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»

Направление 09.03.01 – «Информатика и вычислительная техника»

Дисциплина: «Защита информации»

Профиль: «Автоматизированные системы обработки информации и управления»

Семестр 7

ОТЧЕТ

по лабораторной работе №7

Тема: «Алгоритмы хеширования паролей»

Выполнил: студент группы АСУ-14-1б

Калмыков В. А. \_\_\_\_\_\_\_\_\_\_\_

Проверил: доцент кафедры ИТАС

Шереметьев В. Г. \_\_\_\_\_\_\_\_\_\_\_

Дата \_\_\_\_\_\_

Пермь, 2017

**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 напи-санный 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..2N-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.

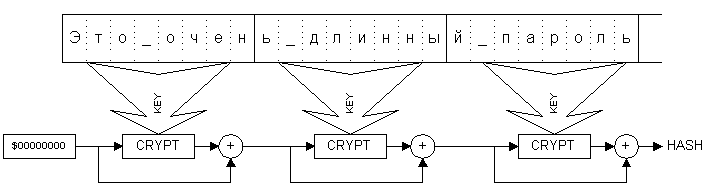
Характер применения блочного шифра для хеширования определяется отношением размера блока используемого криптоалгоритма и разрядности требуемого хеш-результата.

Если указанные выше величины совпадают, то используется схема одноцепочечного блочного шифрования. Первоначальное значение хеш-результата H0 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:

Hj=Hj-1 XOR EnCrypt(Hj-1,PSWj),

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

Последнее значение Hk 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»:

G0=0; H0=0 ;

FOR J = 1 TO N DO

BEGIN

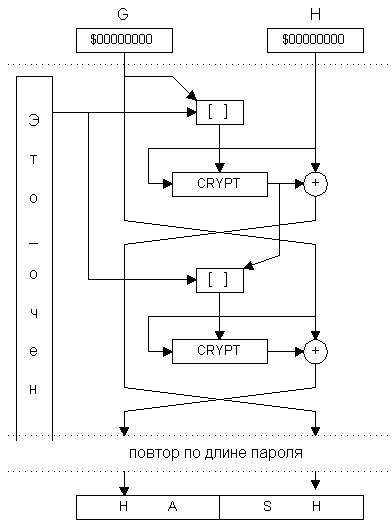
TMP=EnCrypt(H,[G,PSWj]); H'=H XOR TMP;

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

END;

Key=[Gk,Hk]

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.



**ХОД РАБОТЫ**

На рисунке 1 представлена главная форма программы.

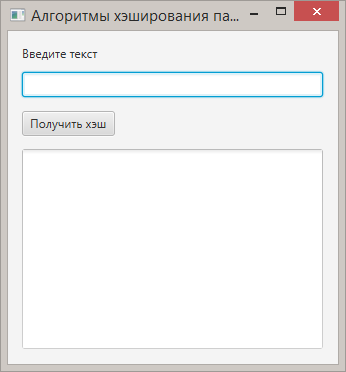


Рисунок 1 – Главная форма программы.

Пример работы программы представлен на рисунке 2.

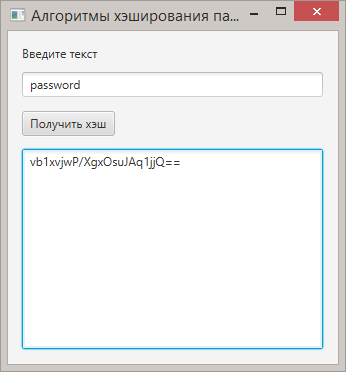


Рисунок 2 – Пример работы программы.

**ПРИЛОЖЕНИЕ А**

**Листинг класса StringHash.java**

package Models;

import javax.crypto.Cipher;

import javax.crypto.KeyGenerator;

import javax.crypto.SecretKey;

public class StringHash {

private Cipher ecipher;

private Cipher dcipher;

public StringHash() throws Exception {

this.ecipher = Cipher.getInstance("DES");

this.dcipher = Cipher.getInstance("DES");

SecretKey key = KeyGenerator.getInstance("DES").generateKey();

this.ecipher.init(Cipher.ENCRYPT\_MODE, key);

this.dcipher.init(Cipher.DECRYPT\_MODE, key);

}

public String encrypt(String str) throws Exception {

byte[] utf8 = str.getBytes("UTF8");

byte[] enc = this.ecipher.doFinal(utf8);

return new sun.misc.BASE64Encoder().encode(enc);

}

public String decrypt(String str) throws Exception {

byte[] dec = new sun.misc.BASE64Decoder().decodeBuffer(str);

byte[] utf8 = this.dcipher.doFinal(dec);

return new String(utf8, "UTF8");

}

}

**ПРИЛОЖЕНИЕ Б**

**Листинг класса SampleController.java**

package Forms;

import Models.StringHash;

import javafx.fxml.FXML;

import javafx.scene.control.Alert;

import javafx.scene.control.Button;

import javafx.scene.control.TextArea;

import javafx.scene.control.TextField;

public class SampleController {

@FXML

private TextField txtMessage;

@FXML

private Button btnGetHash;

@FXML

private TextArea txtResult;

@FXML

private void initialize() {}

@FXML

private void handleBtnGetHash() {

Alert alert;

String message = this.txtMessage.getText();

if (message.length() == 0) {

alert = new Alert(Alert.AlertType.ERROR);

alert.setTitle("Ошибка");

alert.setHeaderText("");

alert.setContentText("Введено пустое сообщение");

alert.showAndWait();

return;

}

try {

StringHash hash = new StringHash();

this.txtResult.setText(hash.encrypt(message));

} catch(Exception e) {

alert = new Alert(Alert.AlertType.ERROR);

alert.setTitle("Ошибка");

alert.setHeaderText("");

alert.setContentText(e.getMessage());

alert.showAndWait();

}

}

}

**ПРИЛОЖЕНИЕ В**

**Листинг класса Main.java**

import javafx.application.Application;

import javafx.fxml.FXMLLoader;

import javafx.scene.Parent;

import javafx.scene.Scene;

import javafx.stage.Stage;

public class Main extends Application {

@Override

public void start(Stage primaryStage) throws Exception {

Parent root = FXMLLoader.load(getClass().getResource("/Forms/sample.fxml"));

primaryStage.setTitle("Алгоритмы хэширования паролей");

primaryStage.setScene(new Scene(root));

primaryStage.show();

}

public static void main(String[] args) {

launch(args);

}

}