Министерство образования и науки Российской Федерации
Федеральное государственное автономное образовательное
учреждение высшего образования

КАЗАНСКИЙ (ПРИВОЛЖСКИЙ) ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ

ИНСТИТУТ ФИЗИКИ

Лабораторная работа №4

Реализация и исследование свойств алгоритмов хэширования

Выполнил
Глазков Андрей
студент 4 курса
группы 06-952

Казань – 2023

Цель работы

Изучить и исследовать свойства криптографического хеширования на примере алгоритма MD4, а также его уязвимости с помощью процедур поиска коллизий и прообразов.

Ход работы:

1. Написали скрипт на python, вычисляющий с помощью алгоритма MD4 хэш-код строки, введенной пользователем. (Листинг в приложении)

```
→  Lab4_py git:(main) python lab4.py
task = 1
input str:
hash: 31d6cfe0d16ae931b73c59d7e0c089c0
→  Lab4_py git:(main) python lab4.py
task = 1
input str: abc
hash: a448017aaf21d8525fc10ae87aa6729d
→  Lab4_py git:(main) python lab4.py
task = 1
input str: ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789
hash: 043f8582f241db351ce627e153e7f0e4
```

Сравнили с теоретическими значениями из методички:

Примеры хэш-кодов для разных тестовых строк

MD4("") = 31d6cfe0d16ae931b73c59d7e0c089c0

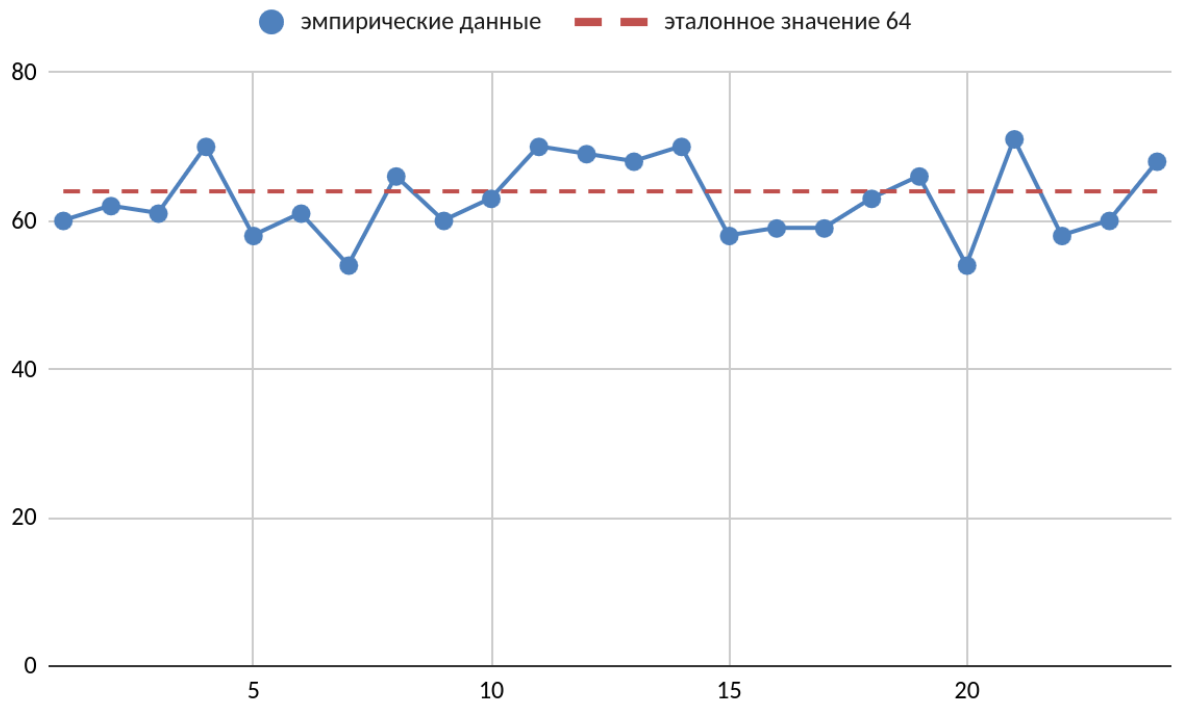
MD4("abc") = a448017aaf21d8525fc10ae87aa6729d

