**Министерство науки и высшего образования Российской Федерации**

Федеральное государственное автономное образовательное учреждение

высшего профессионального образования

**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ**

**ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

Инженерная школа информационных технологий и робототехники

Направление 09.03.02 «Информационные системы и технологии»

Отделение информационных технологий

Отчёт по лабораторной работе № 5

по дисциплине **«Информационная безопасность и защита информации»**

**Алгоритм хэширования MD5.**

Выполнил:

Студент группы 8И6Б Карташев И.А. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Отчет принял:

Ассистент (ОИТ, ИШИТР) Григорьев Д. С. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Томск 2021

# 1 Цель работы

Целью лабораторной работы является создание программной реализации шифрования на основе алгоритма MD5.

# 2 Задание

# 1. Разработка программной реализации соответствующего предложенной теме шифра, цифровой подписи, хэш-функции или комплекса криптоалгоритмов;

# 2. В подготовке отчета по проделанной работе, по форме приведенной выше.

# 3 Информация об алгоритме

# MD5 (англ. Message Digest 5) — 128-битный алгоритм хеширования, разработанный профессором Рональдом Л. Ривестом из Массачусетского технологического института в 1991 году. Предназначен для создания «отпечатков» или дайджестов сообщения произвольной длины и последующей проверки их подлинности. Широко применялся для проверки целостности информации и хранения хешей паролей.

# 4 Общее описание алгоритма

Алгоритм состоит из 5 шагов.

На вход алгоритма поступает входной поток данных, хеш которого необходимо найти. Длина сообщения измеряется в битах и может быть любой (в том числе нулевой). Запишем длину сообщения в L. Это число целое и неотрицательное. Кратность каким-либо числам необязательна. После поступления данных идёт процесс подготовки потока к вычислениям.

**Шаг 1. Выравнивание потока**

Сначала к концу потока дописывают единичный бит.

Затем добавляют некоторое число нулевых бит такое, чтобы новая длина потока стала сравнима с 448 по модулю 512. Выравнивание происходит в любом случае, даже если длина исходного потока уже сравнима с 448.

**Шаг 2. Добавление длины данных**

В конец сообщения дописывают 64-битное представление (в Little Endian) длины данных (количество бит в сообщении) до выравнивания. Сначала записывают младшие 4 байта, затем старшие. Если длина превосходит (2^64)-1, то дописывают только младшие биты.

После этого длина потока станет кратной 512. Вычисления будут основываться на представлении этого потока данных в виде массива слов по 512 бит.

**Шаг 3. Инициализация буфера**

Для вычислений инициализируются четыре переменные размером по 32 бита, начальные значения которых задаются шестнадцатеричными числами (порядок байтов little-endian):

А = 01 23 45 67; // 67452301h

В = 89 AB CD EF; // EFCDAB89h

С = FE DC BA 98; // 98BADCFEh

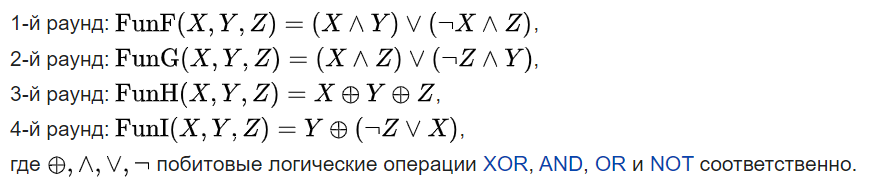
D = 76 54 32 10. // 10325476h

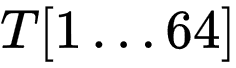
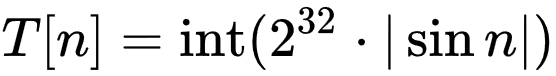
В этих переменных будут храниться результаты промежуточных вычислений. Начальное состояние ABCD называется инициализирующим вектором.

