180(row: int, col: int, n: int) -> Tuple[int, int]:
    return n - 1 - row, n - 1 - col
# Поворот на 270 градусов
def rotate_270(row: int, col: int, n: int) -> Tuple[int, int]:
```

```
    return n - 1 - col, row
# Вычисление позиции ячейки матрицы в решётке
def calc_pos(row: int, col: int, n: int) -> int:
  return row * n + col
# Вычисление позиций ячейки из решётки
def calc_pos2(pos: int, n: int) -> Tuple[int, int]:
  return pos // n, pos % n


# Вращение ключа
def rotate_key_90(key: list[int], side_len: int) -> list[int]:
  return [calc_pos(*rotate_90(*calc_pos2(i, side_len), side_len), side_len) for i
in key]
def rotate_key_180(key: list[int], side_len: int) -> list[int]:
  return [calc_pos(*rotate_180(*calc_pos2(i, side_len), side_len), side_len) for i
in key]
def rotate_key_270(key: list[int], side_len: int) -> list[int]:
  return [calc_pos(*rotate_270(*calc_pos2(i, side_len), side_len), side_len) for i
in key]


# Ищет перемещения окна при поворотах решётки
def find_opposites(side_len: int, window: int) -> list[int]:
  opposites = [window]
  row, col = calc_pos2(window, side_len)
  # Ищем три дополнительных поворота
  w1 = calc_pos(*rotate_90(row, col, side_len), side_len)
  w2 = calc_pos(*rotate_180(row, col, side_len), side_len)
  w3 = calc_pos(*rotate_270(row, col, side_len), side_len)
  # Если повороты одинаковые, нет смысла их добавлять.
  # Это случается, когда размер квадрата нечётный, и выбранное окно - центр
квадрата.
  if window != w1:
    opposites += [w1, w2, w3]
  return opposites


# Создаёт решётку Кардано
def make_cardano_lattice_key(side_len: int) -> list[int]:
  lattice = [i for i in range(side_len ** 2)]
  key = []
  while len(lattice) != 0:
    window = random.choice(lattice)
    key.append(window)
    windows = find_opposites(side_len, window)
    for w in windows:
      lattice.remove(w)
  return key
```

```
# Уточняет, что нужно сделать
def decide_action() -> str:
  def print_prompt() -> None:
    print('Выберите действие: ')
    print('1. Зашифровать текст с помощью ключа')
    print('2. Расшифровать текст с помощью ключа')

  def read_opt(allowed: list) -> Optional[int]:
    try:
      opt = int(input("Введите номер [1-2]: "))
      if opt not in allowed:
        print(f'Ошибка: опция #{opt} недоступна!')
        return None
      return opt
    except Exception as e:
      print(f'Ошибка: {str(e)}')
      return None

  print_prompt()
  opt = read_opt([1, 2])
  while opt is None:
    print_prompt()
    opt = read_opt([1, 2])

  match opt:
    case 1:
      return ENCRYPT_ACTION
    case 2:
      return DECRYPT_ACTION
    case _:
      return None


# Читает в программу текст и ключ
def read(filename: Optional[str], action: str) -> Optional[Union[Tuple[str, str],
str]]:
  if filename is None:
    if action == ENCRYPT_ACTION:
      return read_from_console_enc()
    else:
      return read_from_console_dec()
  else:
    if action == ENCRYPT_ACTION:
      return read_from_file_enc(filename)
    else:
      return read_from_file_dec(filename)
```

```
# Начало программы
if __name__ == '__main__':
  # Если программу запустить с аргументами (именем файла с текстом и ключом), ввод
будет считан из файла
  filename = None
  action = None
  for arg in sys.argv[1:]:
    if arg == ENCRYPT_ACTION:
      action = ENCRYPT_ACTION
    elif arg == DECRYPT_ACTION:
      action = DECRYPT_ACTION
    else:
      filename = arg

  if action is None:
    action = decide_action()

  if action == ENCRYPT_ACTION:
    s = read(filename, action)
    encrypted, key = encrypt(s)
    print(f'Результат: "{encrypted}"')
    print(f'Ключ: "{key}"')
  elif action == DECRYPT_ACTION:
    s, k = read(filename, action)
    decrypted = decrypt(s, k)
    print(f'Результат: "{decrypted}"')
```

Листинг 6. Файл lab_02.py

