

Отчёт по лабораторной работе №8

Шифр гаммирования

Данил Исаев

Содержание

1 Цель работы	4
2 Теоретические сведения	5
2.1 Шифр гаммирования	5
2.2 Идея взлома	6
3 Выполнение работы	8
3.1 Реализация взломщика, шифратора и дешифратора на Python . .	8
3.2 Контрольный пример	11
4 Выводы	12
Список литературы	13

List of Figures

3.1 Работа алгоритма взлома ключа	11
3.2 Работа алгоритма шифрования и дешифровки	11

1 Цель работы

Освоить на практике применение режима однократного гаммирования на примере кодирования различных исходных текстов одним ключом.

2 Теоретические сведения

2.1 Шифр гаммирования

Гаммирование – это наложение (снятие) на открытые (зашифрованные) данные криптографической гаммы, т.е. последовательности элементов данных, вырабатываемых с помощью некоторого криптографического алгоритма, для получения зашифрованных (открытых) данных.

Принцип шифрования гаммированием заключается в генерации гаммы шифра с помощью датчика псевдослучайных чисел и наложении полученной гаммы шифра на открытые данные обратимым образом (например, используя операцию сложения по модулю 2). Процесс дешифрования сводится к повторной генерации гаммы шифра при известном ключе и наложении такой же гаммы на зашифрованные данные. Полученный зашифрованный текст является достаточно трудным для раскрытия в том случае, если гамма шифра не содержит повторяющихся битовых последовательностей и изменяется случайным образом для каждого шифруемого слова. Если период гаммы превышает длину всего зашифрованного текста и неизвестна никакая часть исходного текста, то шифр можно раскрыть только прямым перебором (подбором ключа). В этом случае криптостойкость определяется размером ключа.

Метод гаммирования становится бессильным, если известен фрагмент исходного текста и соответствующая ему шифrogramма. В этом случае простым вычислением по модулю 2 получается отрезок псевдослучайной последовательности и по нему восстанавливается вся эта последовательность.

Метод гаммирования с обратной связью заключается в том, что для получения сегмента гаммы используется контрольная сумма определенного участка шифруемых данных. Например, если рассматривать гамму шифра как объединение непересекающихся множеств $H(j)$, то процесс шифрования можно представить следующими шагами:

1. Генерация сегмента гаммы $H(1)$ и наложение его на соответствующий участок шифруемых данных.
2. Подсчет контрольной суммы участка, соответствующего сегменту гаммы $H(1)$.
3. Генерация с учетом контрольной суммы уже зашифрованного участка данных следующего сегмента гаммы $H(2)$.
4. Подсчет контрольной суммы участка данных, соответствующего сегменту данных $H(2)$ и т.д.

2.2 Идея взлома

Шифротексты обеих телеграмм можно получить по формулам режима однократного гаммирования:

$$C_1 = P_1 \oplus K$$

$$C_2 = P_2 \oplus K$$

Открытый текст можно найти, зная шифротекст двух телеграмм, зашифрованных одним ключом. Для этого оба равенства складываются по модулю 2. Тогда с учётом свойства операции XOR получаем:

$$C_1 \oplus C_2 = P_1 \oplus K \oplus P_2 \oplus K = P_1 \oplus P_2$$

Предположим, что одна из телеграмм является шаблоном — т.е. имеет текст фиксированный формат, в который вписываются значения полей. Допустим, что злоумышленнику этот формат известен. Тогда он получает достаточно много пар $C_1 \oplus C_2$ (известен вид обеих шифровок). Тогда зная P_1 имеем:

$$C_1 \oplus C_2 \oplus P_1 = P_1 \oplus P_2 \oplus P_1 = P_2$$

Таким образом, злоумышленник получает возможность определить те символы сообщения P_2 , которые находятся на позициях известного шаблона сообщения P_1 . В соответствии с логикой сообщения P_2 , злоумышленник имеет реальный шанс узнать ещё некоторое количество символов сообщения P_2 . Затем вновь используется равенство с подстановкой вместо P_1 полученных на предыдущем шаге новых символов сообщения P_2 . И так далее. Действуя подобным образом, злоумышленник даже если не прочитает оба сообщения, то значительно уменьшит пространство их поиска.