MD4("ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789") = 043f8582f241db351ce627e153e7f0e4

2. Убедились в наличии лавинного эффекта.

```
→  Lab4_py git:(main) python lab4.py
task = 2
input str: abc
start bin: 011000010110001001100011     modificatuin 1 bit left: 111000010110001001100011
start hash:a448017aaf21d8525fc10ae87aa6729d     new hash: b700cd1b00dded0d50ab0843e9d225c
Number of changes in hash: 58
→  Lab4_py git:(main) python lab4.py
task = 2
input str: qwerty
start bin: 011100010111011101100101011100100111010001111001     modificatuin 1 bit left: 111100010111011101100101011100100111010001111001
start hash:2a4bbeffd06c016ab4134cc7963496d2     new hash: 8fedf02c10678265b657edb24420743d
Number of changes in hash: 57
```

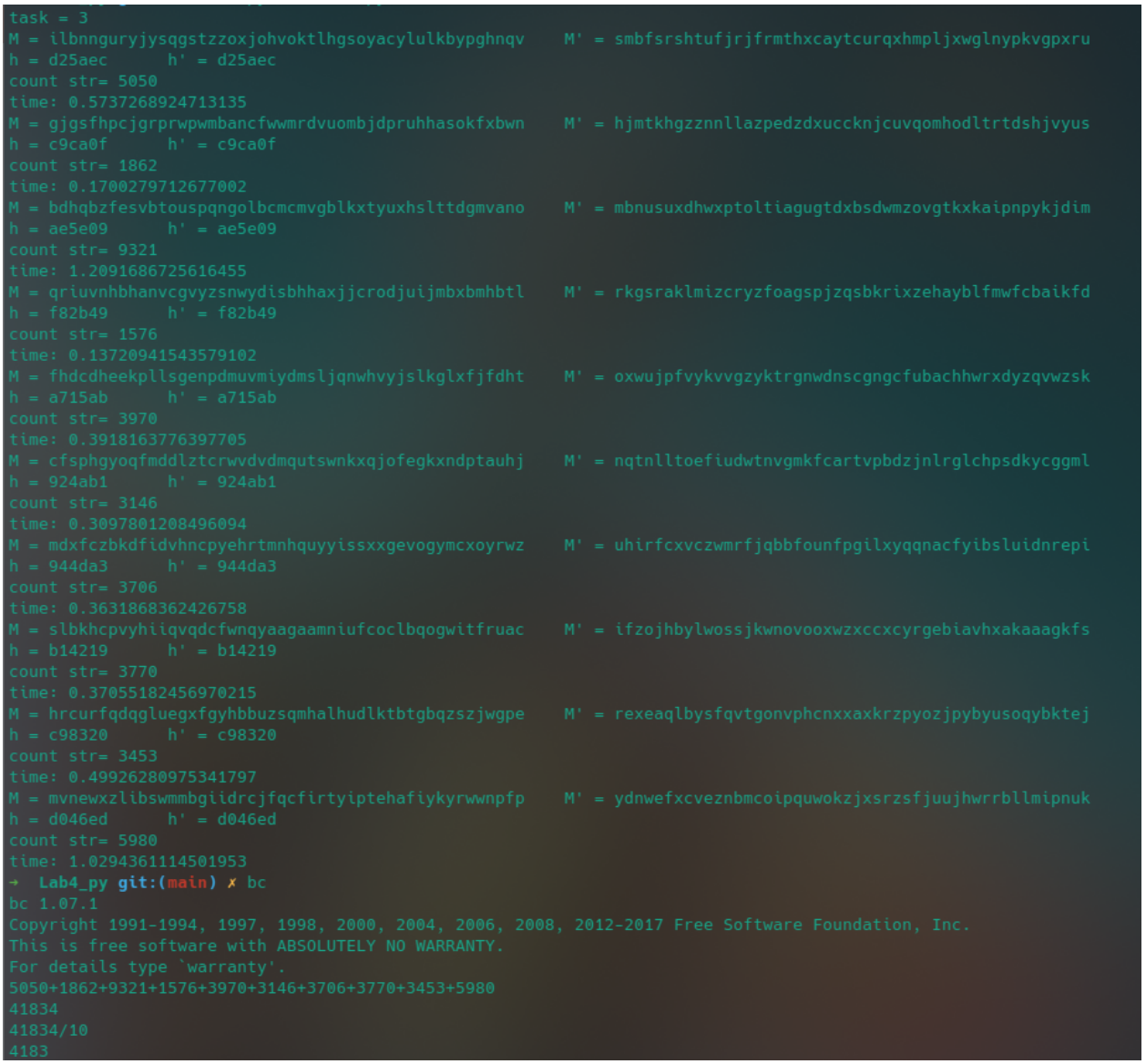
Построили график зависимости количества битов, изменяющихся в хэше от количества измененных битов в исходном сообщении



