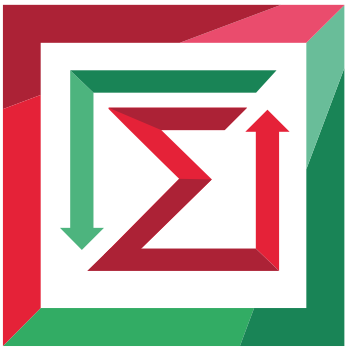
Министерство науки и высшего образования
Российской Федерации

Федеральное государственное бюджетное образовательное
учреждение высшего образования
«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ»

НГТУ
НЭТИ

Кафедра теоретической и прикладной информатики

Лабораторная работа № 2
по дисциплине «Программные Средства Защиты Информации»
Блочные шифры. Сеть Фейстеля



ФАКУЛЬТЕТ:	ПМИ
ГРУППА:	ПМИМ-01
СТУДЕНТЫ:	Ершов П. К. Малышкина Е. Д. Слободчикова А. Э.
БРИГАДА:	2
ПРЕПОДАВАТЕЛЬ:	Авдеенко Т. В.

Новосибирск
2021

1. Цель работы

Ознакомиться с блочными составными шифрами, освоить криптографические преобразования подстановки и перестановки. Изучить и реализовать шифрование информации при помощи сети Фейстеля.

2. Задание

1. Реализовать приложение для шифрования, позволяющее выполнять следующие действия:
   1.1. Шифровать данные при помощи сети Фейстеля:
      1) приложение должно позволять выбирать способ получения подключей из заданного ключа шифрования:
         a) для i-го раунда подключом V_i является цепочка из 32 подряд идущих бит заданного ключа, которая начинается с бита номер , продолжается до последнего бита ключа и при его достижении циклически повторяется, начиная с 1 бита;
         b) для i-го раунда, начиная с бита номер i, берётся цепочка из 8 подряд идущих бит ключа, которая является начальным значением для скремблера вида $0000\ 0011_2$; подключом V_i является сгенерированная этим скремблером последовательность из 32 бит;
      2) приложение должно позволять выбирать вид образующей функции:
         a) функция F – единичная, т.е. $F(V_i) = V_i$;
         b) функция F имеет вид $F(V_i, X) = S(X) \oplus V_i$, где $S(X)$ – левая часть шифруемого блока, на которую посредством операции XOR была наложена 32-битная последовательность, сгенерированная 16 разрядным скремблером вида $0100\ 0000\ 0000\ 0011_2$.
      3) в процессе шифрования предусмотреть возможность просмотра и изменения ключа, шифруемого и зашифрованного текстов в двоичном, шестнадцатеричном и символьном виде.

   1.2. Дешифровать данные в режиме однократного гаммирования:
      1) шифруемый текст должен храниться в файле;
      2) ключ шифрования должен задаваться случайным образом;
      3) зашифрованный текст должен сохраняться в один файл, а использовавшийся при шифровании ключ – в другой;

4) в процессе шифрования предусмотреть возможность просмотра и изменения ключа, шифруемого и зашифрованного текстов в двоичном, шестнадцатеричном и символьном виде.

2. Исследовать лавинный эффект (исследования проводить на одном блоке текста):

2.1. Дешифровать данные при помощи каждого заданного в варианте скремблера:
1) для бита, который будет изменяться, приложение должно позволять задавать его позицию (номер) в открытом тексте или в ключе;
2) приложение должно уметь после каждого раунда шифрования подсчитывать число бит, изменившихся в зашифрованном тексте при изменении одного бита в открытом тексте либо в ключе;
3) приложение может строить графики зависимости числа бит, изменившихся в зашифрованном тексте, от раунда шифрования, либо графики можно строить в стороннем ПО, но тогда приложение для шифрования должно сохранять в файл необходимую для построения графиков информацию.

3. С помощью реализованных приложений выполнить следующие задания:

3.1. Протестировать правильность работы разработанных приложений.
3.2. Исследовать лавинный эффект при изменении одного бита в открытом тексте и в ключе: построить графики зависимостей числа бит, изменившихся в зашифрованном сообщении, от раунда шифрования при всех возможных комбинациях способов выбора ключа и образующей функции (всего должно быть построено 8 графиков).
3.3. Сделать выводы о проделанной работе

