Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

КАФЕДРА ИНФОРМАТИКИ

Лабораторная работа № 5

Компьютерная реализация хэш-функций на примере MD5

Выполнил: Нетецкая Ю.В.

Проверил: Олисейчик В.В.

Минск 2021

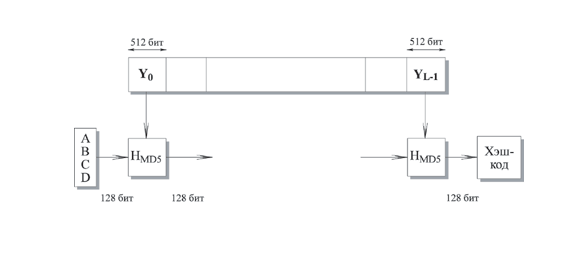
**Постановка задачи**

Реализовать программные средства шифрования и дешифрования текстовых файлов при помощи алгоритмов DES и ГОСТ 28147-89. Проверить корректность выполнения программных средств на примерах.

**Описание использованных алгоритмов**

***Шаг 1: добавление недостающих битов***

Сообщение дополняется таким образом, чтобы его длина стала равна 448 по модулю 512 (длина≡448 mod 512).



***Шаг 2: добавление длины***

64-битное представление длины исходного (до добавления) сообщения в битах присоединяется к результату первого шага. Если первоначальная длина больше, чем 264, то используются только последние 64 бита.

***Шаг 3: инициализация MD-буфера***

Используется 128-битный буфер для хранения промежуточных и окончательных результатов *хэш-функции*. Буфер может быть представлен как четыре 32-битных регистра (A, B, C, D). Эти регистры инициализируются следующими шестнадцатеричными числами:

А = 01234567;

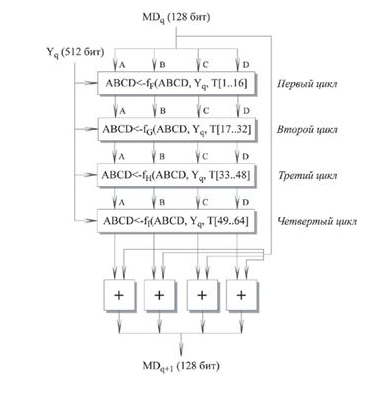
В = 89ABCDEF;

C = FEDCBA98;

D = 76543210

***Шаг 4: обработка последовательности 512-битных (16-словных) блоков***

Основой алгоритма является модуль, состоящий из четырех циклических обработок, обозначенный как HMD5. Четыре цикла имеют похожую структуру, но каждый цикл использует свою элементарную логическую функцию, обозначаемую fF, fG, fH и fI соответственно.



Каждый цикл принимает в качестве входа текущий 512-битный блок Yq, обрабатывающийся в данный момент, и 128-битное значение буфера ABCD, которое является промежуточным значением *дайджеста*, и изменяет содержимое этого буфера. Для получения MDq+1 выход четырех циклов складывается по модулю 232 с MDq. Сложение выполняется независимо для каждого из четырех слов в буфере.

***Шаг 5: выход***

После обработки всех *L* 512-битных блоков выходом *L*-ой стадии является 128-битный *дайджест сообщения*.

**Результат работы программы**



**Код программы**

import binascii  
  
  
SV = [  
 0xd76aa478, 0xe8c7b756, 0x242070db, 0xc1bdceee, 0xf57c0faf,  
 0x4787c62a, 0xa8304613, 0xfd469501, 0x698098d8, 0x8b44f7af,  
 0xffff5bb1, 0x895cd7be, 0x6b901122, 0xfd987193, 0xa679438e,  
 0x49b40821, 0xf61e2562, 0xc040b340, 0x265e5a51, 0xe9b6c7aa,  
 0xd62f105d, 0x2441453, 0xd8a1e681, 0xe7d3fbc8, 0x21e1cde6,  
 0xc33707d6, 0xf4d50d87, 0x455a14ed, 0xa9e3e905, 0xfcefa3f8,  
 0x676f02d9, 0x8d2a4c8a, 0xfffa3942, 0x8771f681, 0x6d9d6122,  
 0xfde5380c, 0xa4beea44, 0x4bdecfa9, 0xf6bb4b60, 0xbebfbc70,  
 0x289b7ec6, 0xeaa127fa, 0xd4ef3085, 0x4881d05, 0xd9d4d039,  
 0xe6db99e5, 0x1fa27cf8, 0xc4ac5665, 0xf4292244, 0x432aff97,  
 0xab9423a7, 0xfc93a039, 0x655b59c3, 0x8f0ccc92, 0xffeff47d,  
 0x85845dd1, 0x6fa87e4f, 0xfe2ce6e0, 0xa3014314, 0x4e0811a1,  
 0xf7537e82, 0xbd3af235, 0x2ad7d2bb, 0xeb86d391]  
  