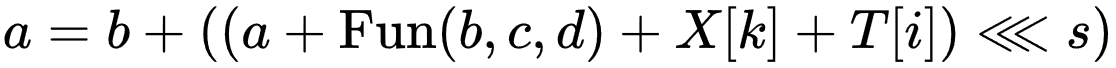
**Шаг 4. Прохождение 4-х раундов**

Определим функции и константы, которые понадобятся нам для вычислений.

* Для каждого раунда потребуется своя функция. Введём функции от трёх параметров — слов, результатом также будет слово:



* Определим таблицу констант  — 64-элементная таблица данных, построенная следующим образом: 
* Каждый 512-битный блок проходит 4 этапа вычислений по 16 раундов.

Для этого блок представляется в виде массива *X* из 16 слов по 32 бита. Все раунды однотипны и имеют вид: [abcd k s i], определяемый как {\displaystyle a=b+((a+\operatorname {Fun} (b,c,d)+X[k]+T[i])\lll s)} , где *k* — номер 32-битного слова из текущего 512-битного блока сообщения, и {\displaystyle \ldots \lll s}<<< — циклический сдвиг влево на *s* бит полученного 32-битного аргумента. Число *s* задается отдельно для каждого раунда.

Заносим в блок данных элемент *n* из массива 512-битных блоков. Сохраняются значения A, B, C и D, оставшиеся после операций над предыдущими блоками (или их начальные значения, если блок первый).

AA = A

BB = B

CC = C

DD = D

**Этап 1**

/\* [abcd k s i] a = b + ((a + F(b,c,d) + X[k] + T[i]) <<< s). \*/

[ABCD 0 7 1][DABC 1 12 2][CDAB 2 17 3][BCDA 3 22 4]

[ABCD 4 7 5][DABC 5 12 6][CDAB 6 17 7][BCDA 7 22 8]

[ABCD 8 7 9][DABC 9 12 10][CDAB 10 17 11][BCDA 11 22 12]

[ABCD 12 7 13][DABC 13 12 14][CDAB 14 17 15][BCDA 15 22 16]

**Этап 2**

/\* [abcd k s i] a = b + ((a + G(b,c,d) + X[k] + T[i]) <<< s). \*/

[ABCD 1 5 17][DABC 6 9 18][CDAB 11 14 19][BCDA 0 20 20]

[ABCD 5 5 21][DABC 10 9 22][CDAB 15 14 23][BCDA 4 20 24]

[ABCD 9 5 25][DABC 14 9 26][CDAB 3 14 27][BCDA 8 20 28]

[ABCD 13 5 29][DABC 2 9 30][CDAB 7 14 31][BCDA 12 20 32]

**Этап 3**

/\* [abcd k s i] a = b + ((a + H(b,c,d) + X[k] + T[i]) <<< s). \*/

[ABCD 5 4 33][DABC 8 11 34][CDAB 11 16 35][BCDA 14 23 36]

[ABCD 1 4 37][DABC 4 11 38][CDAB 7 16 39][BCDA 10 23 40]

[ABCD 13 4 41][DABC 0 11 42][CDAB 3 16 43][BCDA 6 23 44]

[ABCD 9 4 45][DABC 12 11 46][CDAB 15 16 47][BCDA 2 23 48]

**Этап 4**

/\* [abcd k s i] a = b + ((a + I(b,c,d) + X[k] + T[i]) <<< s). \*/

[ABCD 0 6 49][DABC 7 10 50][CDAB 14 15 51][BCDA 5 21 52]

[ABCD 12 6 53][DABC 3 10 54][CDAB 10 15 55][BCDA 1 21 56]

[ABCD 8 6 57][DABC 15 10 58][CDAB 6 15 59][BCDA 13 21 60]

[ABCD 4 6 61][DABC 11 10 62][CDAB 2 15 63][BCDA 9 21 64]

Суммируем с результатом предыдущего цикла:

