Федеральное агентство по образованию

Государственное образовательное учреждение

высшего профессионального образования

Владимирский государственный университет

имени Александра Григорьевича и Николая Григорьевича Столетовых

**Пояснительная записка к докладу**

**«Автоматизированный метод атаки на шифр гаммирования».**

Выполнил:

Ст. группы КЗИ-109

Васильев А.О.

Владимир 2012

**Цель работы:**

Совершить криптографическую атаку на текст, зашифрованный методом гаммирования. Получить в результате исходный текст и ключ-гамму.

**Задание:**

Рассмотрен шифр:

*ЧШЩДАЪРЕЩМСУВЗЕЬГЪЕУЯЛИУЙССДФЛЫУЭЪКТМХЕУУЧОЫ ОЖТЯЖПДЮЬЖС ЬФССЭРРЧЫЕФ ЭСКСЮФИ ЛРЕЬЛНЕЙЧЖДЭЯУЕСЩЛФБЪУНЗМЛСМХЕСБМРДЯФЕТТШЛПТЛУЕСУЖРЪЮАОСЭЛХЧНЦГЯЖТДВФДТЪСТДЮФЦДЩМЕУЬЩФССЪРТТЛХР ЮУУСУЖПБЖШ СФЕП ШУЕГМЕФ АФЛТЛУЕСШФИЪЧЩДЬЯКЕСЩЛДЦЪСКГМЛЧСЩОДЧРОТМХЕМФЯРДУЪСАЙЪЙУСПФХ РЖД РУЕЬЪЕП ШУЕГМЕТЧЛЗ ЭМЕФДЭШЕ*

**Полученный результат:**

Исходный текст:

ему показалось что он вошел в холодный облицованный мрамором склеп после того как зашла луна непроницаемый мрак ни намека на залитый серебряным сиянием мир за окном окна плотно закрыты и комната похожа на могилу куда не долетает ни единый звук большого города однако комната не была пуста

Гамма = **СЛЕД**

**Описание работы.**

1. Суть методики взлома
2. Описание алгоритма
3. Вывод программы
4. Код программы
5. **Суть методики взлома.**

Для нахождения исходного текста и ключа шифра мною был применен автоматизированный метод анализа. Алгоритм поиска ключа и восстановления текста реализован на языке Python, в виде исполняемого модуля.

1. **Описание алгоритма**

* Время выполнения: < 60 сек (для текста < 3000 символов и длины гаммы < 20 символов).
* Потребляемая память: < 10 Мб (для текста > 3000 символов будет пропорционально расти).
* Место на диске без учета интерпретатора: 7 Кб.

На вход процедуры в качестве основных параметров поступает зашифрованный текст и алфавит текста (например, 32 буквы + пробел).

Дополнительно можно указать максимальную длину гаммы (по умолчанию = 20 символов), а также количество выводимых вариантов расшифровки на одну длину гаммы. Они удобны при дальнейшем анализе, если автоматический метод не даст удовлетворительных результатов.

crack(cipher, alphabet, maxGammaLength=10, maxVariants=3, moreInfo = True)

В процедуре первым этапом выполняется вычисление индекса совпадений групп текста для каждой длины гаммы, от 1 до максимальной. Результаты выводятся на экран.

Индекс совпадения вычисляется по формуле:

C:\Users\Artem\Desktop\f267c603ebb73381dbd1c4cfea3cba09.png

Индексы совпадения символов, находящиеся в интервале от 0,3 до 0,5, говорят о низкой вероятности совпадения символов в строке, а значит, о низкой вероятности применения гаммы соответствующей длины. Чем выше закономерность появления одинаковых символов, тем наоборот, выше вероятность, что гамма имела указаную периодичность и одинаковые буквы текста шифровались одинаковыми буквами гаммы.

Затем отсеиваются длины гаммы, величина индекса совпадений которых не превышает среднего значения. На графике мы наблюдаем два ярко выраженных пика. Таким образом, для дальнейшего анализа выбираются только "пиковые" значения. В нашем примере ими являются длины гаммы [4, 8], индекс совпадения которых максимален.

Имея предполагаемую длину гаммы, можно заняться подбором отдельных её символов. Для этого по очереди подставляем все символы алфавита в каждую позицию гаммы, производим гаммирование и вычисляем энтропию получившегося фрагмента текста. Принцип сортировки подходящих символов таков, что чем меньше энтропия получившегося текста, тем выше вероятность символа-кандидата на место в определённой позиции гаммы. Для гаммы длиной 4 симв. и алфавита мощностью 33 символа алгоритм производит 4 \* 33 итераций.

Most probable candidates for symbol 1 : с, р, п, о, ь, м, ч, ъ, щ, н, э, ц...

Most probable candidates for symbol 2 : л, к, й, м, ф, у, и, ч, ж, ш, з, р...

Most probable candidates for symbol 3 : е, д, г, в, б, а, ж, л, р, , к, о...