```
Безумие - это когда ты делаешь одно и то же, ожидая совсем другого результата.
```

Листинг 7. Файл lab_02_encrypt.txt

```
аб. игети  зьикзоо е а-о оэт мдесдысажуож е у,тге  я т н  аоео гь реурладтлмдшво до
74-0-77-37-5-64-56-73-36-30-22-69-29-47-14-2-31-15-79-68-40
```

Листинг 8. Файл lab_02_decrypt.txt

2.2. Шифрование и дешифрование текста

Выберем достаточно большой кусок текста: «Уже близко становились французы; уже князь Андрей, шедший рядом с Багратионом, ясно различал перевязи, красные эполеты, даже лица французов. (Он ясно видел одного старого французского офицера, который вывернутыми ногами в штиблетах, придерживаясь за кусты, с трудом шел в гору.) Князь Багратион не давал нового приказания и все так же молча шел перед рядами. Вдруг между французами треснул один выстрел, другой, третий… и по всем расстроившимся неприятельским рядам разнесся дым и затрещала пальба. Несколько человек наших упало, в том числе и круглолицый офицер, шедший так весело и старательно. Но в то

же мгновение, как раздался первый выстрел, Багратион оглянулся и закричал: Ура!»

Зашифруем его с помощью команды:

```
$ ./lab_02.py --enc lab-02-text.txt
Результат: ".н аажрг а,фсьрти еиваянраинс Взее тсрд  ц врпраннср ггясе рб ов уваиоатрА
слрл рглр м  нмиешзык  о и да:ло  йО веаш,.туитН  Бваьгатрняуго  го имт ушд
гбаелокя рулитла, овя  авр ааасф иыйэ аосос  вкелякжзгз л;ууалть йен,мре  тнк.п  ияр
ыниаицсоонпеч  ыгасйецьуасыь очоиткводкетлкелУы тыевзечи амбигКыецо и румйнктлрнжпит
ниадарндвт  ея,унпурцор  фздр оирндпн во асрялррсак ллн  еанлие хуаарщн,ьуирк о лм
Ннди й  ап  р.д а дх и а дсиро я ! аа шео жл Уад,мсчгвутси нфз  й й,иулосол трсф е (ено
ензме,длнлрсита ла уж  д осе цао арввнеермоед рткио жвсл ск чияетоанр оывБдвы ь,см шл.
ор тыд. озрмшоеф пои га  )ьезекляшяжаш, ио  оов язеа  оглро о еом няяпоае,оези ,омпа
иц.р аиБзс од а ртктви цз еииеес ле  н и шзсс ыевкт и мее кт мл."
Ключ: "276-81-323-494-622-239-625-407-710-574-410-171-434-496-432-208-522-
426-348-186-16-498-352-658-563-329-240-595-473-707-359-247-454-153-431-
290-163-238-262-42-129-554-492-306-104-311-24-339-374-616-444-197-92-
308-160-463-244-516-576-205-4-722-425-427-3-251-300-673-291-214-317-253-
645-43-609-50-703-325-506-166-455-706-361-215-5-46-140-264-537-249-293-
217-560-487-521-316-254-517-36-702-169-699-17-358-157-626-680-679-
336-700-340-199-142-423-614-618-469-266-648-357-573-362-19-587-501-
690-421-650-395-433-245-363-299-164-675-147-281-176-181-61-510-716-661-
509-327-442-536-453-606-665-566-368-246-68-225-491-204-541-596-193-
59-715-260-525-87-570-382-226-242-118-719-148-324-39-381-500-
105-602-653-364-121-527-85".
```

Ключ представляет из себя номера прорезей в решётке, пронумерованной справа налево сверху вниз, со стороной, равной в длину ближайшему квадрату, большему длины текста (длина текста - 686 символов). Для данного текста в решётке 729 клеток, т.е. размеры решётки - 27 на 27.

Расшифруем текст обратно:

```
$ ./lab_02.py
Выберите действие:
1. Зашифровать текст с помощью ключа
2. Расшифровать текст с помощью ключа
Введите номер [1-2]: 2
Введите текст (на русском языке), который требуется зашифровать/расшифровать:
.н аажрг а,фсьрти еиваянраинс Взее тсрд  ц врпраннср ггясе рб ов уваиоатрА  слрл рглр м
нмиешзык  о и да:ло  йО веаш,.туитН  Бваьгатрняуго  го имт ушд гбаелокя рулитла, овя  авр
ааасф иыйэ аосос  вкелякжзгз л;ууалть йен,мре  тнк.п  ияр ыниаицсоонпеч  ыгасйецьуасыь
очоиткводкетлкелУы тыевзечи амбигКыецо и румйнктлрнжпит ниадарндвт  ея,унпурцор  фздр
оирндпн во асрялррсак ллн  еанлие хуаарщн,ьуирк о лм Ннди й  ап  р.д а дх и а дсиро
я ! аа шео жл Уад,мсчгвутси нфз  й й,иулосол трсф е (ено ензме,длнлрсита ла уж  д
осе цао арввнеермоед рткио жвсл ск чияетоанр оывБдвы ь,см шл.  ор тыд. озрмшоеф пои
га  )ьезекляшяжаш, ио  оов язеа  оглро о еом няяпоае,оези ,омпа иц.р аиБзс од а ртктви
цз еииеес ле  н и шзсс ыевкт и мее кт мл.
Введите ключ шифрования: 276-81-323-494-622-239-625-407-...
Результат: "уже близко становились французы; уже князь андрей, шедший рядом с
```

багратионом, ясно различал перевязи, красные эполеты, даже лица французов. (он ясно видел одного старого французского офицера, который вывернутыми ногами в штиблетах, придерживаясь за кусты, с трудом шел в гору.) князь багратион не давал нового приказания и все так же молча шел перед рядами. вдруг между французами треснул один выстрел, другой, третий... и по всем расстроившимся неприятельским рядам разнесся дым и затрещала пальба. несколько человек наших упало, в том числе и круглолицый офицер, шедший так весело и старательно. но в то же мгновение, как раздался первый выстрел, багратион оглянулся и закричал: ура! "

Таким образом, мы убедились, что программа работает корректно.

2.3. Контрольный вопрос

Какие ещё шифры перестановки вам известны?

Ответ: Шифр железнодорожной изгороди, шифр двойной перестановки, шифр карточной колоды, анаграммный шифр, шифр Энигма.

3. Лабораторная работа №3. ПОТОЧНЫЕ ШИФРЫ

Цель работы: Разработать генератор гаммы на базе линейного рекуррентного регистра сдвига и выполнить шифрование путём наложения гаммы на открытый текст.

Ход работы:

3.1. Исходный код программы

Ниже приведён код программы по шифрованию и дешифрованию на основе наложения гаммы на языке программирования Python 3.10 (листинг 9). Программа позволяет вводить данные с клавиатуры и файлов (тестовые данные - листинг 10, листинг 11).

```
import random
import sys
from typing import List, Optional, Tuple, Union

RU_ALPHABET = "абвгдеёжзийклмнопрстуфхцчшщъыьэюя"
ENCRYPT_ACTION = "--enc"
DECRYPT_ACTION = "--dec"


# Считывает текст из файла
def read_from_file_enc(filename: str) -> Optional[str]:
  try:
    with open(filename, "r") as f:
      buf = f.read().split('\n')[0]
      return buf
  except Exception as e:
    print(f'Ошибка: {str(e)}')
    return None
```

```
def read_from_file_dec(filename: str) -> Optional[Tuple[str, str]]:
  try:
    with open(filename, "r") as f:
      buf = f.read().split('\n')
      s, k = buf[0], buf[1]
      return (s, k)
  except Exception as e:
    print(f'Ошибка: {str(e)}')
    return None