3. Исследования

3.1. Демонстрация работы программы

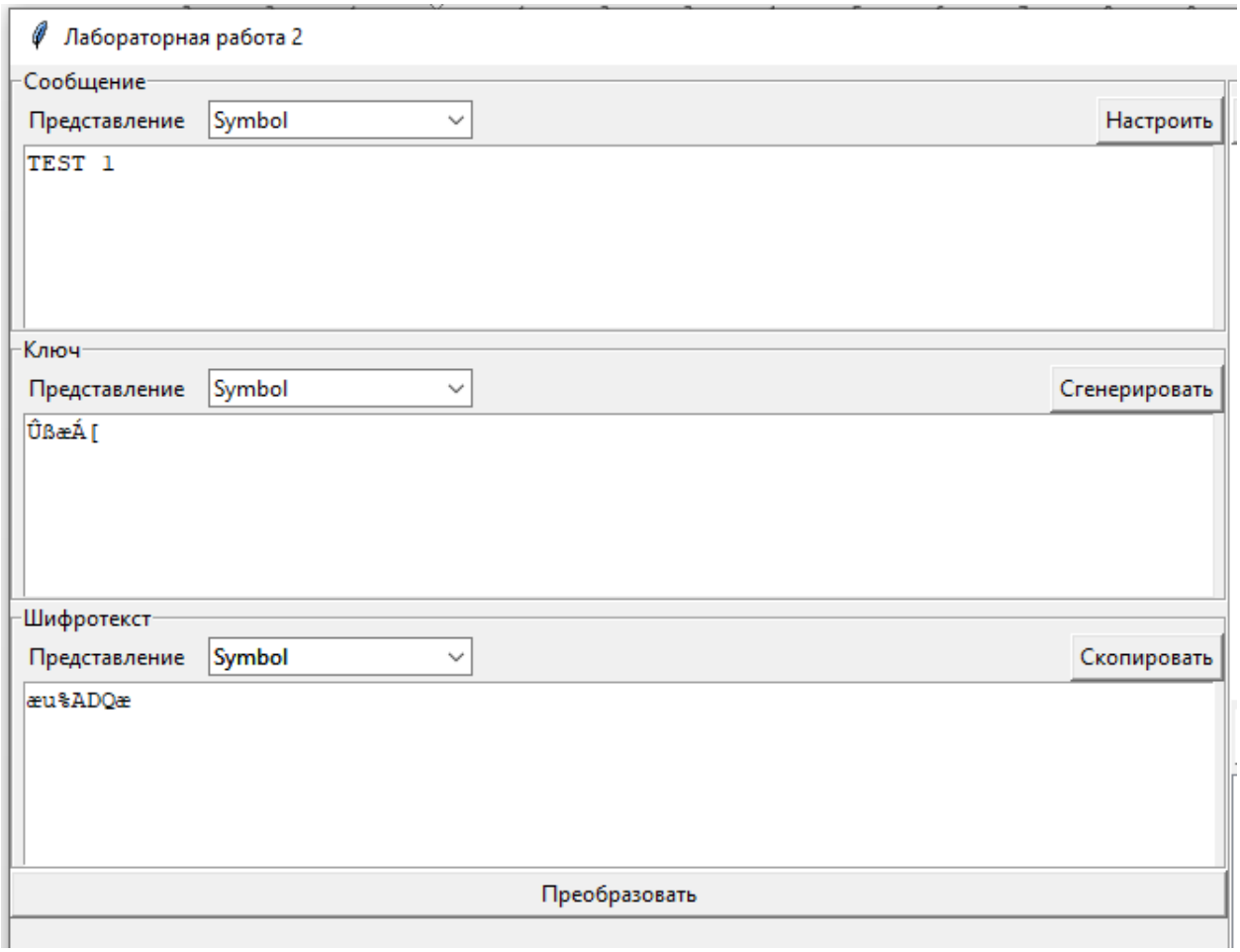


Рисунок 1. Текстовый формат данных

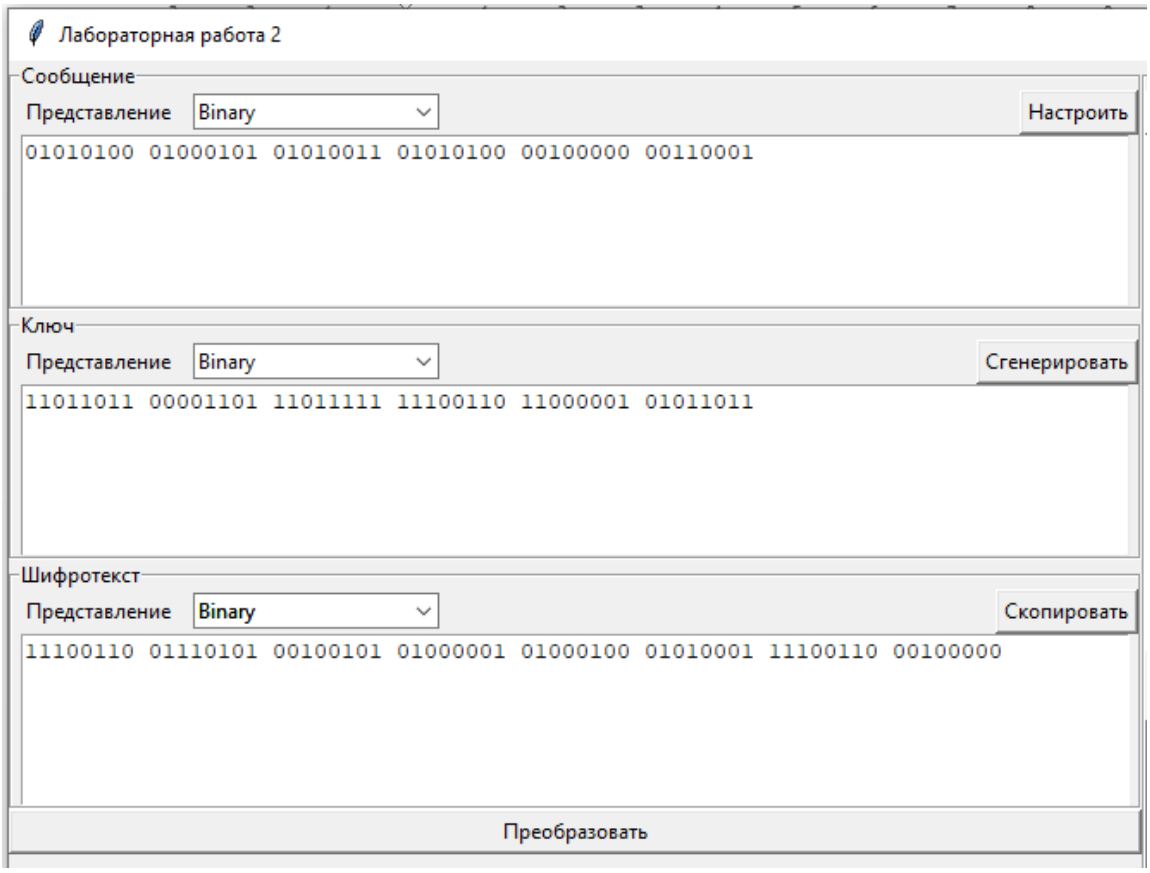


Рисунок 2. Двоичный формат данных

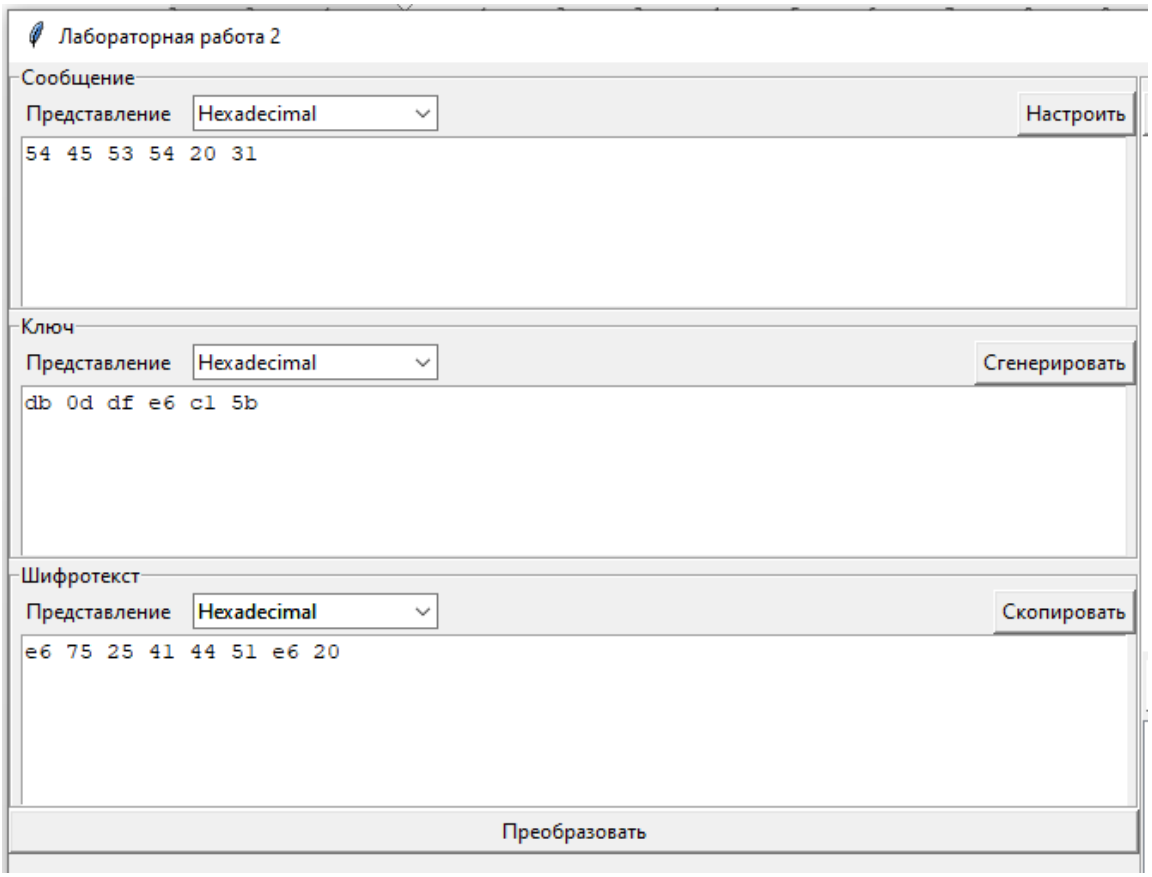


Рисунок 3. Шестнадцатеричный формат данных

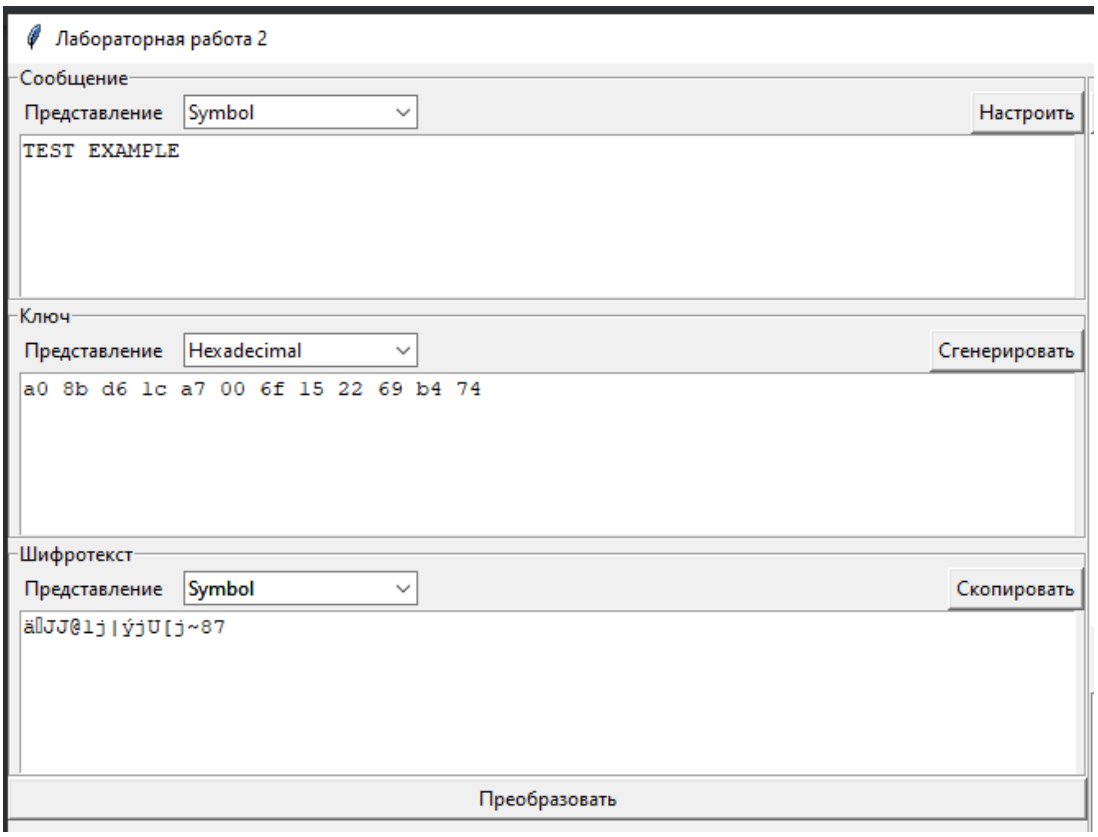


Рисунок 4. Пример шифрования

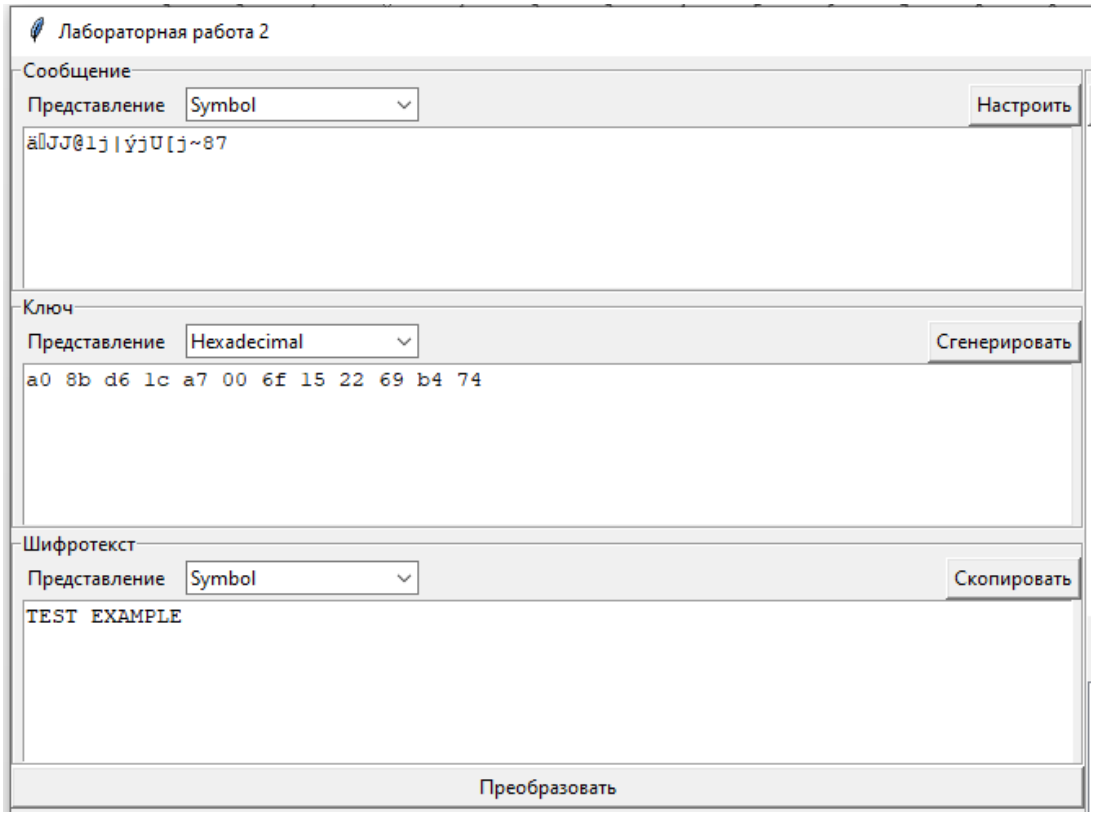


