**Московский авиационный институт**

**(Национальный исследовательский университет)**

Факультет: «Информационные технологии и прикладная математика»

**Лабораторная работа № 3**

по курсу «Криптография»

Студент: Алексюнина Юлия

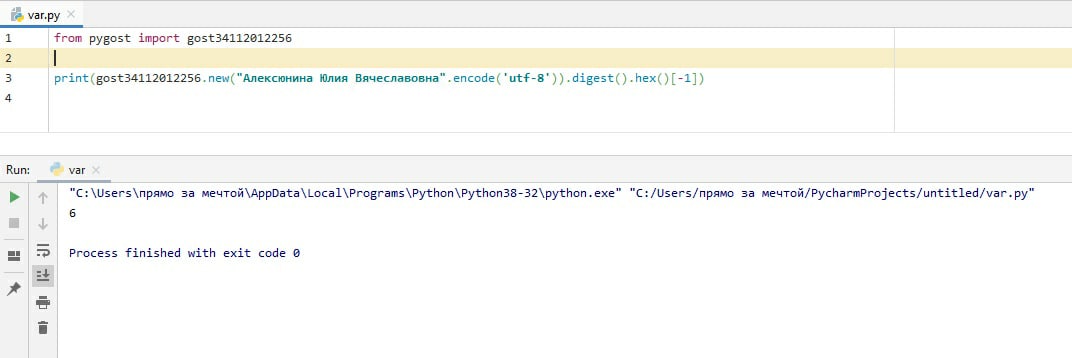
Группа: 8О-307Б

Преподаватель: Борисов А.В.

Оценка:

Москва, 2020

Сначала был получен номер варианта:



Вариант 6: SHA-2

Мною был выбран алгоритм SHA-256 ввиду его распространенности.

Исходный код:

sha256.h

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Filename: sha256.h

\* Author: Brad Conte (brad AT bradconte.com)

\* Copyright:

\* Disclaimer: This code is presented "as is" without any guarantees.

\* Details: Defines the API for the corresponding SHA1 implementation.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#ifndef SHA256\_H

#define SHA256\_H

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* HEADER FILES \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <stddef.h>

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* MACROS \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define SHA256\_BLOCK\_SIZE 32 // SHA256 outputs a 32 byte digest

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* DATA TYPES \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

typedef unsigned char BYTE; // 8-bit byte

typedef unsigned int WORD; // 32-bit word, change to "long" for 16-bit machines

typedef struct {

BYTE data[64];

WORD datalen;

unsigned long long bitlen;

WORD state[8];

} SHA256\_CTX;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* FUNCTION DECLARATIONS \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void sha256\_init(SHA256\_CTX \*ctx);

void sha256\_update(SHA256\_CTX \*ctx, const BYTE data[], size\_t len);

void sha256\_final(SHA256\_CTX \*ctx, BYTE hash[]);

#endif // SHA256\_H

sha256.c

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \* Filename: sha256.c

\* Author:Brad Conte (brad AT bradconte.com)

\* Copyright:

\* Disclaimer: This code is presented "as is" without any guarantees.

\* Details:Implementation of the SHA-256 hashing algorithm.

SHA-256 is one of the three algorithms in the SHA2 specification. The others, SHA-384 and SHA-512, are not offered in this implementation.

Algorithm specification can be found here:

\* http://csrc.nist.gov/publications/fips/fips180-2/fips180-2withchangenotice.pdf

This implementation uses little endian byte order.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* HEADER FILES \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <stdlib.h>

#include <memory.h>

#include "sha256.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* MACROS \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define ROTLEFT(a,b) (((a) << (b)) | ((a) >> (32-(b))))

#define ROTRIGHT(a,b) (((a) >> (b)) | ((a) << (32-(b))))

#define CH(x,y,z) (((x) & (y)) ^ (~(x) & (z)))

#define MAJ(x,y,z) (((x) & (y)) ^ ((x) & (z)) ^ ((y) & (z)))

#define EP0(x) (ROTRIGHT(x,2) ^ ROTRIGHT(x,13) ^ ROTRIGHT(x,22))

#define EP1(x) (ROTRIGHT(x,6) ^ ROTRIGHT(x,11) ^ ROTRIGHT(x,25))