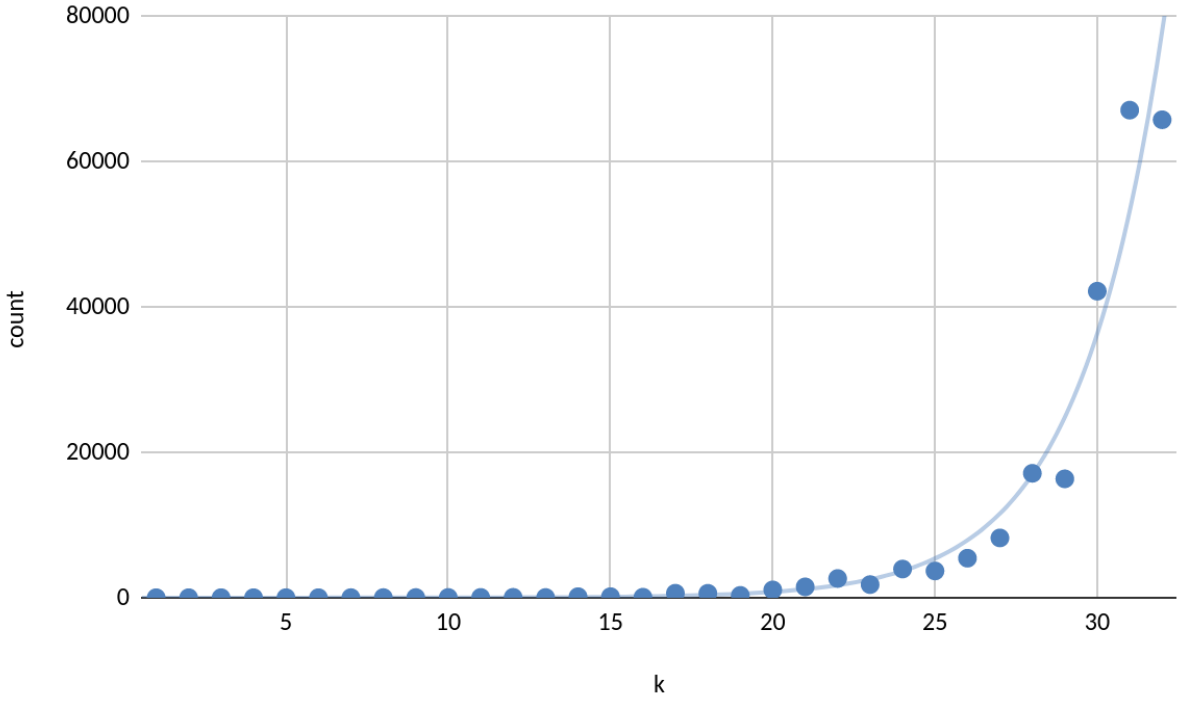
По графику можно сказать, что данная хеш функция достаточно неплоха, среднее значение по всем 24 точкам равно 62, что не сильно отличается от эталонного 64. (исследование проводили на строке “`donpython`” k=1..24)

3. Написали программу, реализующую процедуру поиска коллизий алгоритма MD4 (приложение).