Рисунок 5. Пример дешифрования

3.2. Исследование лавинного эффекта

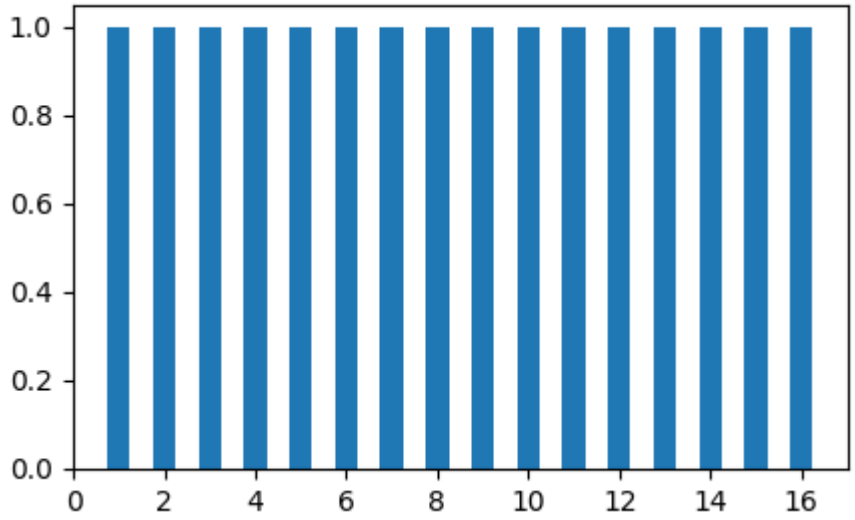


Рисунок 6. Изменение бита текста, единичная функция, циклический ключ

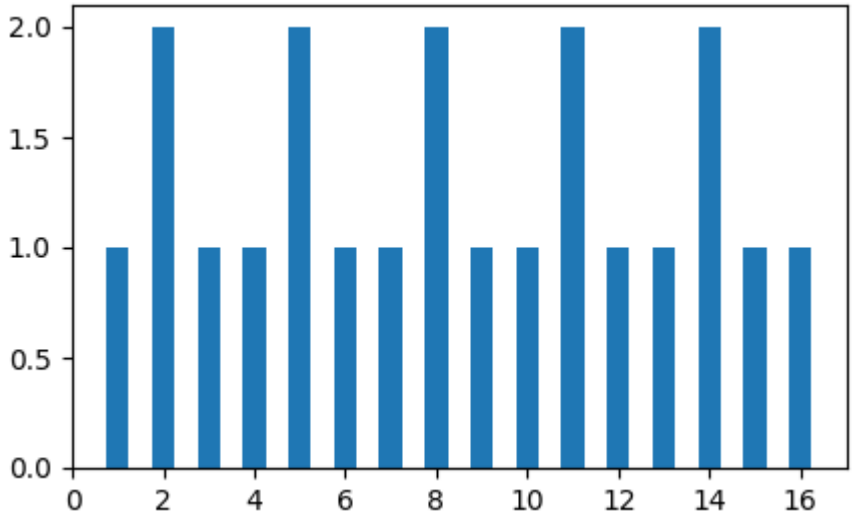


Рисунок 7. Изменение бита текста, функция S(X)⊕ V, циклический ключ

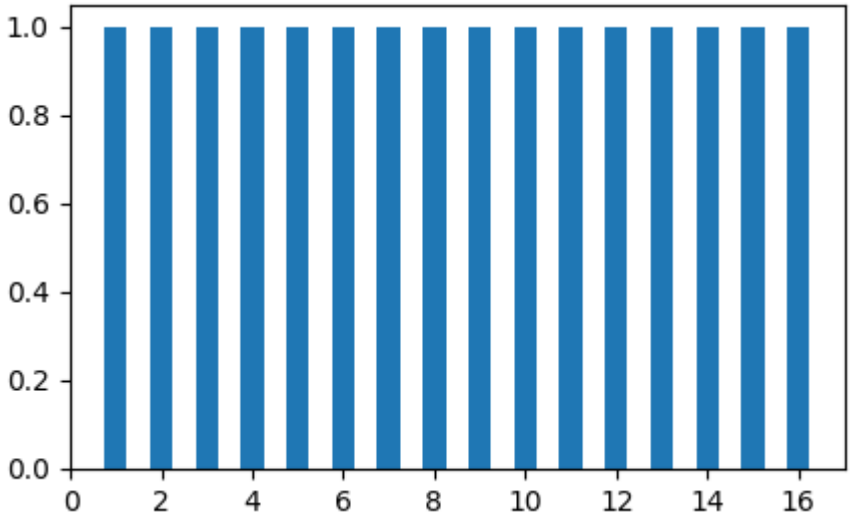


Рисунок 8. Изменение бита текста, единичная функция, ключ скремблер

