Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«Пермский национальный исследовательский политехнический университет»

Электротехнический факультет

Кафедра «Информационные технологии и автоматизированные системы»

Направление 09.03.04 – «Программная инженерия»

Дисциплина: «Защита информации»

Профиль: «Разработка программно-информационных систем»

Семестр 5

ОТЧЕТ

по лабораторной работе №7

Тема: «Алгоритмы хеширования паролей»

Выполнил: студент группы РИС-22-1б

Баяндин К. С. \_\_\_\_\_\_\_\_\_\_\_

Проверил: старший преподаватель кафедры ИТАС

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

Дата \_\_\_\_\_\_\_\_\_\_

Пермь, 2024

**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 (согласно варианту ЛР №4 – Реализовать шифрование текстового сообщения, методом блочного шифрования, используя блоки длиной 64 бит, ключ длиной 32 бит, реализуя в алгоритме шифрования методику DES.).

Максимальная длина пароля – 16 символов (7 вариант).

**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.

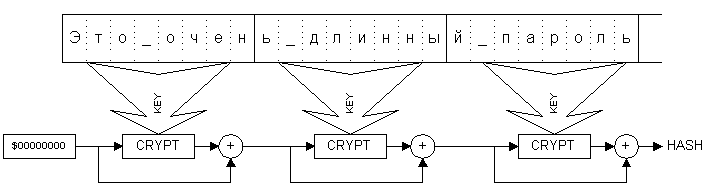
Характер применения блочного шифра для хеширования определяется отношением размера блока используемого криптоалгоритма и разрядности требуемого хеш-результата.

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

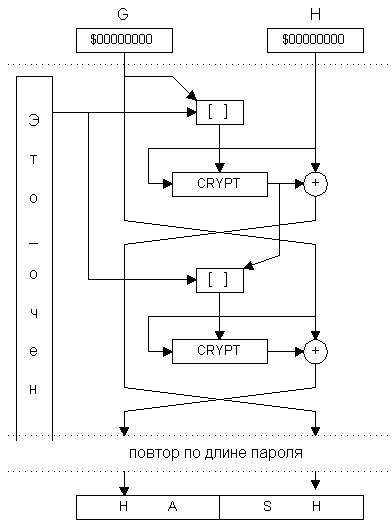
TMP=EnCrypt(H,[G,PSWj]); H'=H XOR TMP;

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**

Для работы был выбран язык C# и фреймворк Windows Forms.

Были созданы: окно для ввода пароля, кнопка получить хеш (хеширование пароля) и вывод полученного результата. Также внизу есть строка *Проверка пароля*, показывающая ошибки на вводе или при хешировании пароля.

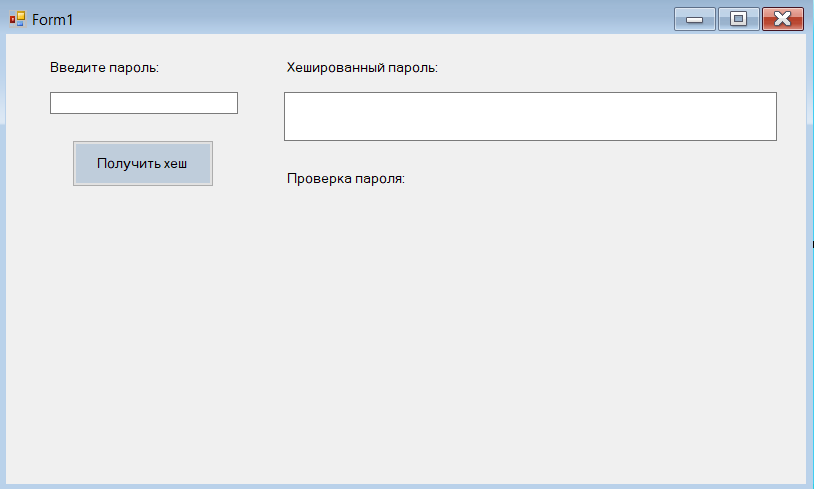


Рисунок 1 – Главная форма программы.

Результат работы представлен на рисунке 2.