Для сбора статистики выполнили цикл из 10 замеров с данными (k=24, L=48). Получили, что на поиск “слова”, которое выдаст тот же хеш в первых 24 битах понадобится 4183, что достаточно близко к теоретическому значения в 4096.



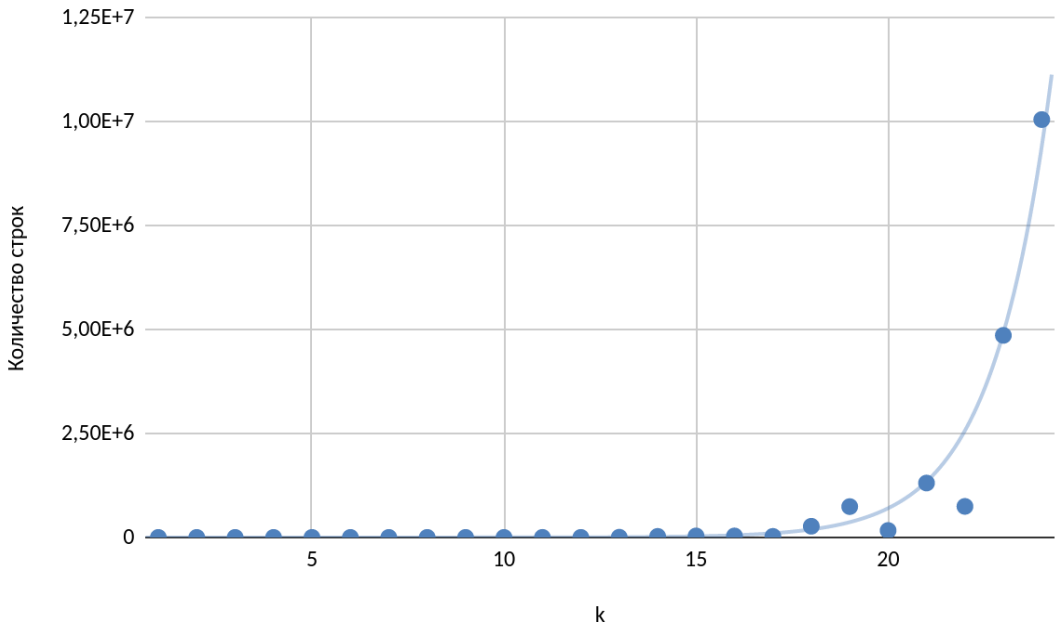
Построили зависимость количества сгенерированных строк от длины хэш-кода.



4. Программно реализовать поиск прообраза для заданного хэш-кода заданной строки (приложение)

```
→  Lab4_py git:(main) ✗ python lab4.py
task = 4
pass M: Say the password and come in
k= 16
hash M: ddfc
M`: dxhrrmxspiynecgnnsirbukkfjqd
hash: ddfc
number of attempts: 76602
```

Построили зависимость количества сгенерированных строк от длины хэш-кода при поиске прообраза



5. Исследовали влияние инициализирующего состояния служебных регистров на качество хэширования.