# Считывает текст из терминала
def read_from_console_enc() -> str:
  print("Введите текст (на русском языке), который требуется зашифровать/
расшифровать:")
  return input().lower()

def read_from_console_dec() -> Tuple[str, str]:
  print("Введите текст (на русском языке), который требуется зашифровать/
расшифровать:")
  s = input().lower()
  k = input("Введите ключ шифрования: ")
  return (s, k)


# Создаёт ключ
def generate_key(length: int, a: int, c: int, m: int, seed: int) -> List[int]:
  key = [seed]
  for _ in range(1, length):
    next_value = (a * key[-1] + c) % m
    key.append(next_value)
  return key


# Конвертация индексов в символы и наоборот
def char_to_index(char: str) -> int:
  return RU_ALPHABET.index(char.lower())
def index_to_char(index: int, is_upper: bool) -> str:
  char = RU_ALPHABET[index]
  return char.upper() if is_upper else char


# Создаёт ключ и шифрует текст с его помощью
def encrypt(text: str, key: List[int]) -> Tuple[str, str]:
  encrypted = []
  for char, k in zip(text, key):
    if char.lower() in RU_ALPHABET:
      index = (char_to_index(char) + k) % 33
```

```
      encrypted_char = index_to_char(index, char.isupper())
      encrypted.append(encrypted_char)
    else:
      encrypted.append(char)
  return ''.join(encrypted), encode_key(key)


# Расшифровывает текст с помощью ключа
def decrypt(encrypted: str, key: str) -> str:
  key = decode_key(key)
  decrypted = []
  for char, k in zip(encrypted, key):
    if char.lower() in RU_ALPHABET:
      index = (char_to_index(char) - k) % 33
      decrypted_char = index_to_char(index, char.isupper())
      decrypted.append(decrypted_char)
    else:
      decrypted.append(char)
  return ''.join(decrypted)


# Кодирует ключ
def encode_key(key: list[int]) -> str:
  return '-'.join(map(str, key))
# Декодирует ключ
def decode_key(key: str) -> list[int]:
  return list(map(int, key.split('-')))


# Уточняет, что нужно сделать
def decide_action() -> str:
  def print_prompt() -> None:
    print('Выберите действие: ')
    print('1. Зашифровать текст с помощью ключа')
    print('2. Расшифровать текст с помощью ключа')

  def read_opt(allowed: list) -> Optional[int]:
    try:
      opt = int(input("Введите номер [1-2]: "))
      if opt not in allowed:
        print(f'Ошибка: опция #{opt} недоступна!')
        return None
      return opt
    except Exception as e:
      print(f'Ошибка: {str(e)}')
      return None

  print_prompt()
  opt = read_opt([1, 2])
```

```
  while opt is None:
    print_prompt()
    opt = read_opt([1, 2])

  match opt:
    case 1:
      return ENCRYPT_ACTION
    case 2:
      return DECRYPT_ACTION
    case _:
      return None


# Читает в программу текст и ключ
def read(filename: Optional[str], action: str) -> Optional[Union[Tuple[str, str],
str]]:
  if filename is None:
    if action == ENCRYPT_ACTION:
      return read_from_console_enc()
    else:
      return read_from_console_dec()
  else:
    if action == ENCRYPT_ACTION:
      return read_from_file_enc(filename)
    else:
      return read_from_file_dec(filename)


# Начало программы
if __name__ == '__main__':
  # Если программу запустить с аргументами (именем файла с текстом и ключом), ввод
будет считан из файла
  filename = None
  action = None
  for arg in sys.argv[1:]:
    if arg == ENCRYPT_ACTION:
      action = ENCRYPT_ACTION
    elif arg == DECRYPT_ACTION:
      action = DECRYPT_ACTION
    else:
      filename = arg

  if action is None:
    action = decide_action()

  if action == ENCRYPT_ACTION:
    # Параметры генерации ключа
    a = random.randint(1, 1000)
    c = random.randint(0, 1000)
```