#define SIG0(x) (ROTRIGHT(x,7) ^ ROTRIGHT(x,18) ^ ((x) >> 3))

#define SIG1(x) (ROTRIGHT(x,17) ^ ROTRIGHT(x,19) ^ ((x) >> 10))

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* VARIABLES \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

static const WORD k[64] = {

0x428a2f98,0x71374491,0xb5c0fbcf,0xe9b5dba5,0x3956c25b,0x59f111f1,0x923f82a4, 0xab1c5ed5,

0xd807aa98,0x12835b01,0x243185be,0x550c7dc3,0x72be5d74,0x80deb1fe,0x9bdc06a7, 0xc19bf174,

0xe49b69c1,0xefbe4786,0x0fc19dc6,0x240ca1cc,0x2de92c6f,0x4a7484aa,0x5cb0a9dc, 0x76f988da,

0x983e5152,0xa831c66d,0xb00327c8,0xbf597fc7,0xc6e00bf3,0xd5a79147,0x06ca6351, 0x14292967,

0x27b70a85,0x2e1b2138,0x4d2c6dfc,0x53380d13,0x650a7354,0x766a0abb,0x81c2c92e, 0x92722c85,

0xa2bfe8a1,0xa81a664b,0xc24b8b70,0xc76c51a3,0xd192e819,0xd6990624,0xf40e3585, 0x106aa070,

0x19a4c116,0x1e376c08,0x2748774c,0x34b0bcb5,0x391c0cb3,0x4ed8aa4a,0x5b9cca4f, 0x682e6ff3,

0x748f82ee,0x78a5636f,0x84c87814,0x8cc70208,0x90befffa,0xa4506ceb,0xbef9a3f7, 0xc67178f2

};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* FUNCTION DEFINITIONS \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void sha256\_transform(SHA256\_CTX\* ctx, const BYTE data[], size\_t times)

{

WORD a, b, c, d, e, f, g, h, i, j, t1, t2, m[64];

for (i = 0, j = 0; i < 16; ++i, j += 4)

m[i] = (data[j] << 24) | (data[j + 1] << 16) | (data[j + 2] << 8) | (data[j + 3]);

for (; i < 64; ++i)

m[i] = SIG1(m[i - 2]) + m[i - 7] + SIG0(m[i - 15]) + m[i - 16];

a = ctx->state[0];

b = ctx->state[1];

c = ctx->state[2];

d = ctx->state[3];

e = ctx->state[4];

f = ctx->state[5];

g = ctx->state[6];

h = ctx->state[7];

for (i = 0; i < times; ++i) {

t1 = h + EP1(e) + CH(e, f, g) + k[i] + m[i];

t2 = EP0(a) + MAJ(a, b, c);

h = g;

g = f;

f = e;

e = d + t1;

d = c;

c = b;

b = a;

a = t1 + t2;

}

ctx->state[0] += a;

ctx->state[1] += b;

ctx->state[2] += c;

ctx->state[3] += d;

ctx->state[4] += e;

ctx->state[5] += f;

ctx->state[6] += g;

ctx->state[7] += h;

}

void sha256\_init(SHA256\_CTX\* ctx)

{

ctx->datalen = 0;

ctx->bitlen = 0;

ctx->state[0] = 0x6a09e667;

ctx->state[1] = 0xbb67ae85;

ctx->state[2] = 0x3c6ef372;

ctx->state[3] = 0xa54ff53a;

ctx->state[4] = 0x510e527f;

ctx->state[5] = 0x9b05688c;

ctx->state[6] = 0x1f83d9ab;

ctx->state[7] = 0x5be0cd19;

}

void sha256\_update(SHA256\_CTX\* ctx, const BYTE data[], size\_t len, size\_t

times)

{

WORD i;

for (i = 0; i < len; ++i) {

ctx->data[ctx->datalen] = data[i]; ctx->datalen++;

if (ctx->datalen == 64) {

sha256\_transform(ctx, ctx->data, times);

ctx->bitlen += 512;

ctx->datalen = 0;

}

}

}

void sha256\_final(SHA256\_CTX\* ctx, BYTE hash[], size\_t times) {

WORD i;

i = ctx->datalen;

// Pad whatever data is left in the buffer.