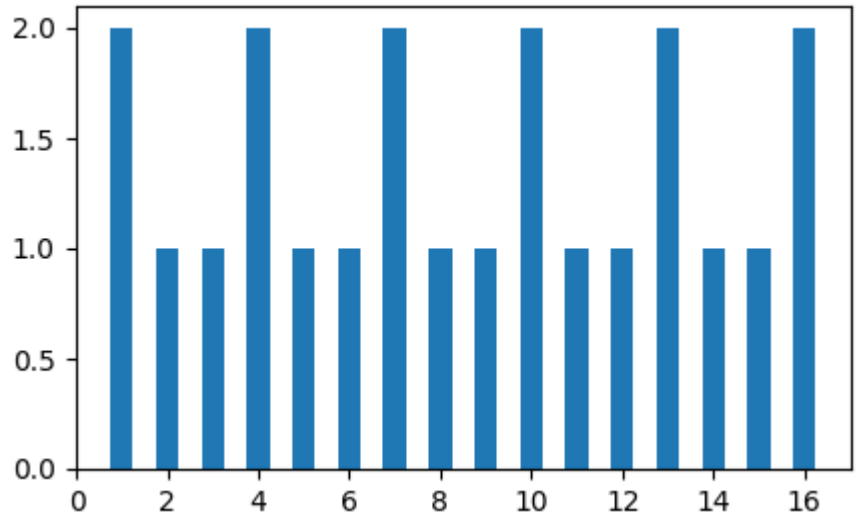


Рисунок 9. Изменение бита текста, функция S(X)⊕ V, ключ скремблер

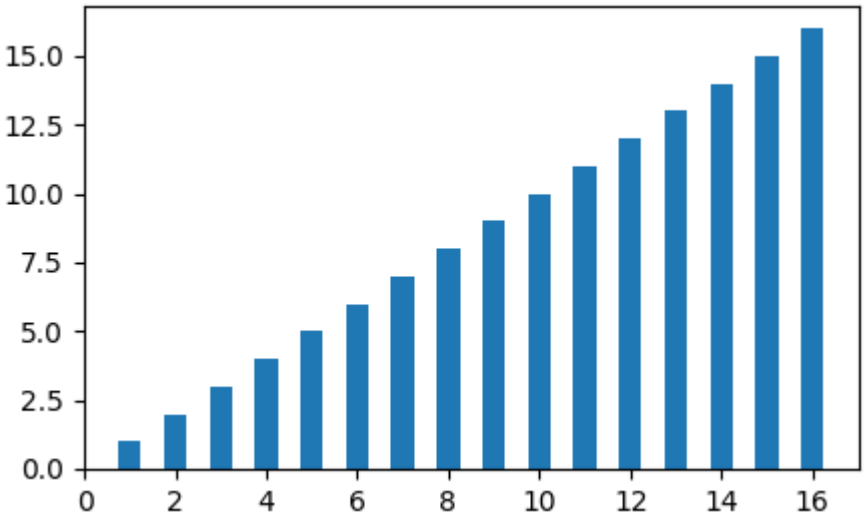


Рисунок 10. Изменение бита ключа, единичная функция, циклический ключ

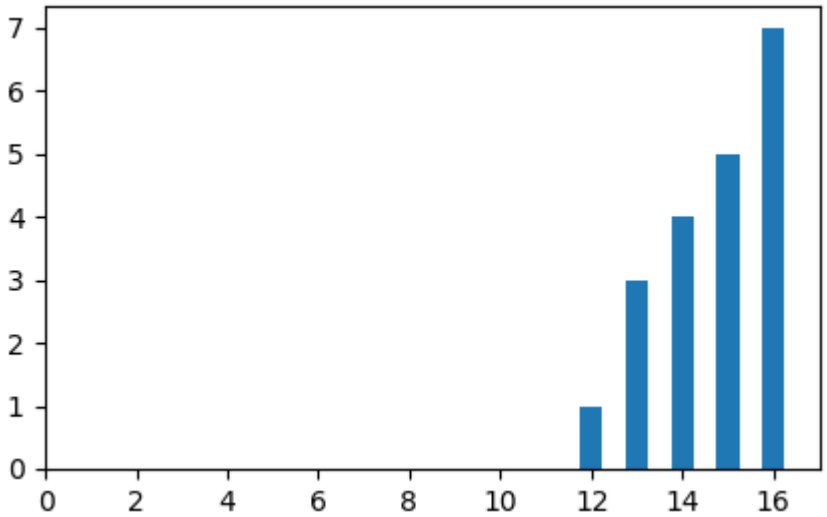


Рисунок 11. Изменение бита ключа, функция S(X)⊕ V, циклический ключ

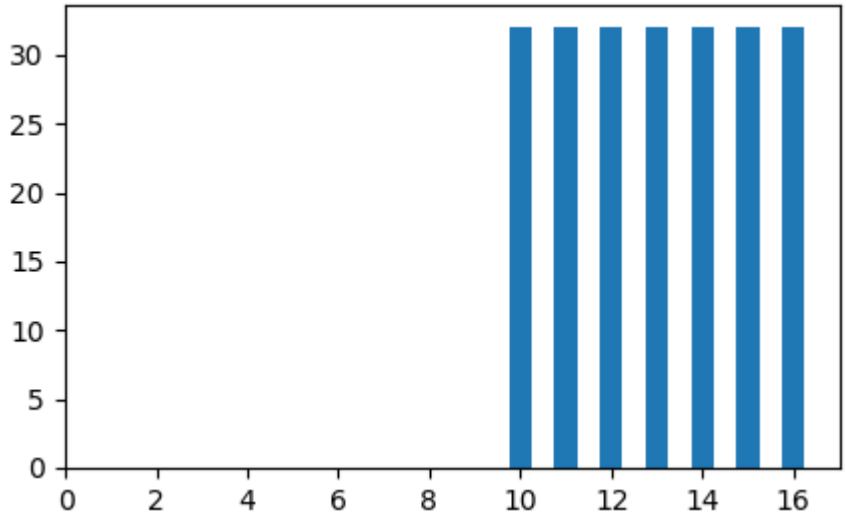


