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-19-1б

Шеретов М.А. \_\_\_\_\_\_\_\_\_\_\_

Проверил: старший преподаватель

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

Дата \_\_\_\_\_\_

Пермь, 2022

**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. (16 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.

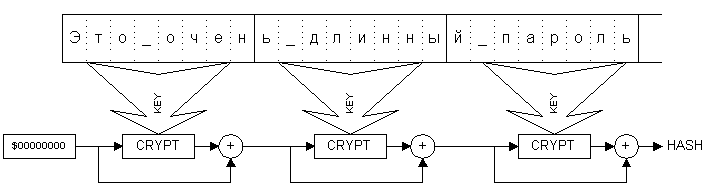
Характер применения блочного шифра для хеширования определяется отношением размера блока используемого криптоалгоритма и разрядности требуемого хеш-результата.

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

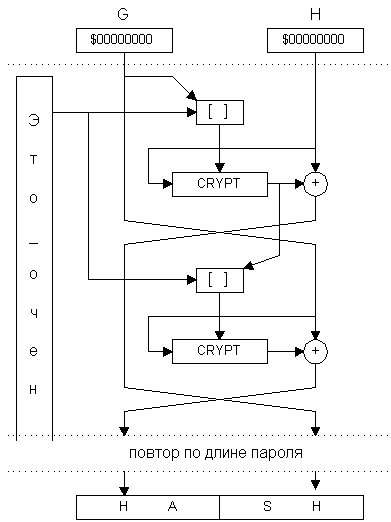
  TMP=EnCrypt(H,[G,PSWj]); H'=H XOR TMP;

  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**

Окно программы представлено на рисунке 1.

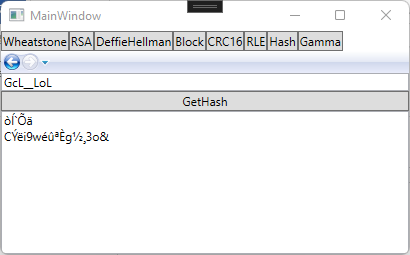


Рисунок 1. Окно программы

В первое поле вводится пароль длиной 16 символов (если символов не хватает, пароль дополняется случайными, или есть их больше 16, то обрезается). При нажатии на кнопку Get hash в поле Hash записывается полученный хеш пароля. Под полем отображаются его байты. В качестве ключа для шифрования используется сам пароль.

**ПРИЛОЖЕНИЕ А**

**Листинг А1 – Классы Crypto и CryptoResult**

import 'dart:convert';

import 'utils.dart';

class CryptoResult {

  final String value;

  final String key;

  CryptoResult(this.value, this.key);

}

class Crypto {

  static const blockLength = 64;

  static const keyLength = 56;

  static const bitNums = 8;

  static const shifts = [1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1];

  static const rounds = 16;

  List<String> keys = [];

  static String internals() {

    return 'Длина блока: $blockLength, длина ключа $keyLength, кол-во бит на символ: $bitNums, величина сдвига: $shifts, раунды: $rounds';

  }

  String \_expandBinary(String bs) {

    var d = bs.length % blockLength;

    bs = bs + List.filled(blockLength - d, '0').join();

    return bs;

  }

  String bytesToBinary(List<int> bytes) {

    var binaryList = bytes

        .map((e) => \_normalizeToSize(e.toRadixString(2), bitNums))

        .toList();

    return binaryList.join();

  }

  List<String> \_binaryToBlocks(String bs) {

    var blocks = <String>[];

    for (var i = 0; i < bs.length; i += blockLength) {

      blocks.add(bs.substring(i, i + blockLength));

    }

    return blocks;

  }

  String stringToBinary(String s) {

    var codes = utf8.encode(s);

    print(codes);

    var binStr = '';

    for (var code in codes) {

      var binCode = code.toRadixString(2);

      binCode = \_normalizeToSize(binCode, bitNums);

      binStr += binCode;

    }

    return binStr;