A = AA + A

B = BB + B

C = CC + C

D = DD + D

5 Листинг программы

Main.go:

package main

import (

    "bufio"

    "bytes"

    "encoding/binary"

    "fmt"

    "io/ioutil"

    "log"

    "math"

    "os"

    "runtime"

)

var (

    A, B, C, D uint32

    T          [64]uint32

)

func main() {

    var md5 string

    in := bufio.NewScanner(os.Stdin)

    for {

        if len(os.Args) != 2 {

            fmt.Print("String to hash: ")

            in.Scan()

            md5 = calcMD5([]byte(in.Text()))

        } else {

            file, err := ioutil.ReadFile(os.Args[1])

            if err != nil {

                log.Fatalln(err)

            }

            md5 = calcMD5(file)

            fmt.Printf("File: %s\n", os.Args[1])

        }

        fmt.Printf("md5:  %s\n", md5)

        in.Scan()

        clearScr(runtime.GOOS)

    }

}

func calcMD5(buf []byte) string {

    dataLen := uint64(len(buf) \* 8)

    appendPaddingBytes(&buf)

    appendLength(&buf, dataLen)

    initMDbuf()

    processMsgIn16WordBlocks(byteToUintArr(buf))

    return rawMD5ToHEX(A) + rawMD5ToHEX(B) + rawMD5ToHEX(C) + rawMD5ToHEX(D)

}

func appendPaddingBytes(buf \*[]byte) {

    fmt.Print(128)

    \*buf = append(\*buf, 128)

    for len(\*buf)%64 != 56 {

        \*buf = append(\*buf, 0)

    }

}

func appendLength(buf \*[]byte, length uint64) {

    buffer := bytes.NewBuffer(nil)

    if err := binary.Write(buffer, binary.LittleEndian, length); err != nil {

        log.Fatalln(err)

    }

    for \_, b := range buffer.Bytes() {

        \*buf = append(\*buf, b)

    }

}

func initMDbuf() {

    A = 0x67452301

    B = 0xEFCDAB89

    C = 0x98BADCFE

    D = 0x10325476

    for i := 0; i < 64; i++ {

        T[i] = uint32(math.Pow(2, 32) \* math.Abs(math.Sin(float64(i+1))))

    }

}

func processMsgIn16WordBlocks(buf []uint32) {

    for n := 0; n < len(buf); n += 16 {

        AA, BB, CC, DD := A, B, C, D

        // Round 1

        A = B + rotateLeft((A+F(B, C, D)+buf[n+0]+T[0]), 7)

        D = A + rotateLeft((D+F(A, B, C)+buf[n+1]+T[1]), 12)

        C = D + rotateLeft((C+F(D, A, B)+buf[n+2]+T[2]), 17)

        B = C + rotateLeft((B+F(C, D, A)+buf[n+3]+T[3]), 22)

        A = B + rotateLeft((A+F(B, C, D)+buf[n+4]+T[4]), 7)

        D = A + rotateLeft((D+F(A, B, C)+buf[n+5]+T[5]), 12)

        C = D + rotateLeft((C+F(D, A, B)+buf[n+6]+T[6]), 17)

        B = C + rotateLeft((B+F(C, D, A)+buf[n+7]+T[7]), 22)

        A = B + rotateLeft((A+F(B, C, D)+buf[n+8]+T[8]), 7)

        D = A + rotateLeft((D+F(A, B, C)+buf[n+9]+T[9]), 12)

        C = D + rotateLeft((C+F(D, A, B)+buf[n+10]+T[10]), 17)

        B = C + rotateLeft((B+F(C, D, A)+buf[n+11]+T[11]), 22)

        A = B + rotateLeft((A+F(B, C, D)+buf[n+12]+T[12]), 7)

        D = A + rotateLeft((D+F(A, B, C)+buf[n+13]+T[13]), 12)

        C = D + rotateLeft((C+F(D, A, B)+buf[n+14]+T[14]), 17)

