

# **Лабораторная 7**

**Отчет**

Карпачев Ярослав

# Содержание

1	Цель работы	5
2	Задание	6
3	Выполнение лабораторной работы	7
4	Выводы	11

# Список иллюстраций

3.1	Работа программы . . . . .	9
-----	----------------------------	---

## **Список таблиц**

# 1 Цель работы

Освоить на практике применение режима однократного гаммирования

## 2 Задание

Нужно подобрать ключ, чтобы получить сообщение «С Новым Годом, друзья!». Требуется разработать приложение, позволяющее шифровать и дешифровать данные в режиме однократного гаммирования. Приложение должно:

1. Определить вид шифротекста при известном ключе и известном открытом тексте.
2. Определить ключ, с помощью которого шифротекст может быть преобразован в некоторый фрагмент текста, представляющий собой один из возможных вариантов прочтения открытого текста.

### 3 Выполнение лабораторной работы

#### 1. Пишим скрипт

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import random
import string
from typing import List, Tuple

EXCHARS = "0123456789ABCDEF"

def generate_key(length: int) -> str:
    """Return pseudo-random printable HEX-like string of *length* symbols."""
    return ''.join(random.choice(EXCHARS) for _ in range(length))

def xor_strings(a: str, b: str) -> str:
    """XOR two equal-length strings, return new string of chars."""
    return ''.join(chr(ord(x) ^ ord(y)) for x, y in zip(a, b))

def to_hex(s: str) -> str:
    """Convert string → 'AA BB CC' spaced HEX representation."""
```

```
return ' '.join(f"{ord(ch):02X}" for ch in s)
```

```
def find_possible_keys(cipher: str, fragment: str) -> List[Tuple[int, str]]:  
    frag_len = len(fragment)  
    out: List[Tuple[int, str]] = []  
    for pos in range(len(cipher) - frag_len + 1):  
        key_candidate = xor_strings(cipher[pos:pos + frag_len], fragment)  
        out.append((pos, key_candidate))  
    return out
```

```
def main() -> None:  
    plain = "С Новым Годом, друзья!"  
  
    key = generate_key(len(plain))  
  
    cipher = xor_strings(plain, key)  
  
    print("1. Open text      :", plain)  
    print("2. Key            :", key)  
    print("3. Crypto text       :", to_hex(cipher))  
  
    fragment = input("4. Input fragment : ")  
  
    cand = find_possible_keys(cipher, fragment)  
    if not cand:  
        print("5. Possible keys : – (фрагмент не найден)")  
        return
```



```

keys_only = [k for _, k in cand]
print("5. Possible keys :", ', '.join(keys_only))

pos0, key0 = cand[0]
decrypted = xor_strings(cipher[pos0:pos0 + len(fragment)], key0)
print("6. Decrypted frag.:", decrypted)

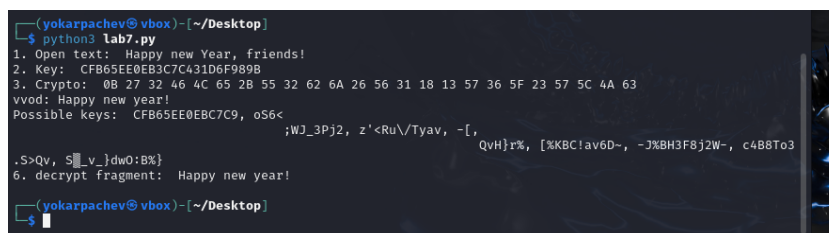
```

```

if name == "__main__":
    main()

```

## 2. проверяем работу программы



```

(yokarpachev@vbox)~/Desktop
$ python3 lab7.py
1. Open text: Happy new Year, friends!
2. Key: CFB65EE0EB3C7C431D6F989B
3. Crypto: 0B 27 32 46 4C 65 2B 55 32 62 6A 26 56 31 18 13 57 36 5F 23 57 5C 4A 63
vвод: Happy new year!
Possible keys: CFB65EE0EBC7C9, oS6<
;WJ_3Pj2, z'<Ru\Tyav, -[, QvH}r%, [%KBC!av6D~, -J%BH3F8j2W-, c4B8To3
.S>Qv, S[_v_]dw0:B%}
6. decrypt fragment: Happy new year!
(yokarpachev@vbox)~/Desktop
$

```

Рис. 3.1: Работа программы

## 3. Контрольные вопросы

4. Смысл однократного гаммирования – сложение (XOR) текста с однократной случайной гаммой той же длины.
5. Недостатки: нужна истинно случайная гамма; ключ хранить/передавать так же долго, как сообщение; ключ нельзя переиспользовать.
6. Преимущества: абсолютная криптостойкость; простота реализации; симметричность (шифр = дешифр).

7. Длины равны, чтобы каждый символ текста «прикрывался» одним символом гаммы; иначе остаётся статистическая избыточность.
8. Операция – XOR (сложение по модулю 2); даёт -- при повторном применении тем же ключом восстанавливается исходник.
9. Шифротекст:  $C_i = P_i \oplus K_i$ .
10. Ключ:  $K_i = C_i \oplus P_i$ .
11. Условия абсолютной стойкости: (а) гамма истинно случайна; (б) длина ключа = длина сообщения; (с) ключ используется лишь однажды.

## 4 Выводы

Я освоить на практике применение режима однократного гаммирования