Изменили служебные регистры

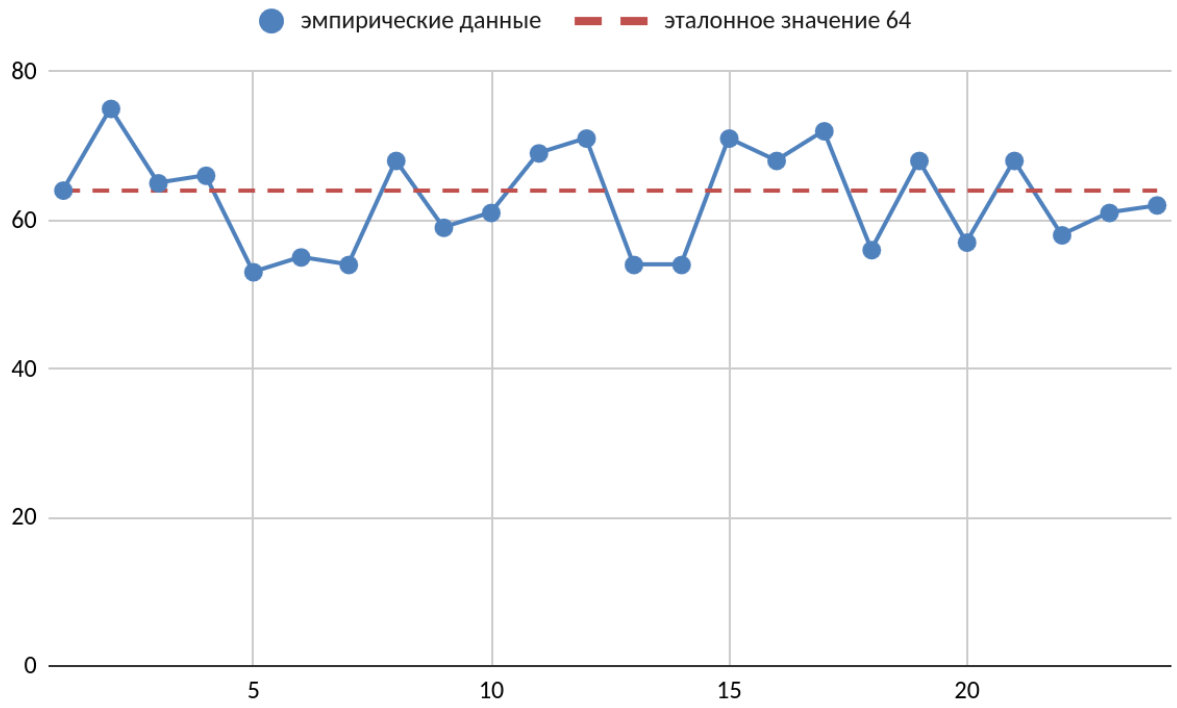
```
# orig    A, B, C, D = 0x67452301, 0xefcdab89, 0x98badcfe, 0x10325476
    A, B, C, D = 0x67452302, 0xefcdab99, 0x98badffe, 0x10325476
```

Проверили протестировали эффективность лавинного эффекта. Результаты отобразили в графике.

По графику видно, эффективность хеширования снижена, относительно изначального случая, так как при k=15..20 увеличен разброс относительно эталонного значения.

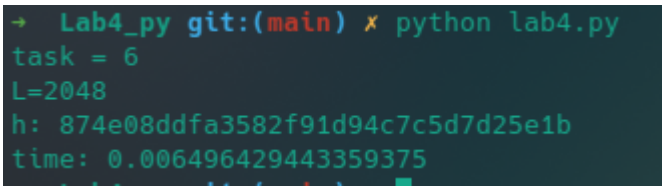
6. Исследовали зависимость качества хеширования от количества раундов при хешировании.

Для этого закомментировали строчки с 3 раундом и провели исследование лавинного эффекта.