Рисунок 12. Изменение бита ключа, единичная функция, ключ скремблер

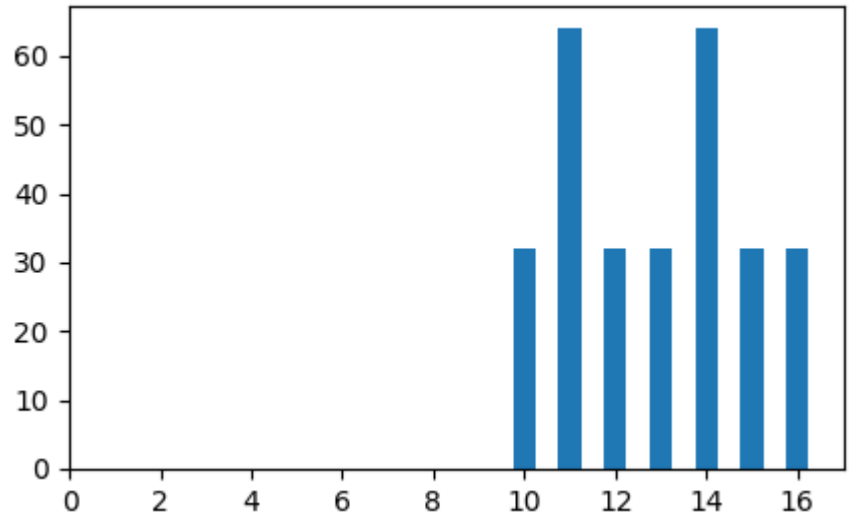


Рисунок 13. Изменение бита ключа, функция S(X)⊕ V, ключ скремблер

4. Выводы

Исходя из полученных результатов, изменение бита в блоке текста повлияет только на сам блок. При этом конечный результат не слишком измениться. В таком случае, если злоумышленнику известен текст, ключ и начальные значения генератора ключей, изменившийся символ в тексте после дешифрации будет принят за опечатку и восстановлен.

В тоже время, при изменении бита ключа, изменения в тексте будут нарастать. В случае достаточно высокой случайности ключа, изменения будут увеличиваться с каждым раундом, что будет усложнять подбор ключа частотным анализом, а, следовательно, повышать криптостойкость.