  
def leftCircularShift(*k*, *bits*):  
 bits = *bits* % 32  
 k = *k* % (2 \*\* 32)  
 upper = (*k* << *bits*) % (2 \*\* 32)  
 result = upper | (*k* >> (32 - *bits*))  
 return result  
  
  
def blockDivide(*block*, *chunks*):  
 result = []  
 size = *len*(*block*) // *chunks* for i in *range*(0, *chunks*):  
 result.append(*int*.from\_bytes(*block*[i \* size:(i + 1) \* size], *byteorder*="little"))  
 return result  
  
  
def F(*X*, *Y*, *Z*):  
 return (*X* & *Y*) | ((~*X*) & *Z*)  
  
  
def G(*X*, *Y*, *Z*):  
 return (*X* & *Z*) | (*Y* & (~*Z*))  
  
  
def H(*X*, *Y*, *Z*):  
 return *X* ^ *Y* ^ *Z*def I(*X*, *Y*, *Z*):  
 return *Y* ^ (*X* | (~*Z*))  
  
  
def FF(*a*, *b*, *c*, *d*, *M*, *s*, *t*):  
 result = *b* + leftCircularShift((*a* + F(*b*, *c*, *d*) + *M* + *t*), *s*)  
 return result  
  
  
def GG(*a*, *b*, *c*, *d*, *M*, *s*, *t*):  
 result = *b* + leftCircularShift((*a* + G(*b*, *c*, *d*) + *M* + *t*), *s*)  
 return result  
  
  
def HH(*a*, *b*, *c*, *d*, *M*, *s*, *t*):  
 result = *b* + leftCircularShift((*a* + H(*b*, *c*, *d*) + *M* + *t*), *s*)  
 return result  
  
  
def II(*a*, *b*, *c*, *d*, *M*, *s*, *t*):  
 result = *b* + leftCircularShift((*a* + I(*b*, *c*, *d*) + *M* + *t*), *s*)  
 return result  
  
  
def fmt8(*num*):  
 bighex = "{0:08x}".format(*num*)  
 binver = binascii.unhexlify(bighex)  
 result = "{0:08x}".format(*int*.from\_bytes(binver,*byteorder*='little'))  
 return result  
  
  
def bitlen(*bitstring*):  
 return *len*(*bitstring*) \* 8  
  
  
def md5sum(*msg*):  
 msgLen = bitlen(*msg*) % (2 \*\* 64)  
 msg = *msg* + b'\x80'  
 zeroPad = (448 - (msgLen + 8) % 512) % 512  
 zeroPad //= 8  
 msg = *msg* + b'\x00'\*zeroPad + msgLen.to\_bytes(8, *byteorder*='little')  
 msgLen = bitlen(*msg*)  
 iterations = msgLen//512  
 A = 0x67452301  
 B = 0xefcdab89  
 C = 0x98badcfe  
 D = 0x10325476  
 for i in *range*(0, iterations):  
 a = A  
 b = B  
 c = C  
 d = D  
 block = *msg*[i \* 64:(i + 1) \* 64]  
 M = blockDivide(block, 16)  
 for i in *range*(0, 16, 4):  
 a = FF(a, b, c, d, M[i], 7, SV[i])  
 d = FF(d, a, b, c, M[i+1], 12, SV[i+1])  
 c = FF(c, d, a, b, M[i+2], 17, SV[i+2])  
 b = FF(b, c, d, a, M[i+3], 22, SV[i+3])  
  