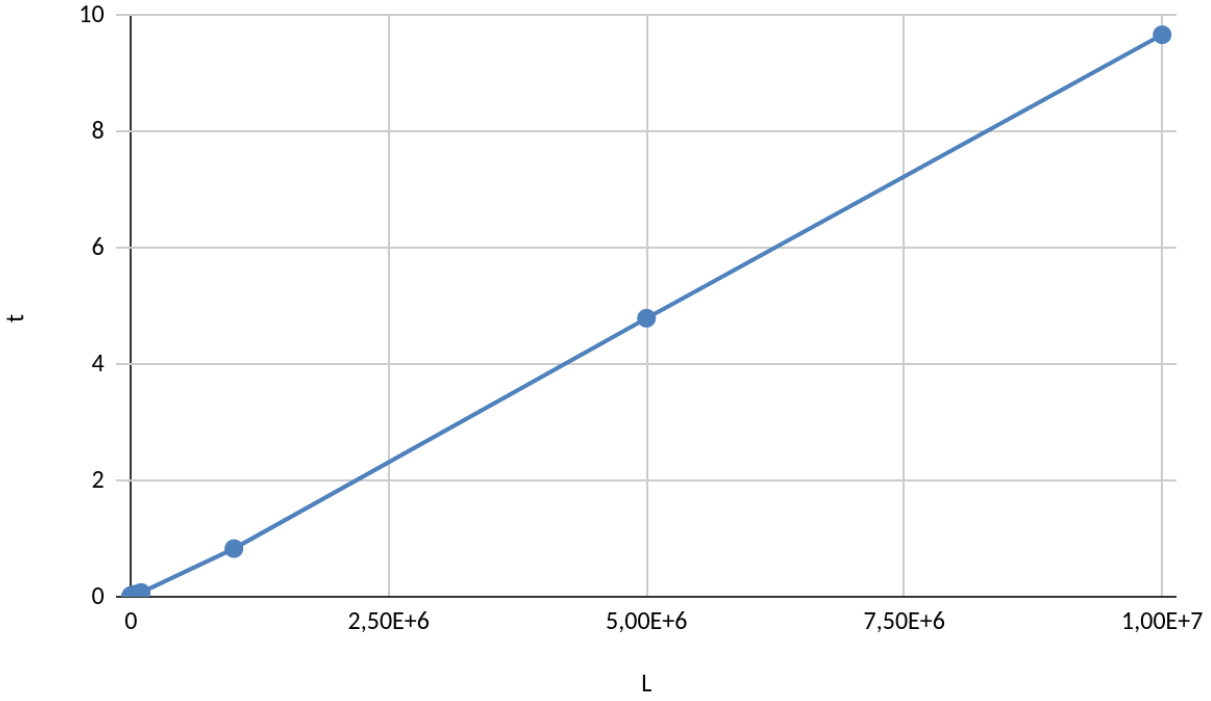


Nav для усеченного 2-раундового алгоритма будет больше, чем для полного 3-раундового алгоритма. Это связано с тем, что третий раунд является важной частью алгоритма, где происходит окончательное перемешивание данных, и его отсутствие может снизить качество хэширования.

7. Выявить зависимость времени хэширования от длины сообщения.



Построили зависимость длительности хэширования от длины сообщения



Время генерации хэш-значения в MD4 растет линейно на участке [100..1000000] с увеличением длины сообщения L, за исключением дополнительного времени, которое может потребоваться для дополнения сообщения до кратности 512 битам.

Вывод:

В ходе выполнения лабораторной работы была разработана программа, которая реализует 128-битовую хэш-функцию MD4. Также проверена теория о лавинном изменение хэш-функции, проведен анализ уязвимости с помощью процедур поиска коллизий и прообразов.