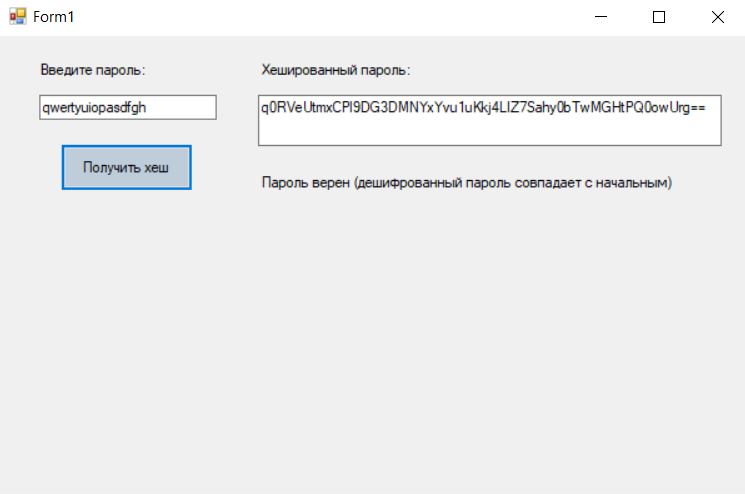


Рисунок 2 – Результат работы программы

Весь алгоритм представлен в классе *HashPassword*. Пароль преобразуется в двоичный код, далее генерируется случайный ключ и вектор инициализации. С помощью ключа пароль шифруется методом блочного шифрованя DES. Наконец, ключ, вектор инициализации и зашифрованный пароль объединяются в один байтовый массив – это и есть результат программы. Также есть метод *VerifyPassword()*, который дешифрует пароль и результат сравнивает с начальным паролем. Если они совпадают, то выводится сообщение о верности пароля, если нет – выводится ошибка.

К примеру, алгоритм DES не шифрует кириллицу. Если мы введем пароль на русском, то хеш будет неверным и при дешифровке пароль будет не совпадать с настоящим.

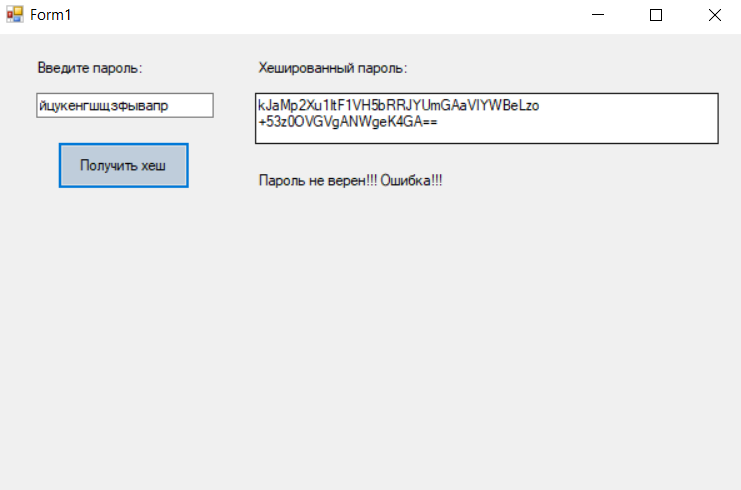


Рисунок 3 – Вывод ошибки при неверном хешировании пароля

**ЛИСТИНГ ПРОГРАММЫ**

**Class HashPassword.cs**

using System;

using System.Security.Cryptography;

using System.Text;

namespace \_7lab

{

internal class PasswordHash

{

private const int BlockSize = 8; // Размер блока в байтах (64 бита) - 4 вариант

public string HashPassword(string password)

{

// Преобразование пароля в байтовый массив

byte[] passwordBytes = Encoding.ASCII.GetBytes(password);

// Генерация случайного ключа и вектора инициализации (IV)

byte[] key = new byte[8]; // 64-битный ключ для DES

byte[] iv = new byte[BlockSize];

RandomNumberGenerator.Create().GetBytes(key);

RandomNumberGenerator.Create().GetBytes(iv);

// Создание объекта DES

DESCryptoServiceProvider des = new DESCryptoServiceProvider();

// Создание объекта шифрования

ICryptoTransform encryptor = des.CreateEncryptor(key, iv);

// Шифрование пароля

byte[] encryptedPassword = encryptor.TransformFinalBlock(passwordBytes, 0, passwordBytes.Length);

// Объединение ключа, IV и зашифрованного пароля в один байтовый массив

byte[] hashedPassword = new byte[key.Length + iv.Length + encryptedPassword.Length];

Array.Copy(key, 0, hashedPassword, 0, key.Length);

Array.Copy(iv, 0, hashedPassword, key.Length, iv.Length);

Array.Copy(encryptedPassword, 0, hashedPassword, key.Length + iv.Length, encryptedPassword.Length);

// Преобразование байтового массива в строку Base64

return Convert.ToBase64String(hashedPassword);

}

public bool VerifyPassword(string hashedPassword, string password)

{

// Разделение хешированного пароля на ключ, IV и зашифрованный пароль

byte[] hashedPasswordBytes = Convert.FromBase64String(hashedPassword);

byte[] key = new byte[8]; // 64-битный ключ для DES

Array.Copy(hashedPasswordBytes, 0, key, 0, 8);

byte[] iv = new byte[BlockSize];

Array.Copy(hashedPasswordBytes, 8, iv, 0, BlockSize);

byte[] encryptedPassword = new byte[hashedPasswordBytes.Length - 8 - BlockSize];

Array.Copy(hashedPasswordBytes, 8 + BlockSize, encryptedPassword, 0, encryptedPassword.Length);

// Создание объекта DES

DESCryptoServiceProvider des = new DESCryptoServiceProvider();

// Создание объекта дешифрования

ICryptoTransform decryptor = des.CreateDecryptor(key, iv);

try

{

// Дешифрование пароля

byte[] decryptedPassword = decryptor.TransformFinalBlock(encryptedPassword, 0, encryptedPassword.Length);

return Encoding.ASCII.GetString(decryptedPassword) == password;

}

catch (CryptographicException)

{

return false;

}

}

}

}