Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Пермский национальный исследовательский политехнический университет»
Электротехнический факультет
Кафедра «Информационные технологии и автоматизированные системы»

Дисциплина: «Защита информации»

Профиль: «Автоматизированные системы обработки информации и управления»

Семестр 5

ОТЧЕТ

по лабораторной работе №7

Тема: «Алгоритмы хеширования паролей»

Выполнил: студент группы РИС-19-1б
Миннахметов Э.Ю. _____
Проверил: доцент кафедры ИТАС
Шереметьев В. Г. _____
Дата _____

Пермь, 2021

ЦЕЛЬ РАБОТЫ

Получить практические навыки по созданию алгоритмов хеширования паролей.

ЗАДАНИЕ

Написать программу, реализующую методику хеширования паролей, используя в качестве блочного шифра для реализации алгоритма написанный ранее в лабораторной работе №4 блочный шифр. Максимальная длина пароля выбирается разработчиком алгоритма на его усмотрение, но не должна быть меньше 4 символов.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Хеширование паролей

От методов, повышающих криптостойкость системы в целом, перейдем к блоку хеширования паролей – методу, позволяющему пользователям запоминать не 128 байт, то есть 256 шестнадцатиричных цифр ключа, а некоторое осмысленное выражение, слово или последовательность символов, называющуюся паролем. Действительно, при разработке любого криптоалгоритма следует учитывать, что в половине случаев конечным пользователем системы является человек, а не автоматическая система. Это ставит вопрос о том, удобно, и вообще реально ли человеку запомнить 128-битный ключ (32 шестнадцатиричные цифры). На самом деле предел запоминаемости лежит на границе 8-12 подобных символов, а, следовательно, если мы будем заставлять пользователя оперировать именно ключом, тем самым мы практически вынудим его к записи ключа на каком-либо листке бумаги или электронном носителе, например, в текстовом файле. Это, естественно, резко снижает защищенность системы.

Для решения этой проблемы были разработаны методы, преобразующие произносимую, осмысленную строку произвольной длины – пароль, в указанный ключ заранее заданной длины. В подавляющем большинстве случаев для этой операции используются так называемые хеш-функции (от англ. hashing – мелкая нарезка и перемешивание). Хеш-функцией называется такое математическое или алгоритмическое преобразование заданного блока данных, которое обладает следующими свойствами:

1. хеш-функция имеет бесконечную область определения,
2. хеш-функция имеет конечную область значений,
3. она необратима,
4. изменение входного потока информации на один бит меняет около половины всех бит выходного потока, то есть результата хеш-функции.

Эти свойства позволяют подавать на вход хеш-функции пароли, то есть текстовые строки произвольной длины на любом национальном языке и, ограничив область значений функции диапазоном $0..2^N-1$, где N – длина ключа в битах, получать на выходе достаточно равномерно распределенные по области значения блоки информации – ключи.

Нетрудно заметить, что требования, подобные 3 и 4 пунктам требований к хеш-функции, выполняют блочные шифры. Это указывает на один из возможных путей реализации стойких хеш-функций – проведение блочных криптопреобразований над материалом строки-пароля. Этот метод и используется в различных вариациях практически во всех современных криптосистемах. Материал строки-пароля многократно последовательно используется в качестве ключа для шифрования некоторого заранее известного блока данных – на выходе получается зашифрованный блок информации, однозначно зависящий только от пароля и при этом имеющий достаточно хорошие статистические характеристики. Такой блок или несколько таких блоков и используются в качестве ключа для дальнейших криптопреобразований.