Приложение.

```
import string

import time

import random


def padTo512bits(binary):

	binaryLen = bin(len(binary))[2:]

	binary += '1'

	mod = len(binary) % 512

	while mod != 448:

	binary += '0'

	mod = len(binary) % 512

	binary = ''.join(splitByElements(binary, 8)[::-1])

	binary = ''.join(splitByElements(binary, 32)[::-1])

	binaryLen = '0' * (64 - len(binaryLen)) + binaryLen

	binaryLen = ''.join(splitByElements(binaryLen, 32)[::-1])

	binary += binaryLen

	return binary


def splitByElements(string, Range):

	return [string[i:i+Range] for i in range(0, len(string), Range)]


def stringToBinary(str):

	return ''.join(format(ord(j), '08b') for j in str)


def leftRot(a, s):

	return ((a << s) | (a >> (32 - s)))


def  F(x, y, z):

	return (x & y) | (~x & z)


def  G(x, y, z):

	return (x & y) | (x & z) | (y & z)
```

```
def  H(x, y, z):
       return x ^ y ^ z


def R1(A, B, C, D, Xk, s):
       return leftRot((A + F(B, C, D) + Xk) % (2**32), s)


def R2(A, B, C, D, Xk, s):
       return leftRot((A + G(B, C, D) + Xk + 0x5A827999) % (2**32), s)


def R3(A, B, C, D, Xk, s):
       return leftRot((A + H(B, C, D) + Xk + 0x6ED9EBA1) % (2**32), s)


def MD4(R, fl):
       N = len(R)


       A, B, C, D = 0x67452301, 0xefcdab89, 0x98badcfe, 0x10325476
       # A, B, C, D = 0x67452302, 0xefcdab99, 0x98badffe, 0x10325476
       X = [0] * 16


       for i in range(0, N):
       X = [int(x, 2) for x in splitByElements(R[i], 32)]


       AA, BB, CC, DD = A, B, C, D
       A = R1(A, B, C, D, X[0], 3); D = R1(D, A, B, C, X[1], 7); C = R1(C, D, A, B,
X[2], 11); B = R1(B, C, D, A, X[3], 19);
       A = R1(A, B, C, D, X[4], 3); D = R1(D, A, B, C, X[5], 7); C = R1(C, D, A, B,
X[6], 11); B = R1(B, C, D, A, X[7], 19);
       A = R1(A, B, C, D, X[8], 3); D = R1(D, A, B, C, X[9], 7); C = R1(C, D, A, B,
X[10], 11); B = R1(B, C, D, A, X[11], 19);
       A = R1(A, B, C, D, X[12], 3); D = R1(D, A, B, C, X[13], 7); C = R1(C, D, A, B,
X[14], 11); B = R1(B, C, D, A, X[15], 19);


       A = R2(A, B, C, D, X[0], 3); D = R2(D, A, B, C, X[4], 5); C = R2(C, D, A, B,
X[8], 9); B = R2(B, C, D, A, X[12], 13);
       A = R2(A, B, C, D, X[1], 3); D = R2(D, A, B, C, X[5], 5); C = R2(C, D, A, B,
X[9], 9); B = R2(B, C, D, A, X[13], 13);
```

```
	A = R2(A, B, C, D, X[2], 3); D = R2(D, A, B, C, X[6], 5); C = R2(C, D, A, B, X[10], 9); B = R2(B, C, D, A, X[14], 13);

	A = R2(A, B, C, D, X[3], 3); D = R2(D, A, B, C, X[7], 5); C = R2(C, D, A, B, X[11], 9); B = R2(B, C, D, A, X[15], 13);



	# A = R3(A, B, C, D, X[0], 3); D = R3(D, A, B, C, X[8], 9); C = R3(C, D, A, B, X[4], 11); B = R3(B, C, D, A, X[12], 15);

	# A = R3(A, B, C, D, X[2], 3); D = R3(D, A, B, C, X[10], 9); C = R3(C, D, A, B, X[6], 11); B = R3(B, C, D, A, X[14], 15);

	# A = R3(A, B, C, D, X[1], 3); D = R3(D, A, B, C, X[9], 9); C = R3(C, D, A, B, X[5], 11); B = R3(B, C, D, A, X[13], 15);

	# A = R3(A, B, C, D, X[3], 3); D = R3(D, A, B, C, X[11], 9); C = R3(C, D, A, B, X[7], 11); B = R3(B, C, D, A, X[15], 15);



	A = (A + AA) % (2**32)

	B = (B + BB) % (2**32)

	C = (C + CC) % (2**32)

	D = (D + DD) % (2**32)



	if fl: return ''.join(splitByElements(hex(A)[2:],2)[::-1])

	return ''.join(splitByElements(hex(A)[2:],2)[::-1]) +
''.join(splitByElements(hex(B)[2:],2)[::-1]) +