3 Выполнение работы

3.1 Реализация взломщика, шифратора и дешифратора на Python

```
a = ord("a")
liters = [chr(i) for i in range(a, a + 32)]
a = ord("0")
for i in range(a, a+10):
    liters.append(chr(i))

a = ord("A")
for i in range(1040, 1072):
    liters.append(chr(i))

P1 = "КодофаяФраза1"
P2 = "Безопасность2"

def vzlom(P1, P2):
    code = []
    for i in range(len(P1)):
        code.append(liters[(liters.index(P1[i]) + liters.index(P2[i])) % len(liters)])
    print(code)
    pr = "".join(code)
```

```

print(pr)

def shifr(P1, gamma):

dicts = {"а": 1, "б": 2, "в": 3, "г": 4, "д": 5, "е": 6, "ё": 7, "ж": 8, "з": 9,
         "м": 14, "н": 15, "օ": 16, "ռ": 17, "պ": 18, "ս": 19, "տ": 20, "յ": 21,
         "շ": 26, "Щ": 27, "՚": 28, "՚": 29, "՚": 30, "՚": 31, "՚": 32, "՚": 33,
         "՚": 37, "՚": 38, "՚": 39, "՚": 40, "՚": 41, "՚": 42, "՚": 43, "՚": 44,
         "՚": 49, "՚": 50, "՚": 51, "՚": 52, "՚": 53, "՚": 54, "՚": 55, "՚": 56,
         "՚": 61, "՚": 62, "՚": 63, "՚": 64, "՚": 65, "՚": 66, "՚": 67, "՚": 68,
         "՚": 73, "՚": 74, "՚": 75}

dicts2 = {v: k for k, v in dicts.items()}

text = P1

digits_text = []
digits_gamma = []

for i in text:
    digits_text.append(dicts[i])
print("Числа текста ", digits_text)

for i in gamma:
    digits_gamma.append(dicts[i])
print("Числа гаммы ", digits_gamma)

digits_result = []
ch = 0

for i in text:
    try:
        a = dicts[i] + digits_gamma[ch]

```

```

except:
    ch = 0
    a = dicts[i] + digits_gamma[ch]
if a > 75:
    a = a%75
    print(a)
ch += 1
digits_result.append(a)
print("Числа шифротекста ", digits_result)

text_cr = ""
for i in digits_result:
    text_cr += dicts2[i]
print("Шифротекст ", text_cr)

digits = []
for i in text_cr:
    digits.append(dicts[i])
ch = 0
digits1 = []
for i in digits:
    try:
        a = i - digits_gamma[ch]
    except:
        ch = 0
        a = i - digits_gamma[ch]
    if a < 1:
        a = 75 + a
    digits1.append(a)

```

```

ch += 1

text_decr = ""

for i in digits1:

    text_decr += dicts2[i]

print("Расшифрованный текст: ", text_decr)

```

3.2 Контрольный пример

```

13
14 def vzlom(P1, P2):
15     code = []
16     for i in range(len(P1)):
17         code.append(liters[(liters.index(P1[i]) + liters.index(P2[i])) % len(liters)])
18     print(code)
19     pr = "".join(code)
20     print(pr)

In [11]: 1 len(P1)
Out[11]: 13

In [12]: 1 len(P2)
Out[12]: 13

In [13]: 1 vzlom(P1, P2)
['x', 'у', 'л', 'в', 'з', 'а', 'ж', 'б', 'и', 'о', 'с', 'щ', 'ь', 'щ']
хульЗакбесщш

```

Figure 3.1: Работа алгоритма взлома ключа

```

In [24]: 1 P1 = "КодофаяФраза1"
2 gamma = "хульЗакбесщш"

In [25]: 1 shifr(P1, gamma)
Числа текста [44, 16, 5, 16, 22, 1, 32, 54, 18, 1, 9, 1, 66]
Числа гаммы [23, 21, 13, 30, 68, 1, 40, 2, 32, 19, 27, 30, 59]
15
50
Числа шифротекста [67, 37, 18, 46, 15, 2, 72, 56, 50, 20, 36, 31, 50]
Шифртекст 2дрии67ЦРтЭР
Расшифрованный текст: КодофаяФраза1

```

Figure 3.2: Работа алгоритма шифрования и дешифровки

4 Выводы

В ходе выполнения лабораторной работы было разработано приложение, позволяющее шифровать тексты в режиме однократного гаммирования.

Список литературы

1. Шифрование методом гаммирования
2. Режим гаммирования в блочном алгоритме шифрования