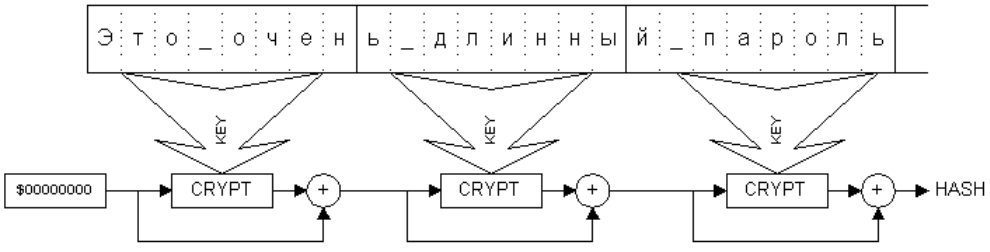
Характер применения блочного шифра для хеширования определяется отношением размера блока используемого криптоалгоритма и разрядности требуемого хеш-результата.

Если указанные выше величины совпадают, то используется схема одноцепочечного блочного шифрования. Первоначальное значение хеш-результата H_0 устанавливается равным 0, вся строка-пароль разбивается на блоки байт, равные по длине ключу используемого для хеширования блочного шифра, затем производятся преобразования по реккурентной формуле:

$H_j=H_{j-1}$ XOR EnCrypt(H_{j-1},PSW_j),

где EnCrypt(X,Key) – используемый блочный шифр.

Последнее значение H_k используется в качестве искомого результата.



В том случае, когда длина ключа ровно в два раза превосходит длину блока, а подобная зависимость довольно часто встречается в блочных шифрах, используется схема, напоминающая сеть Фейштеля. Характерным недостатком и приведенной выше формулы, и хеш-функции, основанной на сети Фейштеля, является большая ресурсоемкость в отношении пароля. Для проведения только одного преобразования,

например, блочным шифром с ключом длиной 128 бит используется 16 байт строки-пароля, а сама длина пароля редко превышает 32 символа. Следовательно, при вычислении хеш-функции над паролем будут произведено максимум 2 «полноценных» криптопреобразования.

Решение этой проблемы можно достичь двумя путями : 1) предварительно «размножить» строку-пароль, например, записав ее многократно последовательно до достижения длины, скажем, в 256 символов; 2) модифицировать схему использования криптоалгоритма так, чтобы материал строки-пароля "медленнее" тратился при вычислении ключа.

По второму пути пошли исследователи Девис и Майер, предложившие алгоритм также на основе блочного шифра, но использующий материал строки-пароля многократно и небольшими порциями. В нем просматриваются элементы обеих приведенных выше схем, но криптостойкость этого алгоритма подтверждена многочисленными реализациями в различных криптосистемах. Алгоритм получил название «Tandem DM»:

$G_0=0$; $H_0=0$;

FOR J = 1 TO N DO

BEGIN

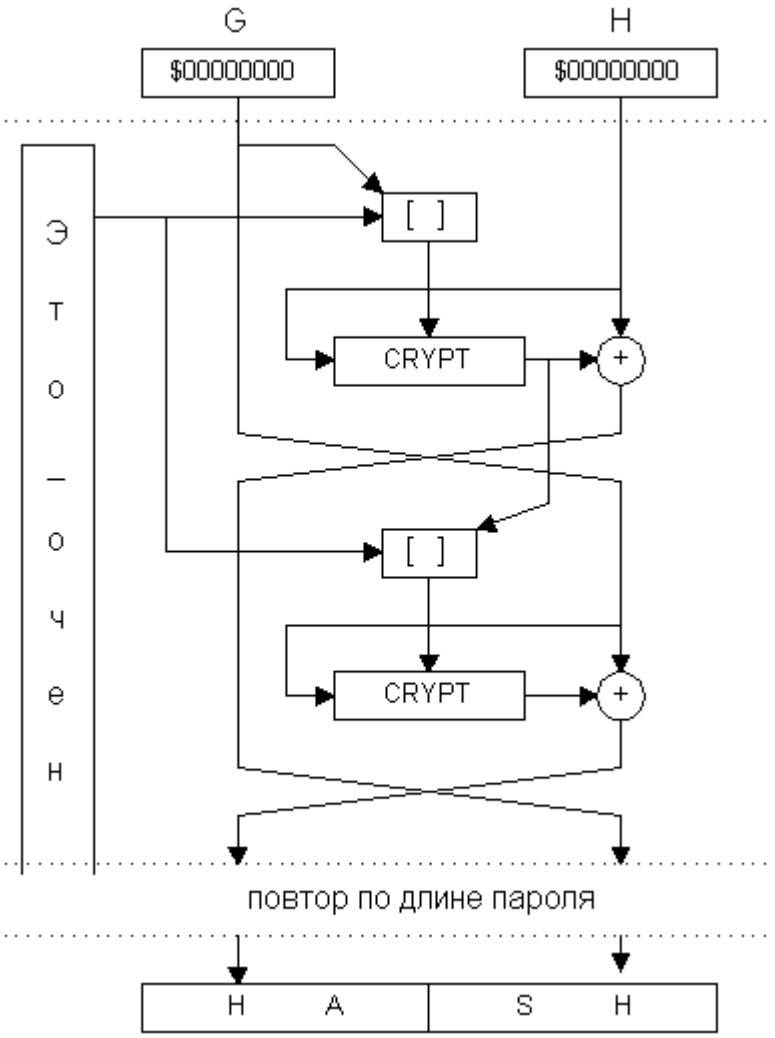
TMP=EnCrypt(H,[G,PSW_j]); H'=H XOR TMP;