''.join(splitByElements(hex(C)[2:],2)[::-1]) +
''.join(splitByElements(hex(D)[2:],2)[::-1])



def getHash(binary,flag):

	binaryBy512 = padTo512bits(binary)

	binaryBlocks = splitByElements(binaryBy512, 512)

	hash = MD4(binaryBlocks, flag)

	return hash



def Task2(binary,k,hash1):

	binaryChange = list(binary)

	for i in range(0,k):

	binaryChange[i] = str(int(binary[i])^1)

	binaryChange = ''.join(binaryChange)

	print(f'start bin: {binary}\tmodificatuin {k} bit left: {binaryChange}')

	hash2 = getHash(binaryChange,0)
```

```
        print(f'start hash:{hash1}\tnew hash: {hash2}')
        changes = str(bin(int(hash1,16)^int(hash2,16))).count('1')
        print(f'Number of changes in hash: {changes}')

def generateString(length):
        return ''.join(random.choices(string.ascii_lowercase, k=length))

def Task3(k, L):
        hash_list=list()
        words=list()
        word=generateString(L)
        start = time.time()
        hash=hex(int(bin(int(getHash(stringToBinary(word),1),base=16))[2:k+2],2))[2:]
        while True :
        if hash in hash_list and word not in words:
                break
        hash_list.append(hash)
        words.append(word)
        word=generateString(L)
        hash=hex(int(bin(int(getHash(stringToBinary(word),1),base=16))[2:k+2],2))[2:]
        for i in range(len(hash_list)):
        if hash==hash_list[i]:
                collision_hash=hash_list[i]
                collision_word=words[i]
                break
        print (f'M = {word}\tM\' = {collision_word}')
        print (f'h = {hash}\th\' = {collision_hash}')
        print(f"count str= {len(words)}")
        print(f'time: {time.time()-start}')

def Task4(password, k):
        #  password=input('pass M: ')
        #  k=int(input('k= '))
```

```
hash_pass=hex(int(bin(int(getHash(stringToBinary(password),1),16))[2:k+2],2))[2:]
        print(f'hash M: {hash_pass}')
        kol=0
        while True:
        word=generateString(len(password))
        hash=hex(int(bin(int(getHash(stringToBinary(word),1),16))[2:k+2],2))[2:]
        kol += 1
        if hash == hash_pass and password != word:
                print(f"M`: {word}\nhash: {hash}")
                print(f"number of attempts: {kol}")
                break


if __name__ == '__main__':
        task = (int(input("task = ")))
        if (task == 1):
        binary = stringToBinary(input('input str: '))
        hash = getHash(binary,0)
        print(f'hash: {hash}')
        elif (task == 2):
        # binary = stringToBinary(input('input str: '))
        binary = stringToBinary('donpython')
        hash = getHash(binary,0)
        for i in range(24):
                i+=1
        # count_sym = int(input('count add 1: '))
                print(i)
                Task2(binary, i ,hash)
        elif (task == 3):
        for i in range(32):
                i+=1
                print(i)
                Task3(i, 48)
        elif (task == 4):
```

```
password = 'Say the password and come in'
for i in range(24):
       i += 1
       Task4(password, i)
elif (task == 6):
L = int(input('L='))
w = generateString(L)
b = stringToBinary(w)
start = time.time()
hash = getHash(b,0)
stop = time.time()
print(f'h: {hash}')
print(f'time: {stop - start}')
```