if (ctx->datalen < 56) {

ctx->data[i++] = 0x80; while (i < 56)

ctx->data[i++] = 0x00;

}

else {

ctx->data[i++] = 0x80; while (i < times) // !

ctx->data[i++] = 0x00; sha256\_transform(ctx, ctx->data, times); memset(ctx->data, 0, 56);

}

// Append to the padding the total message's length in bits and transform.

ctx->bitlen += ctx->datalen \* 8; ctx->data[63] = ctx->bitlen; ctx->data[62] = ctx->bitlen >> 8; ctx->data[61] = ctx->bitlen >> 16; ctx->data[60] = ctx->bitlen >> 24;

ctx->data[59] = ctx->bitlen >> 32; ctx->data[58] = ctx->bitlen >> 40; ctx->data[57] = ctx->bitlen >> 48; ctx->data[56] = ctx->bitlen >> 56; sha256\_transform(ctx, ctx->data, times);

// Since this implementation uses little endian byte ordering and SHA uses big endian,

// reverse all the bytes when copying the final state to the output hash.

for (i = 0; i < 4; ++i) {

hash[i] = (ctx->state[0] >> (24 - i \* 8)) & 0x000000ff;

hash[i + 4] = (ctx->state[1] >> (24 - i \* 8)) & 0x000000ff;

hash[i + 8] = (ctx->state[2] >> (24 - i \* 8)) & 0x000000ff;

hash[i + 12] = (ctx->state[3] >> (24 - i \* 8)) & 0x000000ff;

hash[i + 16] = (ctx->state[4] >> (24 - i \* 8)) & 0x000000ff;

hash[i + 20] = (ctx->state[5] >> (24 - i \* 8)) & 0x000000ff;

hash[i + 24] = (ctx->state[6] >> (24 - i \* 8)) & 0x000000ff;

hash[i + 28] = (ctx->state[7] >> (24 - i \* 8)) & 0x000000ff;

}

}

lab3.c

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* HEADER FILES \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <memory.h>

#include <string.h>

#include <stdbool.h>

#include "sha256.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* FUNCTION DEFINITIONS \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int sha256\_test(const BYTE text[], char filename[]) {

BYTE buf1[SHA256\_BLOCK\_SIZE];

BYTE buf2[SHA256\_BLOCK\_SIZE];

SHA256\_CTX ctx;

FILE\* f = fopen(filename, "wb");

if (f) {

size\_t text\_length = strlen(text);

sha256\_init(&ctx);

sha256\_update(&ctx, text, text\_length, 16); sha256\_final(&ctx, buf1, 16);

for (size\_t times = 17; times <= 64; times++) {

sha256\_init(&ctx);

sha256\_update(&ctx, text, text\_length, times); sha256\_final(&ctx, buf2, times);

size\_t cnt = 0;

for (int i = 0; i < SHA256\_BLOCK\_SIZE; i++) {

for (int j = 0; j < 8; j++) {

if (((buf1[i] << j) & 0x80) != ((buf2[i] << j) & 0x80)) {

cnt++;

}

}

}

fprintf(f, "%d %d\n", times, cnt);

strncpy(buf1, buf2, SHA256\_BLOCK\_SIZE);

buf1[SHA256\_BLOCK\_SIZE - 1] = '\0';

}

fflush(f);

fclose(f);

}

else {

printf("Can't open file.");

}

return 1;

}

int main()

{

// 2 chars

BYTE text1[] = { "ju" };

char filename1[] = { "C:/Users/hui/Desktop/diffs1.csv" }; sha256\_test(text1, filename1);

// 20 chars

BYTE text2[] = { "heygoodnewseveryone!" };

char filename2[] = { "C:/Users/hui/Desktop/diffs2.csv" }; sha256\_test(text2, filename2);

// 200 chars

BYTE text3[] = { "hello,darkness,myoldfriend/i'vecometotalkwithyouagain/becauseavisionsoftlycreeping/leftitsseedswhileiwassleeping/andthevisionthatwasplantedinmybrain/stillremains/withinthesoundofsilence/soundofsilence" };

char filename3[] = { "C:/Users/hui/Desktop/diffs3.csv" }; sha256\_test(text3, filename3);

return 0;

}

Исходный алгоритм был модифицирован таким образом, чтобы можно было задавать количество раундов. Введенное количество раундов должно быть не меньше 16, так как в раунде используется вспомогательный массив, который должен быть по крайней мере длины 16. Также количество раундов не должно превышать 64, так как алгоритм опирается на массив дробных частей кубических корней простых чисел k длиной 64. Теоретически, можно продолжить этот массив для большего количества раундов.