TMP=EnCrypt(G,[PSW_j,TMP]); G'=G XOR TMP;

END;

Key=[G_k,H_k]

Квадратными скобками (X16=[A8,B8]) здесь обозначено простое объединение (склеивание) двух блоков информации равной величины в один – удвоенной разрядности. А в качестве процедуры EnCrypt(X,Key) опять может быть выбран любой стойкий блочный шифр. Как видно из формул, данный алгоритм ориентирован на то, что длина ключа двукратно превышает размер блока криптоалгоритма. А характерной особенностью схемы является тот факт, что строка пароля считывается блоками по половине длины ключа, и каждый блок используется в создании хеш-результата дважды. Таким образом, при длине пароля в 20 символов и необходимости создания 128 битного ключа внутренний цикл хеш-функции повторится 3 раза.


G
H
$00000000
$00000000
Это_очен
[ ]
CRYPT
+
[ ]
CRYPT
+
повтор по длине пароля
H A
S H

ХОД РАБОТЫ

На рисунке 1 представлена главная форма программы.

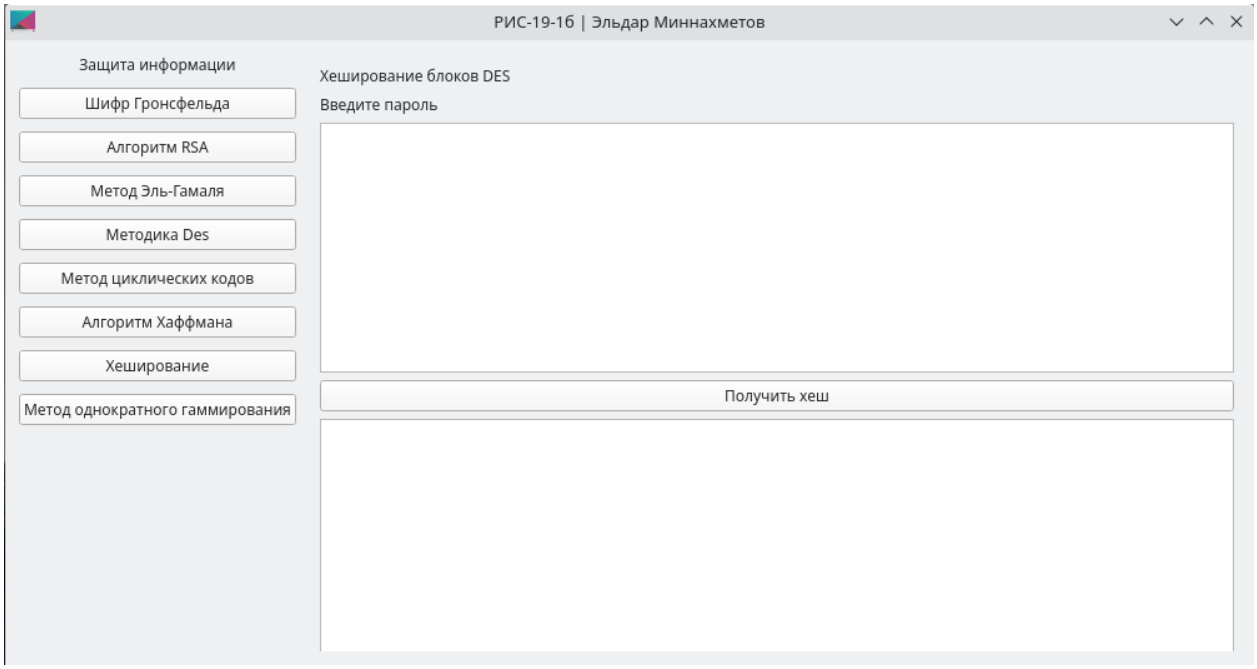


Рисунок 1 – Главная форма программы.

Пример работы программы представлен на рисунке 2.

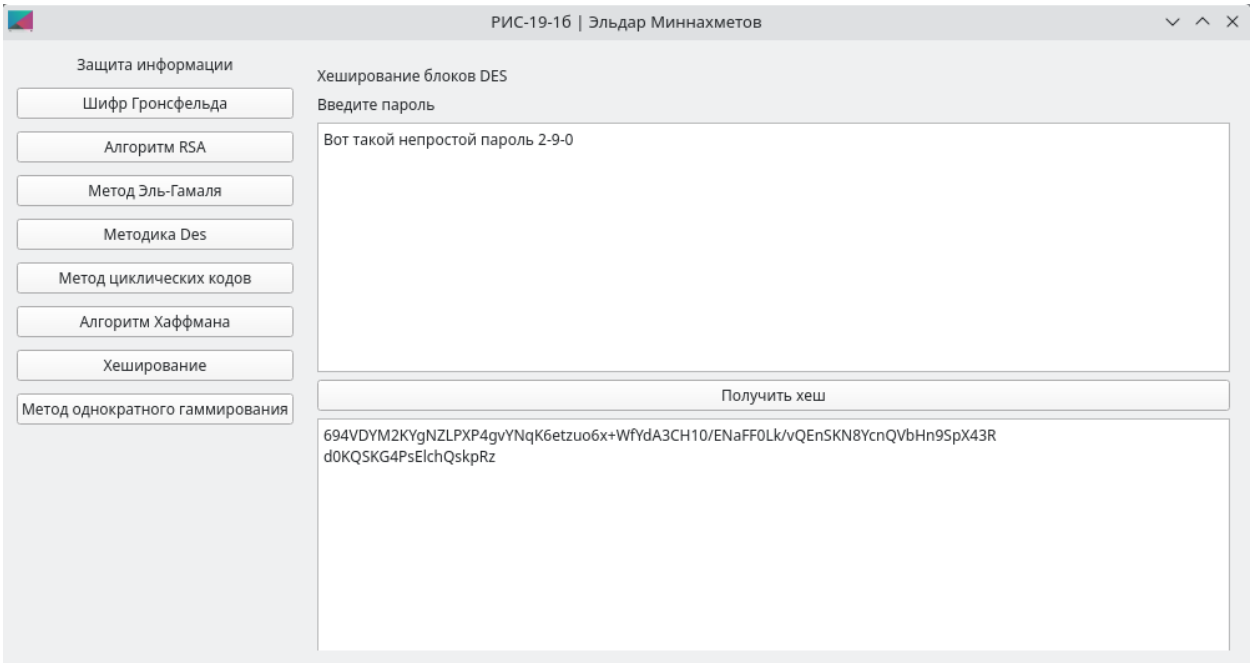


Рисунок 2 – Пример работы программы.