Most probable candidates for symbol 4 : д, г, в, б, а, , п, я, о, р, н, с...

Всё, что осталось произвести в рамках автоматизации, это вывести первые нескольно текстов с наименьшей энтропией, полученные в результате гаммирования. Символы для гаммы были подготовлены и отсортированы на предыдущем шаге. В нашем примере программа выводит первые 3 варианта расшифровки и соответствующие им скомбинированные гаммы. Для этого потребовалось расшифровать текст 24 кол-во раз (используя по 2 символа для каждой позиции гаммы из списка кандидатов) и вывести варианты с наименьшей энтропией.

Most probable text variants:

Gamma ='след'; text = ему показалось что он вошел в холодный облицован...

Gamma ='слег'; text = емуапокбзалпсь што пн впшелав хплодоый пбличовао...

Gamma ='слдд'; text = емф полазамосьачтоаон гошем в цолоеныйаоблйцовбн...

Иногда, даже самый оптимальный с точки зрения программы вариант не всегда является окончательным ответом, и в этом случае придется прибегнуть к ручному поиску и корректировке символов гаммы. Благо, этот процесс не займет много времени и сил. Большую часть работы программа всё же сделала за нас.

В нашем примере программа «ошиблась» всего на один символ гаммы (вместо «тнже» или «тэже» следует применить «тоже».

Итого, полное время расшифровки, включая ручную корректировку, составляет порядка нескольких минут.

1. **Вывод программы:**

G:\Programs\Python27\python.exe G:/Docs/text/Учеба/Криптология/Lab2/vigenere\_cracker/vigenere\_cracker.py

|Cipher| = 288 symbols

|Alphabet| = 33 symbols

Max |Gamma| is set to 10

Gamma length = 1 ; Index of coincidence = 0.0410133565621

Gamma length = 2 ; Index of coincidence = 0.0434634809635

Gamma length = 3 ; Index of coincidence = 0.0385233918129

Gamma length = 4 ; Index of coincidence = 0.0685641627543

Gamma length = 5 ; Index of coincidence = 0.0432978999222

Gamma length = 6 ; Index of coincidence = 0.0429964539007

Gamma length = 7 ; Index of coincidence = 0.0366351418616

Gamma length = 8 ; Index of coincidence = 0.0700396825397

Gamma length = 9 ; Index of coincidence = 0.0344982078853

Gamma length = 10 ; Index of coincidence = 0.0438970990695

Most probable gamma length:

[4, 8]

|Gamma| = 4:

Most probable candidates for symbol 1 : с, р, п, о, ь, м, ч, ъ, щ, н, э, ц...

Most probable candidates for symbol 2 : л, к, й, м, ф, у, и, ч, ж, ш, з, р...

Most probable candidates for symbol 3 : е, д, г, в, б, а, ж, л, р, , к, о...

Most probable candidates for symbol 4 : д, г, в, б, а, , п, я, о, р, н, с...

Most probable text variants:

Gamma ='след'; text = ему показалось что он вошел в холодный облицованный мрамо...

Gamma ='слег'; text = емуапокбзалпсь што пн впшелав хплодоый пбличоваоныйамрано...

Gamma ='слдд'; text = емф полазамосьачтоаон гошем в цолоеныйаоблйцовбннык мрбмо...

|Gamma| = 8:

Most probable candidates for symbol 1 : с, р, п, ч, т, о, ц, ь, ъ, щ, н, м...

Most probable candidates for symbol 2 : л, к, й, м, и, ф, у, з, ж, р, т, с...

Most probable candidates for symbol 3 : е, д, г, в, б, а, , я, о, н, л, к...

Most probable candidates for symbol 4 : д, г, в, б, п, о, н, а, м, , с, р...

Most probable candidates for symbol 5 : с, р, я, ю, п, , э, ь, о, м, ы, ъ...

Most probable candidates for symbol 6 : л, ш, ч, к, х, ц, й, ф, у, ж, м, и...

Most probable candidates for symbol 7 : е, д, г, ж, в, р, б, л, к, п, о, а...

Most probable candidates for symbol 8 : д, г, в, , б, а, я, е, ю, р, п, э...

Most probable text variants:

Gamma ='следслед'; text = ему показалось что он вошел в холодный облицованный мрамо...

Gamma ='слегслед'; text = емуапоказалпсь что пн вошелав холодоый обличованныйамрамо...

Gamma ='слдгслед'; text = емфапоказампсь чтоапн вошемав холоеоый облйчованныкамрамо...

Process finished with exit code 0

1. **Код программы**

#coding: UTF-8

import math

def frequences(str):

"""

Возвращает словарь, состоящий из частот появления отдельных символов в данной строке.

Аргументы:

str - строка для анализа.

"""

dict = {}

for c in str:

if c in dict:

dict[c] += 1

else:

dict[c] = 1

for c in dict:

dict[c] = 1.0 \* dict[c] / len(str)

return dict

def entropy(string, alphabet):

"""

Возвращает энтропию текста.