5. Код программы

main.py

```
import re
import tkinter as tk
from random import randint

import matplotlib
matplotlib.use("TkAgg")
from matplotlib.figure import Figure
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg, NavigationToolbar2Tk

import feistel
import serializers
import tkwidgets2 as tkw

SERIALIZER_OBJECTS = serializers.Binary(), serializers.Decimal(), serializers.Hexadecimal(),
serializers.Symbol()
SERIALIZER_PATTERNS = r"[^01 ]", r"[^\d ]", r"[^\da-f ]", r"[а-я]"

FEISTEL_FUNC_NAMES = "Единичная", "XOR"
FEISTEL_FUNCS = feistel.unit_func, feistel.xor_func
FEISTEL_KEY_NAMES = "Цикличный", "Скремблер"
FEISTEL_KEYS = feistel.cycled_key, feistel.scrambled_key

class NotatedField(tk.LabelFrame):
   def __init__(self, *args, **kwargs):
      default = kwargs.pop("default", 3)
      state = kwargs.pop("state", "normal")
      super().__init__(*args, **kwargs)

      self.text = bytes()
      self._serializer = SERIALIZER_OBJECTS[default] # symbol

      self.notation = tkw.DropDownList(self, text="Представление", items=tuple(map(str,
SERIALIZER_OBJECTS)), default=default, onSelectionChanged=self.onSelectionChanged)
      self.notation.pack(side=tk.TOP, fill=tk.X)
      self.out = tkw.Textbox(self, state=state, height=6, onTextChanged=self.onTextChanged)
      self.out.pack(side=tk.TOP, fill=tk.BOTH)

   def __getattr__(self, name):
      return getattr(self.out, name)

   ## EVENTS
   def onSelectionChanged(self, *args, **kwargs):
      cur = self.notation.cur
      self._serializer = SERIALIZER_OBJECTS[cur]
      self.set(self._serializer.encode(self.text))

   def onTextChanged(self, *args, **kwargs):
      if not kwargs.get("after"):
         return
      try:
         self.text = bytes(self._serializer.decode(kwargs["after"]))
      except ValueError:
         cur = self.notation.cur
         kwargs["after"] = re.sub(SERIALIZER_PATTERNS[cur], "", kwargs["after"], flags=re.I).strip()
         kwargs["recursion"] = True
         if not kwargs.get("recursion"):
            self.onTextChanged(*args, **kwargs)

class Translator(tk.Frame):
   def __init__(self, *args, **kwargs):
      super().__init__(*args, **kwargs)

      self.message = NotatedField(self, text="Сообщение")
      self.key = NotatedField(self, text="Ключ", default=0)
      tk.Button(self.key.notation, text="Сгенерировать", command=self.generate).pack(side=tk.RIGHT)
      self.res = NotatedField(self, text="Шифротекст", default=2, state=tk.DISABLED)
      tk.Button(self.res.notation, text="Скопировать", command=self.copy).pack(side=tk.RIGHT)

      # show elements
      self.message.pack(side=tk.TOP, fill=tk.X)
```

```
        self.key.pack(side=tk.TOP, fill=tk.X)
        self.res.pack(side=tk.TOP, fill=tk.X)

    def copy(self, *args, **kwargs):
        text = self.res.out.get()
        self.clipboard_clear()
        self.clipboard_append(text)
        self.update()

    def generate(self, *args, **kwargs):
        nums = []
        for _ in range(len(self.message.text)):
            nums.append(randint(0, 255))

        cur = self.key.notation.cur
        key = SERIALIZER_OBJECTS[cur].encode(bytes(nums))
        self.key.call_event("onTextChanged", tkw.EmptyEvent(self.key), after=key)
        self.key.onSelectionChanged()

class TranslatorModal(tkw.SimpleDialog):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, title="Лавинный эффект", width=700, height=400, confirm=False)
        self.cont = Translator(self)
        self.cont.message.notation._list.current(0)
        self.cont.key.notation._list.current(0)
        self.cont.res.pack_forget()
        self.cont.pack(side=tk.TOP, fill=tk.BOTH)
        tk.Button(self, text="Исследовать", command=self.onButtonPressed).pack(side=tk.TOP, expand=tk.YES)

    def onButtonPressed(self, *args, **kwargs):
        self._ret = self.cont.message.text, self.cont.key.text
        self.destroy()

class SettingsModal(tkw.SimpleDialog):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, title="Настройки", height=110, confirm=False)
        self.func = tkw.DropDownList(self, text="Образующая функция", items=FEISTEL_FUNC_NAMES, default=0)
        self.func.pack(side=tk.TOP, fill=tk.X)
        self.key = tkw.DropDownList(self, text="Генератор ключа", items=FEISTEL_KEY_NAMES, default=0)
        self.key.pack(side=tk.TOP, fill=tk.X)
        self.mode = tkw.DropDownList(self, text="Режим", items=("Шифрование","Дешифрование"), default=0)
        self.mode.pack(side=tk.TOP, fill=tk.X)
        tk.Button(self, text="Сохранить", command=self.onButtonPressed).pack(side=tk.TOP, expand=tk.YES)

    def onButtonPressed(self, *args, **kwargs):
        self._ret = self.func.cur, self.key.cur, self.mode.cur
        self.destroy()

class App(tk.Tk):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.title("Лабораторная работа 2")
        self._feistel_settings = 0, 0, 0

        left = tk.Frame(self)
        self.cont = Translator(left)
        tk.Button(self.cont.message.notation, text="Настроить",
command=self.onSettingsButtonPressed).pack(side=tk.RIGHT)
        self.cont.pack(side=tk.TOP, fill=tk.BOTH)
        tk.Button(left, text="Преобразовать", command=self.onConvertButtonPressed).pack(side=tk.TOP,
fill=tk.X)
        left.pack(side=tk.LEFT, expand=tk.YES, fill=tk.BOTH)

        right = tk.LabelFrame(self, text="Лавинный эффект")
        tk.Button(right, text="Исследовать", command=self.onAvalancheCalculate).pack(side=tk.TOP, fill=tk.X)

        self.fig = Figure(figsize=(5, 3), dpi=100)
        self.fig.ax = self.fig.add_subplot(111, xlim=(1,17), ylim=(0,None))
        cont = tk.Frame(right)
        self.plot = FigureCanvasTkAgg(self.fig, cont)
        self.plot.get_tk_widget().pack(side=tk.TOP, expand=tk.YES, fill=tk.BOTH)

        self.toolbar = NavigationToolbar2Tk(self.plot, cont)
        self.toolbar.update()
        # self.plot.get_tk_widget().pack(side=tk.TOP, expand=tk.YES, fill=tk.BOTH)
        self.plot._tkcanvas.pack(side=tk.TOP, expand=tk.YES, fill=tk.BOTH)
        cont.pack(side=tk.TOP)
```

```
        self.changes = tkw.CheckList(right, columns=({"name": "#", "width": 50}, {"name": "Различий",
"width": 70}))
        self.changes.pack(side=tk.TOP, fill=tk.BOTH)
        right.pack(side=tk.LEFT, expand=tk.YES, fill=tk.BOTH)

    def __getattr__(self, name):
        return getattr(self.cont, name)

    ## EVENTS
    def onAvalancheCalculate(self, *args, **kwargs):
        win = TranslatorModal(self)
        win.cont.message.text = self.message.text
        win.cont.message.onSelectionChanged()
        win.cont.key.text = self.key.text
        win.cont.key.onSelectionChanged()
        ret = win.wait()

        if ret is not None:
            message1 = self.message.text
            key1 = self.key.text
            message2, key2 = ret

            func = FEISTEL_FUNCS[self._feistel_settings[0]]
            key_func = FEISTEL_KEYS[self._feistel_settings[1]]
            fest = feistel.Feistel(func, key_func)
            call = fest.decrypt if self._feistel_settings[2] else fest.encrypt

            changes1 = []; changes2 = []
            res1 = call(message1, key1, rounds=16, changes=changes1)
            res2 = call(message2, key2, rounds=16, changes=changes2)
            diffs = tuple(feistel.binarize(feistel.xor(x, y)).count("1") for x, y in zip(changes1, changes2))

            self.fig.ax.clear()
            self.changes.clear()

            self.fig.ax.bar(list(range(1, 17)), diffs[:16], 0.5)
            self.plot.draw()
            self.changes.add(*diffs[:16])

    def onConvertButtonPressed(self, *args, **kwargs):
        message = self.message.text
        key = self.key.text
        if len(message) == 0 or len(key) == 0:
            if len(message) == 0:
                self.message.decline()
            if len(key) == 0:
                self.key.decline()
            return
        self.message.onSelectionChanged() # fix incorrect input
        self.message.accept()
        self.key.accept()

        func = FEISTEL_FUNCS[self._feistel_settings[0]]
        key_func = FEISTEL_KEYS[self._feistel_settings[1]]
        fest = feistel.Feistel(func, key_func)

        res = fest.decrypt(message, key) if self._feistel_settings[2] else fest.encrypt(message, key)
        self.res.text = res
        self.res.onSelectionChanged()

    def onSettingsButtonPressed(self, *args, **kwargs):
        ret = SettingsModal(self).wait()
        if ret is not None:
            print("New Feistel settings:", ret)
            self._feistel_settings = ret

if __name__ == "__main__":
    App().mainloop()
```