```
    m = 33
    seed = random.randint(0, 32)

    s = read(filename, action)
    key = generate_key(len(s), a, c, m, seed)
    encrypted, key_str = encrypt(s, key)
    print(f'Результат: "{encrypted}"')
    print(f'Ключ: "{key_str}"')
  elif action == DECRYPT_ACTION:
    s, k = read(filename, action)
    decrypted = decrypt(s, k)
    print(f'Результат: "{decrypted}"')
```

Листинг 9. Файл lab_03.py

```
Безумие - это когда ты делаешь одно и то же, ожидая совсем другого результата.
```

Листинг 10. Файл lab_03_encrypt.txt

```
пхвюйвк - лги ззииа бл овеёйшд яюшл о тц чя, змндзн лщяккс лядэщаз хепвьцээлё.
15-17-27-11-30-26-6-5-0-8-15-17-27-11-30-26-6-5-0-8-15-17-27-11-30-
26-6-5-0-8-15-17-27-11-30-26-6-5-0-8-15-17-27-11-30-26-6-5-0-8-15-
17-27-11-30-26-6-5-0-8-15-17-27-11-30-26-6-5-0-8-15-17-27-11-
30-26-6-5
```

Листинг 11. Файл lab_03_decrypt.txt

3.2. Шифрование и дешифрование текста

В процессе выполнения программы ключ гаммирования (шифрующая последовательность ЛРС) генерируется автоматически и представляет из себя набор регистров. Сымитируем генерацию ЛРС и шифротекста и расшифровку текста:

```
$ ./lab_03.py
Выберите действие:
1. Зашифровать текст с помощью ключа
2. Расшифровать текст с помощью ключа
Введите номер [1-2]: 1
Введите текст (на русском языке), который требуется зашифровать/расшифровать:
Невероятно, но очень интересно!
Результат: "тэетисмкрь, ыж ьпзыф цёхтизяёс!"
Ключ:
"5-25-3-14-25-3-14-25-3-14-25-3-14-25-3-14-25-3-14-25-3-14-25-3-14-25-3-14"
```

Передаём другому адресату данные, и он проводит дешифровку текста:

```
$ ./lab_03.py
Выберите действие:
1. Зашифровать текст с помощью ключа
2. Расшифровать текст с помощью ключа
Введите номер [1-2]: 2
```

```
Введите текст (на русском языке), который требуется зашифровать/расшифровать:
тэетисмкрь, ыж ьпзыф цёхтизяёс!
Введите ключ шифрования:
5-25-3-14-25-3-14-25-3-14-25-3-14-25-3-14-25-3-14-25-3-14-25-3-14-25-3-14
Результат: "невероятно, но очень интересно!"
```

Таким образом, мы убедились, что программа работает корректно.

3.3. Контрольный вопрос

Перечислите разновидности поточных шифров и их особенности.

Ответ:

Поточные шифры делятся на два основных типа: синхронные и самосинхронизирующиеся. Синхронные поточные шифры генерируют ключевой поток независимо от открытого текста и зависят только от ключа шифрования и начального значения. Ключевой поток в этом случае комбинируется с открытым текстом обычно с использованием операции сложения по модулю два (XOR). Примеры таких шифров включают RC4 и A5/1. Их особенности заключаются в высокой скорости шифрования и дешифрования, что делает их идеальными для потоковых данных, таких как аудио- и видеопередача. Однако они уязвимы к атаке на повторный ключ, если тот же ключ используется дважды.

Самосинхронизирующиеся поточные шифры зависят не только от ключа и начального значения, но и от определенного числа предыдущих символов шифртекста. Это позволяет восстанавливать синхронизацию при потере части данных, что является их ключевым преимуществом в условиях ненадежных каналов связи. Примеры таких шифров включают шифр CFB (Cipher Feedback Mode) и OFB (Output Feedback Mode). Их использование позволяет достичь устойчивости к ошибкам передачи, однако скорость шифрования может быть ниже по сравнению с синхронными поточными шифрами из-за необходимости учитывать предыдущие данные.