        B = C + rotateLeft((B+F(C, D, A)+buf[n+15]+T[15]), 22)

        // Round 2

        A = B + rotateLeft((A+G(B, C, D)+buf[n+1]+T[16]), 5)

        D = A + rotateLeft((D+G(A, B, C)+buf[n+6]+T[17]), 9)

        C = D + rotateLeft((C+G(D, A, B)+buf[n+11]+T[18]), 14)

        B = C + rotateLeft((B+G(C, D, A)+buf[n+0]+T[19]), 20)

        A = B + rotateLeft((A+G(B, C, D)+buf[n+5]+T[20]), 5)

        D = A + rotateLeft((D+G(A, B, C)+buf[n+10]+T[21]), 9)

        C = D + rotateLeft((C+G(D, A, B)+buf[n+15]+T[22]), 14)

        B = C + rotateLeft((B+G(C, D, A)+buf[n+4]+T[23]), 20)

        A = B + rotateLeft((A+G(B, C, D)+buf[n+9]+T[24]), 5)

        D = A + rotateLeft((D+G(A, B, C)+buf[n+14]+T[25]), 9)

        C = D + rotateLeft((C+G(D, A, B)+buf[n+3]+T[26]), 14)

        B = C + rotateLeft((B+G(C, D, A)+buf[n+8]+T[27]), 20)

        A = B + rotateLeft((A+G(B, C, D)+buf[n+13]+T[28]), 5)

        D = A + rotateLeft((D+G(A, B, C)+buf[n+2]+T[29]), 9)

        C = D + rotateLeft((C+G(D, A, B)+buf[n+7]+T[30]), 14)

        B = C + rotateLeft((B+G(C, D, A)+buf[n+12]+T[31]), 20)

        // Round 3

        A = B + rotateLeft((A+H(B, C, D)+buf[n+5]+T[32]), 4)

        D = A + rotateLeft((D+H(A, B, C)+buf[n+8]+T[33]), 11)

        C = D + rotateLeft((C+H(D, A, B)+buf[n+11]+T[34]), 16)

        B = C + rotateLeft((B+H(C, D, A)+buf[n+14]+T[35]), 23)

        A = B + rotateLeft((A+H(B, C, D)+buf[n+1]+T[36]), 4)

        D = A + rotateLeft((D+H(A, B, C)+buf[n+4]+T[37]), 11)

        C = D + rotateLeft((C+H(D, A, B)+buf[n+7]+T[38]), 16)

        B = C + rotateLeft((B+H(C, D, A)+buf[n+10]+T[39]), 23)

        A = B + rotateLeft((A+H(B, C, D)+buf[n+13]+T[40]), 4)

        D = A + rotateLeft((D+H(A, B, C)+buf[n+0]+T[41]), 11)

        C = D + rotateLeft((C+H(D, A, B)+buf[n+3]+T[42]), 16)

        B = C + rotateLeft((B+H(C, D, A)+buf[n+6]+T[43]), 23)

        A = B + rotateLeft((A+H(B, C, D)+buf[n+9]+T[44]), 4)

        D = A + rotateLeft((D+H(A, B, C)+buf[n+12]+T[45]), 11)

        C = D + rotateLeft((C+H(D, A, B)+buf[n+15]+T[46]), 16)

        B = C + rotateLeft((B+H(C, D, A)+buf[n+2]+T[47]), 23)

        // Round 4

        A = B + rotateLeft((A+I(B, C, D)+buf[n+0]+T[48]), 6)

        D = A + rotateLeft((D+I(A, B, C)+buf[n+7]+T[49]), 10)

        C = D + rotateLeft((C+I(D, A, B)+buf[n+14]+T[50]), 15)

        B = C + rotateLeft((B+I(C, D, A)+buf[n+5]+T[51]), 21)