 a = GG(a, b, c, d, M[1], 5, SV[16])  
 d = GG(d, a, b, c, M[6], 9, SV[17])  
 c = GG(c, d, a, b, M[11], 14, SV[18])  
 b = GG(b, c, d, a, M[0], 20, SV[19])  
 a = GG(a, b, c, d, M[5], 5, SV[20])  
 d = GG(d, a, b, c, M[10], 9, SV[21])  
 c = GG(c, d, a, b, M[15], 14, SV[22])  
 b = GG(b, c, d, a, M[4], 20, SV[23])  
 a = GG(a, b, c, d, M[9], 5, SV[24])  
 d = GG(d, a, b, c, M[14], 9, SV[25])  
 c = GG(c, d, a, b, M[3], 14, SV[26])  
 b = GG(b, c, d, a, M[8], 20, SV[27])  
 a = GG(a, b, c, d, M[13], 5, SV[28])  
 d = GG(d, a, b, c, M[2], 9, SV[29])  
 c = GG(c, d, a, b, M[7], 14, SV[30])  
 b = GG(b, c, d, a, M[12], 20, SV[31])  
  
 a = HH(a, b, c, d, M[5], 4, SV[32])  
 d = HH(d, a, b, c, M[8], 11, SV[33])  
 c = HH(c, d, a, b, M[11], 16, SV[34])  
 b = HH(b, c, d, a, M[14], 23, SV[35])  
 a = HH(a, b, c, d, M[1], 4, SV[36])  
 d = HH(d, a, b, c, M[4], 11, SV[37])  
 c = HH(c, d, a, b, M[7], 16, SV[38])  
 b = HH(b, c, d, a, M[10], 23, SV[39])  
 a = HH(a, b, c, d, M[13], 4, SV[40])  
 d = HH(d, a, b, c, M[0], 11, SV[41])  
 c = HH(c, d, a, b, M[3], 16, SV[42])  
 b = HH(b, c, d, a, M[6], 23, SV[43])  
 a = HH(a, b, c, d, M[9], 4, SV[44])  
 d = HH(d, a, b, c, M[12], 11, SV[45])  
 c = HH(c, d, a, b, M[15], 16, SV[46])  
 b = HH(b, c, d, a, M[2], 23, SV[47])  
  
 a = II(a, b, c, d, M[0], 6, SV[48])  
 d = II(d, a, b, c, M[7], 10, SV[49])  
 c = II(c, d, a, b, M[14], 15, SV[50])  
 b = II(b, c, d, a, M[5], 21, SV[51])  
 a = II(a, b, c, d, M[12], 6, SV[52])  
 d = II(d, a, b, c, M[3], 10, SV[53])  
 c = II(c, d, a, b, M[10], 15, SV[54])  
 b = II(b, c, d, a, M[1], 21, SV[55])  
 a = II(a, b, c, d, M[8], 6, SV[56])  
 d = II(d, a, b, c, M[15], 10, SV[57])  
 c = II(c, d, a, b, M[6], 15, SV[58])  
 b = II(b, c, d, a, M[13], 21, SV[59])  
 a = II(a, b, c, d, M[4], 6, SV[60])  
 d = II(d, a, b, c, M[11], 10, SV[61])  
 c = II(c, d, a, b, M[2], 15, SV[62])  
 b = II(b, c, d, a, M[9], 21, SV[63])  
  
 A = (A + a) % (2 \*\* 32)  
 B = (B + b) % (2 \*\* 32)  
 C = (C + c) % (2 \*\* 32)  
 D = (D + d) % (2 \*\* 32)  
  
 result = fmt8(A)+fmt8(B)+fmt8(C)+fmt8(D)  
 return result  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 data = b'Lab 5 algo'  
 *print*(md5sum(data))  
 *print*(*str*(md5sum(data)) == 'fc3e0bf1183535262df4ae24d3000cf2')

**Вывод**

В ходе данной лабораторной работы были изучены теоретические сведения. Создана программа, читающая данные из файла и формирующая в качестве выхода *дайджест сообщения* длиной 128 бит по алгоритму MD5.