feistel.py

```python
import math
import typing as ty

import lfsr
from serializers import Binary

# utils
def xor(left: ty.Iterable[int], right: ty.Iterable[int]) -> bytes:
   return bytes(x ^ y for x, y in zip(left, right))

def split(arr: bytes, *, size: int = 4) -> ty.Sequence[bytes]:
   res = []
   if len(arr) % size != 0:
      arr = bytes(list(arr) + [ 10 ] * (size - len(arr) % size))
   for i in range(0, len(arr), size):
      res.append(arr[i:i+size])
   return res

def merge(left: ty.Iterable[int], right: ty.Iterable[int]) -> bytes:
   return bytes(left) + bytes(right)

def binarize(num: bytes) -> str:
   return Binary().encode(num).replace(" ", "")

def debinarize(num: str) -> bytes:
   return bytes(int(num[i:i+8], 2) for i in range(0, len(num), 8))

# activation functions
def unit_func(val: bytes, key: bytes) -> bytes:
   return bytes(key)

def xor_func(val: bytes, key: bytes) -> bytes:
   bits = [] # type: ty.List[int]
   gen = lfsr.generator(0b1000, taps=(14,1,0), size=16) # 0100 0000 0000 0011
   while len(bits) < 32:
      seed = next(gen)
      bit = lfsr.lsb(seed)
      bits.append(bit)
   gamma = debinarize("".join(map(str, bits)))
   return xor(xor(val, gamma), key)

# key functions
def cycled_key(init: bytes, *, size: int = 32) -> ty.Generator[bytes, int, None]:
   key = "" # first (init) value
   seed = binarize(init)[::-1]
   while True:
      i = yield debinarize(key)
      gen = lfsr.cycle(seed)
      for _ in range(i): # skip i elements
         next(gen)
      key = "".join(next(gen) for _ in range(size))

def scrambled_key(init: bytes) -> ty.Generator[bytes, int, None]:
   key_gen = cycled_key(init, size=8)
   next(key_gen) # init generator

   key = "" # first (init) value
   while True:
      i = yield debinarize(key)
      seed = key_gen.send(i)[0]
      gen = lfsr.generator(seed, taps=(1,0), size=8) # 0000 0011
      key = "".join(map(str, map(lfsr.lsb, (next(gen) for _ in range(32)))))

class Feistel:
   def __init__(self, func: ty.Callable[[bytes, bytes], bytes], key_func: ty.Callable[[bytes],
ty.Generator[bytes, int, None]]) -> None:
      self.func = func
      self.key_func = key_func

   def _apply_block(self, message: bytes, key: bytes, *, rounds: int = 16, changes: ty.List[bytes] = [],
reverse: bool = False) -> bytes:
      key_gen = self.key_func(key)
      next(key_gen)
```

```
        subkeys = [] # type: ty.List[bytes]
        for i in range(rounds):
            subkey = key_gen.send(i + 1)
            subkeys.append(subkey)

        left, right = split(message)
        for i in (reversed(range(rounds)) if reverse else range(rounds)):
            x = self.func(right, subkeys[i])
            left, right = right, xor(x, left)
            changes.append(merge(left, right))

        res = merge(right, left) if reverse else merge(left, right)
        return res

    def decrypt(self, cypher: bytes, key: bytes, *, rounds: int = 16, changes: ty.List[bytes] = []) ->
bytes:
        res = bytes()
        for x in split(cypher, size=8):
            left, right = split(x)
            res += self._apply_block(merge(right, left), key, rounds=rounds, changes=changes, reverse=True)
        return res.strip()

    def encrypt(self, message: bytes, key: bytes, *, rounds: int = 16, changes: ty.List[bytes] = [],
reverse: bool = False) -> bytes:
        res = bytes()
        for x in split(message, size=8):
            res += self._apply_block(x, key, rounds=rounds, changes=changes)
        return res

if __name__ == "__main__":
    ser = Feistel(xor_func, scrambled_key)
    message = "hirasawa"
    data = message.encode()
    key = "k-on".encode()

    out1 = [] # type: ty.List[bytes]
    res1 = ser.encrypt(data, key, changes=out1)
    print(data, key, res1)

    out2 = [] # type: ty.List[bytes]
    res2 = ser.decrypt(res1, key, changes=out2)
    print(res1, key, res2)
```

lfsr.py

```
import typing as ty

def lsb(num: int) -> int:
    return num & 0x01
def shift(num: int, bit: int, *, size: int = 8) -> int:
    return (num >> 1) | (bit << (size - 1))

def generator(seed: int, *, taps: ty.Iterable[int], size: ty.Optional[int] = None) -> ty.Iterator[int]:
    if size is None:
        size = max(taps)
    while True:
        bit = 0
        for x in taps:
            bit ^= lsb(seed >> x)
        yield shift(seed, bit, size=size)


if __name__ == "__main__":
    i = 0
    res = []
    vals = [ 0b11101011 ]
    gen = generator(vals[0], taps=(7,3,1), size=8)
    while vals.count(vals[-1]) < 2 or i < 10:
        vals.append(next(gen))
        res.append(lsb(vals[-1]))
        i += 1
```