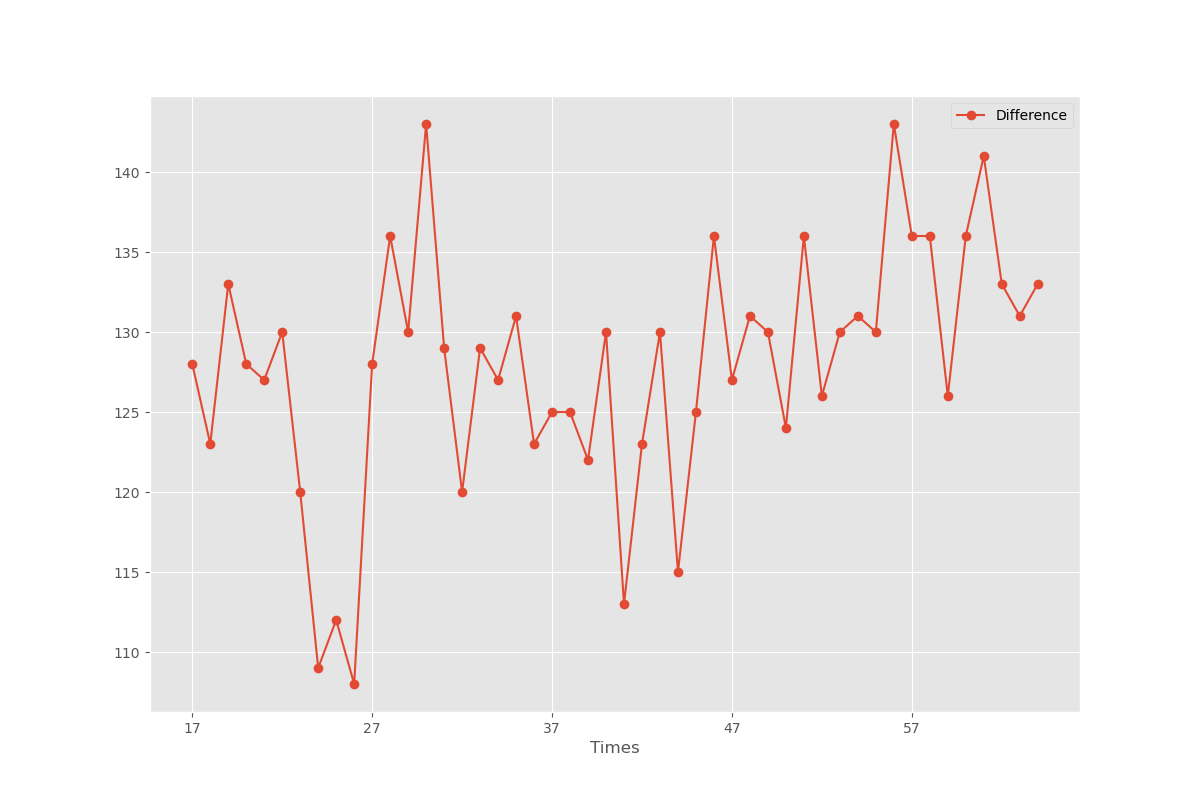
В программе были созданы файлы с разностями в битах для каждого раунда для трех тестовых строк.

