

Отчёт по лабораторной работе №8

Шифр гаммирования

Эттеев Сулейман

Содержание

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

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 шифрограмма. В этом случае простым вычитанием 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 гамм $H(2)$.
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 это 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

```
# создаем алфавит из русских букв и цифр
# он нужен для гаммирования
a = ord("а")
alphabeth = [chr(i) for i in range(a, a + 32)]
a = ord("0")
for i in range(a, a+10):
    alphabeth.append(chr(i))

a = ord("А")
for i in range(1040, 1072):
    alphabeth.append(chr(i))
print(alphabeth)
P1 = "НаВашисходящийот1204"
P2 = "ВСеверныйфилиалБанка"
# длина ключа 20
key = "05 0C 17 7F 0E 4E 37 D2 94 10 09 2E 22 57 FF C8 0B B2 70 54"

def vzlom(P1, P2):
    code = []
    for i in range(20):
        code.append(alphabeth[(alphabeth.index(P1[i]) + alphabeth.index(P2[i]))
% len(alphabeth)])
    #получили известные символы в шаблоне
    print(code)
    print(code[16], " и ", code[19])
    p3 = "".join(code)
    print(p3)

vzlom(P1, P2)

def shifr(P1):
    # создаем алфавит
    dicts = {"а": 1, "б": 2, "в": 3, "г": 4, "д": 5, "е": 6, "ё": 7, "ж": 8,
"з": 9, "и": 10, "й": 11, "к": 12, "л": 13,
"м": 14, "н": 15, "о": 16, "п": 17,
"р": 18, "с": 19, "т": 20, "у": 21, "ф": 22, "х": 23, "ц": 24, "ч":
25, "ш": 26, "щ": 27, "ъ": 28,
"ы": 29, "ь": 30, "э": 31, "ю": 32, "я": 32, "А":33 , "Б": 34, "В":
35 , "Г":36 , "Д":37 , "Е":38 , "Ё":39 , "Ж":40 , "З":41 ,
"И":42,"Й":43 , "К":44 , "Л":45 , "М":46 , "Н":47 , "О":48 ,
"П":49 , "Р":50 , "С":51 , "Т":52 , "У":53 , "Ф":54 , "Х":55 , "Ц":56 , "Ч":57 ,
"Ш":58,"Щ":59 , "Ъ":60 , "Ы":61 , "Ь":62 , "Э":63 , "Ю":64 , "Я":65 , "1":66
, "2":67 , "3":68 , "4":69 , "5":70 , "6":71 , "7": 72, "8":73 , "9":74 , "0":75
```

```

}
# меняем местами ключ и значение, такой словарь понадобится в будущем
dict2 = {v: k for k, v in dicts.items()}
text = P1
gamma = input("Введите гамму(на русском языке! Да и пробелы тоже нельзя!
Короче, только символы из dict")
listofdigitsoftext = list() # сюда будем записывать числа букв из текста
listofdigitsofgamma = list() # для гаммы
# запишем числа в список
for i in text:
    listofdigitsoftext.append(dicts[i])
print("Числа текста", listofdigitsoftext)
# то же самое сделаем с гаммой
for i in gamma:
    listofdigitsofgamma.append(dicts[i])
print("числа гаммы", listofdigitsofgamma)
listofdigitsresult = list() # сюда будем записывать результат
ch = 0
for i in text:
    try:
        a = dicts[i] + listofdigitsofgamma[ch]
    except:
        ch = 0
        a = dicts[i] + listofdigitsofgamma[ch]
    if a > 75:
        a = a%75
    print(a)
    ch += 1
    listofdigitsresult.append(a)
print("Числа зашифрованного текста", listofdigitsresult)
# теперь обратно числа представим в виде букв
textencrypted = ""
for i in listofdigitsresult:
    textencrypted += dict2[i]
print("Зашифрованный текст: ", textencrypted)
# теперь приступим к реализации алгоритма дешифровки
listofdigits = list()
for i in textencrypted:
    listofdigits.append(dicts[i])
ch = 0
listofdigits1 = list()
for i in listofdigits:
    try:
        a = i - listofdigitsofgamma[ch]
    except:
        ch=0
        a = i - listofdigitsofgamma[ch]
    if a < 1:
        a = 75 + a
    listofdigits1.append(a)
    ch += 1
textdecrypted = ""

```

```

for i in listofdigits1:
    textdecrypted += dict2[i]
print("Расшифрованный текст", textdecrypted)

```

shifr(P1)

3.2 Контрольный пример

```

In [2]: 1 def vzlom(P1, P2):
        2     code = []
        3     for i in range(20):
        4         code.append(alph[(alph.index(P1[i]) + alph.index(P2[i])) %len(alph) ])
        5     print(code)
        6     p3 = "".join(code)
        7     print(p3)

In [3]: 1 vzlom(P1, P2)

['щ', 'С', 'З', 'в', 'э', 'Ш', 'ю', 'Ж', 'ч', 'Ш', '7', '4', 'р', 'й', 'щ', 'У', '1', 'Е', 'А', '4']
щСЗвэШюЖчШ74рйщУ1ЕА4

```

Figure 1: Работа алгоритма взлома ключа

```

In [13]: 1 shifr(P1, gamma)

числа текста  [47, 1, 35, 1, 26, 42, 19, 23, 16, 5, 32, 27, 10, 11, 16, 20, 66, 67, 75, 69]
Числа гаммы  [27, 51, 41, 3, 31, 58, 32, 40, 25, 58, 72, 69, 18, 11, 27, 53, 66, 38, 33, 69]
1
25
29
21
57
30
33
63
Числа шифра:  [74, 52, 1, 4, 57, 25, 51, 63, 41, 63, 29, 21, 28, 22, 43, 73, 57, 30, 33, 63]
шифровка 9ТагЧсЭЗЭуьфЙ8чьАЭ
Расшифровка  НаВашИсходящийот1204

```

Figure 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