Аргументы:

string - анализируемая строка

alphabet - коллекция символов алфавита

"""

freq = frequences(string)

e = 0

for ch in freq:

e += (alphabet.index(ch) + 1) \* freq[ch]

return e

def indexOfCoincidence(text, alphabet):

"""

Возвращает индекс совпадения (вероятность совпадения двух произвольных символов в тексте).

Мин. значение: 1 / len(text) (для случайного текста)

Макс. значение: 1

Аргументы:

text - текст, на основе которого вычисляется индекс

alphabet - коллекция символов, которая включает все символы текста

"""

counts = {}

for a in alphabet:

counts[a] = 0

for ch in text:

counts[ch] += 1

index = 0

for c in counts:

index += (counts[c] \* (counts[c] - 1) \* 1.0) / (len(text) \* (len(text) - 1) \* 1.0)

return index

def extractSymbols(str, period, shift=0):

"""

Возвращает строку, состоящую из каждого i-го символа исходной строки.

Аргументы:

str - исходная строка

period - период повторений

shift - порядковый номер символа, с которого начнется отсчет.

"""

ret = ""

i = shift

while i < len(str):

ret += str[i]

i += period

return ret

def applyGamma(text, gamma, alphabet, decode=False):

"""

Складывает гамму с исходным текстом и возвращает результат. Является реализацией шифра Виженера.

Аргументы:

text - исходный текст

gamma - ключ шифрования

alphabet - список символов, на основе которого применяется криптографическое преобразование

decode - лог. значение; True означает сложение гаммы из текстом, False - вычитание.

"""

ret = ""

for i, ch in enumerate(text):

if decode:

ret += alphabet[(alphabet.index(ch) + len(alphabet) - alphabet.index(gamma[i % len(gamma)])) % len(alphabet)]

else:

ret += alphabet[(alphabet.index(ch) + alphabet.index(gamma[i % len(gamma)])) % len(alphabet)]

return ret

def crack(cipher, alphabet, maxGammaLength = 20, maxVariants = 3, moreInfo = False):

indexes = {}

maxGammaLength = min(maxGammaLength, len(cipher) - 1)

print "|Cipher| =", len(cipher), "symbols\n"\

"|Alphabet| =", len(alphabet), "symbols\n"\

"Max |Gamma| is set to", maxGammaLength, "\n"

for length in range(1, maxGammaLength + 1): # длина гаммы (от 1 до макс. включительно)

index = 0 # индекс совпадений

for i in range(length): # считаем для каждой подгруппы симовлов

sequence = extractSymbols(cipher, period=length, shift=i)

index += indexOfCoincidence(sequence, alphabet)

index /= 1.0 \* length # находим средний индекс для всех подгрупп

indexes[length] = index

if (moreInfo): print "Gamma length =", length, "; Index of coincidence =", index

mediumIndex = 1.0 \* sum(indexes.values()) / len(indexes) # средний индекс для всех длин гамм

gammaLengths = []

for l in indexes: # выбираем только те длины гаммы, индексы совпадений

if indexes[l] > mediumIndex: gammaLengths.append(l) # для которых выше среднего

# опционально: останавливаемся на минимальной длине гаммы

print "\nMost probable gamma length:\n", gammaLengths

for length in gammaLengths:

print "\n|Gamma| = %d:" % length

probableGamma = []

for l in range(length):

group = extractSymbols(cipher, length, l)

symbols = {}

for ch in alphabet:

val = entropy(applyGamma(group, ch, alphabet, True), alphabet)

symbols[ch] = val

print "\tEntropy of group for gamma =", ch, ":", val

charList = sorted(symbols, key=symbols.get, reverse=False)

if (moreInfo): print "\tMost probable candidates for symbol", l+1, ":", ", ".join(charList)

probableGamma.append(charList) # добавляем массив всех символов-кандидатов

variantsCount = 2\*\*(len(probableGamma)) # выведем два варианта на каждую букву гаммы

textVariants = {} # словарь текстов по ключу энтропии

for variant in range(variantsCount):

gamma = ""

for index in range(len(probableGamma)):

gamma += probableGamma[index][(variant >> index) % 2]

result = applyGamma(cipher, gamma, alphabet, True)

e = entropy(result, alphabet)

textVariants[e] = (gamma, result[:80])

print "\n\tMost probable text variants:"

for e in sorted(textVariants.keys())[:maxVariants]: # выберем тексты с наименьшей энтропией

print "\tGamma ='%s'; text = %s..." % textVariants[e]

cipher = open(u"../Задачи лр №2/5.txt").read().replace("\n", "").decode("UTF-8")

alphabet = open(u"data/alphabet.txt").read().decode("CP1251")

crack(cipher, alphabet, maxGammaLength=10, maxVariants=3, moreInfo = True)