После этого при помощи python были построены графики зависимости разности в битах от количества раундов:

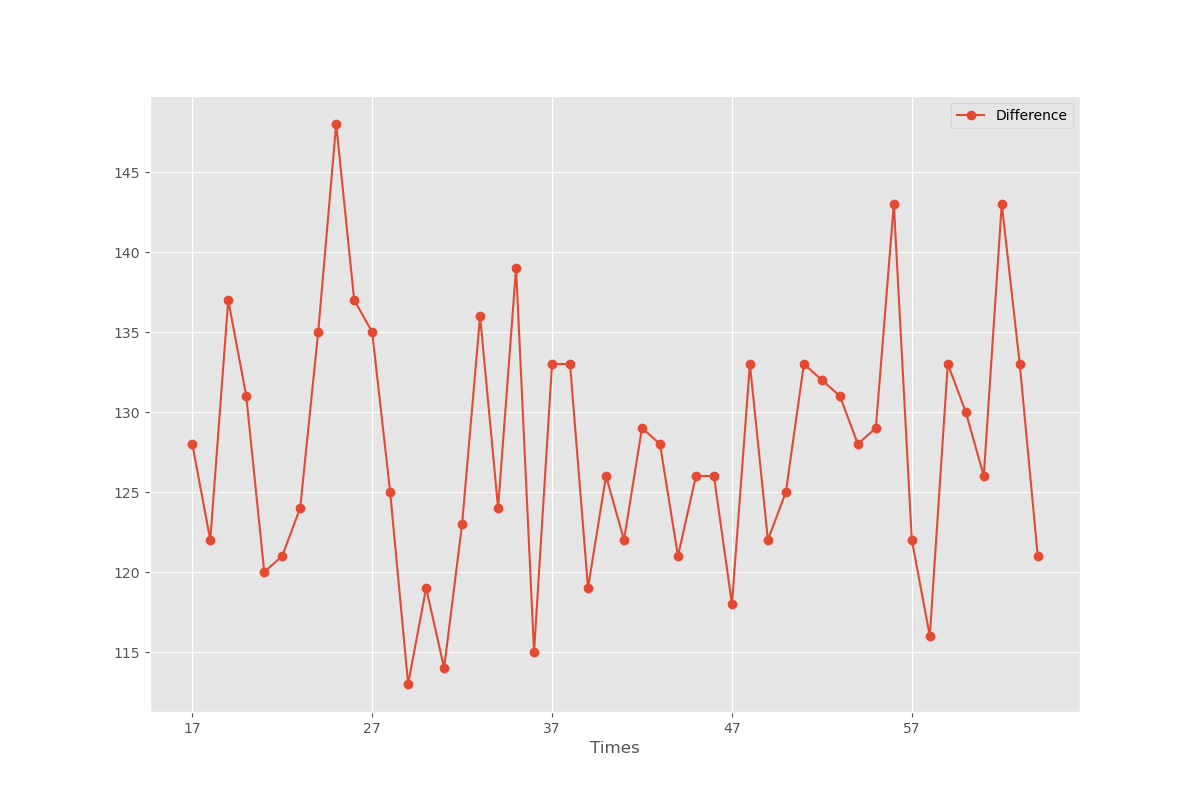
import pandas as pd  
import matplotlib.pyplot as plt  
import matplotlib  
  
matplotlib.style.use(**'ggplot'**)  
  
fn = **r'C:/Users/hui/Desktop/diffs1.csv'**df = pd.read\_csv(fn, sep=**'\s+'**, header=None, names=[**'Times'**, **'Difference'**], parse\_dates=[**'Times'**])  
df.plot(x=**'Times'**, y=**'Difference'**, rot=0, figsize=(12, 8), grid=True, marker=**'o'**)  
plt.savefig(**'1.png'**)  
plt.close()  
  
fn = **r'C:/Users/hui/Desktop/diffs2.csv'**df = pd.read\_csv(fn, sep=**'\s+'**, header=None, names=[**'Times'**, **'Difference'**], parse\_dates=[**'Times'**])  
df.plot(x=**'Times'**, y=**'Difference'**, rot=0, figsize=(12, 8), grid=True, marker=**'o'**)  
plt.savefig(**'2.png'**)  
plt.close()  
  
fn = **r'C:/Users/hui/Desktop/diffs3.csv'**df = pd.read\_csv(fn, sep=**'\s+'**, header=None, names=[**'Times'**, **'Difference'**], parse\_dates=[**'Times'**])  
df.plot(x=**'Times'**, y=**'Difference'**, rot=0, figsize=(12, 8), grid=True, marker=**'o'**)  
plt.savefig(**'3.png'**)

Графики:

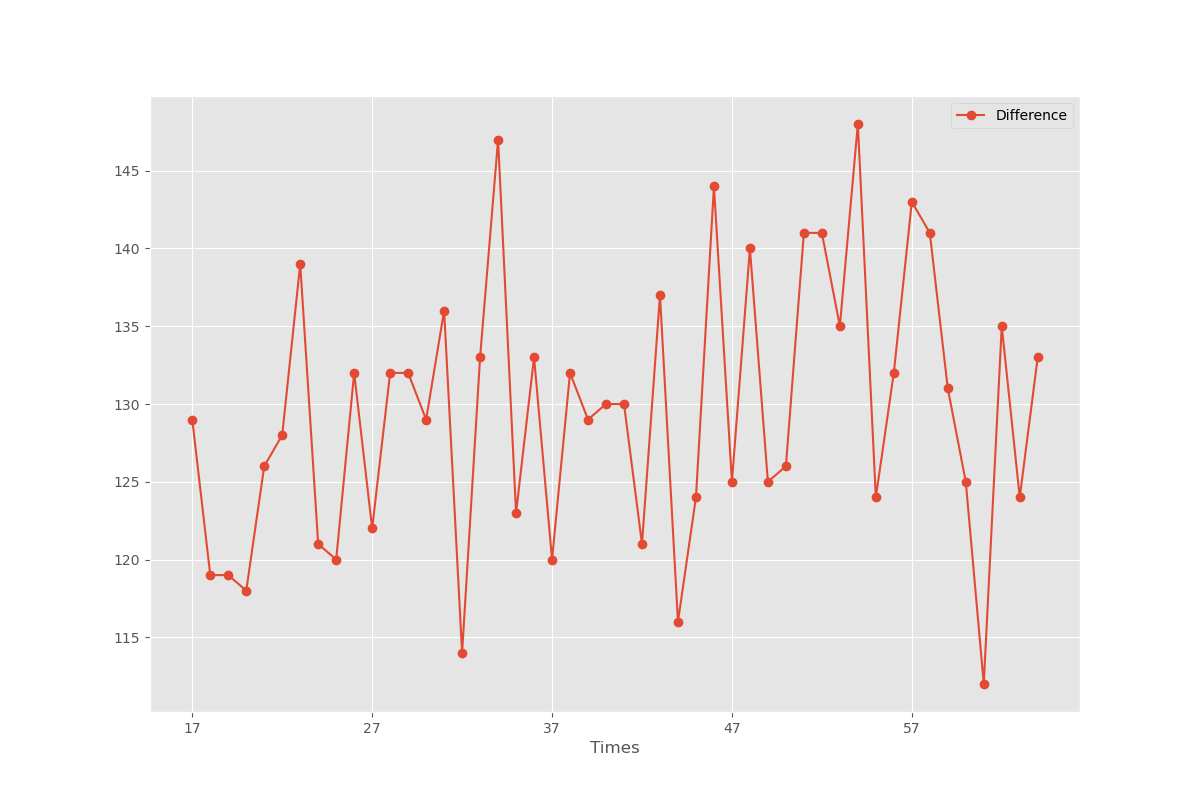
Для строки "ju":



Для строки "heygoodnewseveryone!":



Для строки "hello,darkness,myoldfriend/i'vecometotalkwithyouagain/becauseavisionsoftlycreeping/leftitsseedswhileiwassleeping/andthevisionthatwasplantedinmybrain/stillremains/withinthesoundofsilence/soundofsilence":



Как видно из графиков, при добавление раундов изменяется в среднем половина битов.

Дифференциальный криптоанализ:

До 2008 года не было известно об уязвимости алгоритмов SHA-2. Сейчас же известно, что существуют различные атаки, в том числе дифференциальные, которые на определенном раунде находят коллизии. На данный момент самой «быстрой» коллизией, обнаруженной с помощью дифференциального криптоанализа, является коллизия на 27 раунде в SHA-512, о чем сказано в “Analysis of SHA-512/224 and SHA-512/256” (<https://link.springer.com/chapter/10.1007/978-3-662-48800-3_25>).

Выводы

Во время выполнения лабораторной работы я познакомилась с семейством алгоритмов SHA-2, модифицировала реализацию алгоритма SHA-256 таким образом, чтобы можно было задать количество раундов. Также я познакомилась с криптостойкостью алгоритмов, дифференциальным криптоанализом и различными атаками на хеш-функции.