        A = B + rotateLeft((A+I(B, C, D)+buf[n+12]+T[52]), 6)

        D = A + rotateLeft((D+I(A, B, C)+buf[n+3]+T[53]), 10)

        C = D + rotateLeft((C+I(D, A, B)+buf[n+10]+T[54]), 15)

        B = C + rotateLeft((B+I(C, D, A)+buf[n+1]+T[55]), 21)

        A = B + rotateLeft((A+I(B, C, D)+buf[n+8]+T[56]), 6)

        D = A + rotateLeft((D+I(A, B, C)+buf[n+15]+T[57]), 10)

        C = D + rotateLeft((C+I(D, A, B)+buf[n+6]+T[58]), 15)

        B = C + rotateLeft((B+I(C, D, A)+buf[n+13]+T[59]), 21)

        A = B + rotateLeft((A+I(B, C, D)+buf[n+4]+T[60]), 6)

        D = A + rotateLeft((D+I(A, B, C)+buf[n+11]+T[61]), 10)

        C = D + rotateLeft((C+I(D, A, B)+buf[n+2]+T[62]), 15)

        B = C + rotateLeft((B+I(C, D, A)+buf[n+9]+T[63]), 21)

        A += AA

        B += BB

        C += CC

        D += DD

    }

}

Funcs.go:

package main

import (

    "fmt"

    "os"

    "os/exec"

)

func F(X, Y, Z uint32) uint32 {

    return (X & Y) | (^X & Z)

}

func G(X, Y, Z uint32) uint32 {

    return (X & Z) | (Y & ^Z)

}

func H(X, Y, Z uint32) uint32 {

    return X ^ Y ^ Z

}

func I(X, Y, Z uint32) uint32 {

    return Y ^ (X | ^Z)

}

func rotateLeft(x, n uint32) uint32 {

    return (x << n) | (x >> (32 - n))

}

func byteToUintArr(buf []byte) []uint32 {

    words := make([]uint32, len(buf)/4)

    for i := 0; i < len(buf); i += 4 {

        words[i/4] += uint32(buf[i+0]) << 0

        words[i/4] += uint32(buf[i+1]) << 8

        words[i/4] += uint32(buf[i+2]) << 16

        words[i/4] += uint32(buf[i+3]) << 24

    }

    return words

}

func rawMD5ToHEX(value uint32) string {

    res := ""

    for i := 0; i < 4; i++ {

        res += fmt.Sprintf("%02X", value%256)

        value /= 256

    }

    return res

}

func clearScr(osName string) {

    var cmd \*exec.Cmd

    switch osName {

    case "linux":

        cmd = exec.Command("clear")

    case "windows":

        cmd = exec.Command("cmd", "/c", "cls")

    default:

        fmt.Println()

        return

    }

    cmd.Stdout = os.Stdout

    cmd.Run()

}

# 6 Результаты работы программы

Результат программы приведен на рисунке 2.

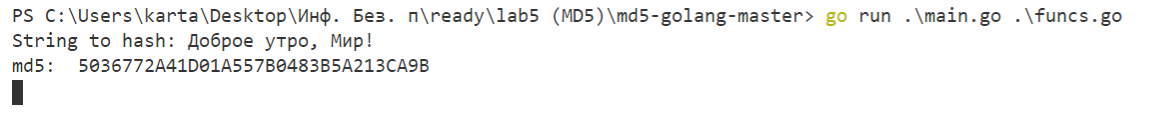


Рисунок 1 – Результат работы программы

# 6 Вывод

В ходе выполнения лабораторной работы были изучены теоретические сведения об алгоритме хэширования MD5, была реализована программа, шифрующая текстовые сообщения с помощью этого алгоритма.

В настоящее время алгоритм не рекомендуют к использованию для некоторых целей (например, хэширования паролей). Но для проверки файлов на целостность он подходит как нельзя лучше.

Ссылка на github: https://github.com/Vano-100/inf-security