ПРИЛОЖЕНИЕ А

Листинг файла BackApplication.kt

```
package org.eldarian.backend

import org.springframework.boot.autoconfigure.SpringBootApplication
import org.springframework.boot.runApplication
import org.springframework.web.bind.annotation.*
import java.math.BigInteger
import java.util.*

@SpringBootApplication
class BackApplication

fun main(args: Array<String>) {
    runApplication<BackApplication>(*args)
}

@RestController
class LController {
    @PostMapping("/hash")
    fun hash(@RequestBody body: HashRequest): HashResponse {
        return HashResponse(body.text)
    }
}
```

ПРИЛОЖЕНИЕ Б

Листинг класса common.kt

```
package org.eldarian.backend

import sun.misc.BASE64Encoder
import java.util.*
import javax.crypto.Cipher
import javax.crypto.KeyGenerator

data class HashRequest (val text: String)

data class HashResponse (var hash: String) {
  init {
    val cipher = Cipher.getInstance("DES")
    val key = KeyGenerator.getInstance("DES").generateKey()
    cipher.init(Cipher.ENCRYPT_MODE, key)
    val utf8 = hash.toByteArray(Charsets.UTF_16)
    val enc = cipher.doFinal(utf8)
    hash = BASE64Encoder().encode(enc)
  }
}
```

ПРИЛОЖЕНИЕ В

Листинг файла HashTask.h

```
#pragma once

#include <QWidget>

#include "Task.h"
#include "Loader.h"

class QLabel;
class QTextEdit;
class QPushButton;
class QVBoxLayout;

class HashTask: public QWidget, public Task {
Q_OBJECT

private:
   QVBoxLayout *lytMain;
   QLabel *lblName;
   QLabel *lblEnter;
   QTextEdit *teIn;
   QTextEdit *teOut;
   QPushButton *btnHash;

public:
   HashTask(): Task("Хеширование") {}

   void initWidget(QWidget *wgt) override;

   void run() const override { }
   void setHash(QString text);

private slots:
   void hash();

};

class HashLoader : public PostLoadTask {
private:
   HashTask* task;
   QString text;

public:
   explicit HashLoader(HashTask* task, QString text) : task(task), text(text) {}
   QString query() override;
   QJsonDocument request() override;
   void done(QJsonObject& json) override;

};
```

ПРИЛОЖЕНИЕ Г

Листинг файла HashTask.cpp

```
#include <QJsonObject>
#include <QJsonDocument>
#include <QByteArray>
#include <QVBoxLayout>
#include <QTextEdit>
#include <QPushButton>
#include <QLabel>

#include "HashTask.h"

void HashTask::initWidget(QWidget *wgt) {
   lytMain = new QVBoxLayout;
   lblName = new QLabel("Хеширование блоков DES", wgt);
   lblEnter = new QLabel("Введите пароль", wgt);
   teIn = new QTextEdit;
   teOut = new QTextEdit;
   btnHash = new QPushButton("Получить хеш");

   wgt->setLayout(lytMain);
   lytMain->addWidget(lblName);
   lytMain->addWidget(lblEnter);
   lytMain->addWidget(teIn);
   lytMain->addWidget(btnHash);
   lytMain->addWidget(teOut);

   connect(btnHash, SIGNAL(released()), SLOT(hash()));
}

void HashTask::setHash(QString text) {
   teOut->setText(text);
}

void HashTask::hash() {
   (new HashLoader(this, teIn->toPlainText()))->run();
}

QString HashLoader::query() {
   return "hash";
}

QJsonDocument HashLoader::request() {
   QJsonObject obj;
   obj["text"] = text;
   QJsonDocument doc(obj);
   return doc;
}

void HashLoader::done(QJsonObject& json) {
   task->setHash(json["hash"].toString());
}
```