  }

  String binaryToString(String b) {

    var codes = <int>[];

    for (var i = 0; i < b.length; i += bitNums) {

      var code = int.parse(b.substring(i, i + bitNums), radix: 2);

      codes.add(code);

    }

    return String.fromCharCodes(codes);

  }

  List<int> binaryToBytes(String b) {

    var codes = <int>[];

    for (var i = 0; i < b.length; i += bitNums) {

      var code = int.parse(b.substring(i, i + bitNums), radix: 2);

      codes.add(code);

    }

    return codes;

  }

  /// XOR

  String \_xor(String s1, String s2) {

    var xorString = <String>[];

    for (var i = 0; i < s1.length; i++) {

      var b1 = s1[i] == '1' ? true : false;

      var b2 = s2[i] == '1' ? true : false;

      xorString.add(b1 ^ b2 ? '1' : '0');

    }

    return xorString.join();

  }

  String \_encryptOnce(String bInput, String bKey) {

    var bLeft = bInput.substring(0, bInput.length ~/ 2);

    var bRight = bInput.substring(bInput.length ~/ 2);

    return bRight + \_xor(bLeft, feistel(bRight, bKey));

  }

  String \_decryptOnce(String bInput, String bKey) {

    var bLeft = bInput.substring(0, bInput.length ~/ 2);

    var bRight = bInput.substring(bInput.length ~/ 2);

    return \_xor(feistel(bLeft, bKey), bRight) + bLeft;

  }

  String \_shiftLeft(String bKey, int shift) {

    for (var i = 0; i < shift; i++) {

      bKey = bKey + bKey[0];

      bKey = bKey.substring(1);

    }

    return bKey;

  }

  String permutation(String block, List<List<int>> matrix) {

    var sl = <String>[];

    for (var row in matrix) {

      for (var elem in row) {

        sl.add(block[elem - 1]);

      }

    }

    return sl.join();

  }

  String \_normalizeToSize(String bBlock, int size) {

    var zerosToAdd = size - bBlock.length;

    return List.filled(zerosToAdd, '0').join() + bBlock;

  }

  String feistel(String r, String key) {

    var rExp = permutation(r, Utils.expandMatrix);

    // print('feistel $rExp $key');

    var bStr = \_xor(rExp, key);

    var bVec = Utils.splitToChunks(bStr, 6);

    var i = 0;

    var res = '';

    for (var bi in bVec) {

      var a = int.parse(bi[0] + bi[bi.length - 1], radix: 2);

      var b = int.parse(bi.substring(1, bi.length - 1), radix: 2);

      var sMatrix = Utils.selectS(i);

      var n = sMatrix[a][b];

      var binN = \_normalizeToSize(n.toRadixString(2), 4);

      res += binN;

      i++;

    }

    return permutation(res, Utils.pPermutationMatrix);

  }

  String generateKey(String bKey) {

    var keyChunks = Utils.splitToChunks(bKey, 8);

    for (var i = 0; i < keyChunks.length; i++) {

      var oneCount = Utils.charsCount('1', keyChunks[i]);

      keyChunks[i] += oneCount % 2 == 0 ? '1' : '0';

    }

    bKey = permutation(keyChunks.join(), Utils.keyPermutationMatrix1);

    return bKey;

  }

  CryptoResult encrypt(String binaryData, String key) {

    var input = \_expandBinary(binaryData);

    keys.clear();

    var blocksBinary = \_binaryToBlocks(input);

    for (var i = 0; i < blocksBinary.length; i++) {

      blocksBinary[i] = permutation(blocksBinary[i], Utils.permutationMatrix1);

    }

    var bkey = stringToBinary(key);

    var cd = generateKey(\_normalizeToSize(bkey, 56));

    var cdSplit = Utils.splitToChunks(cd, cd.length ~/ 2);

    var c = cdSplit[0];

    var d = cdSplit[1];

    for (var r = 0; r < rounds; r++) {

      c = \_shiftLeft(c, shifts[r]);

      d = \_shiftLeft(d, shifts[r]);

      cd = permutation(c + d, Utils.cdPermutationMatrix);

      var roundKey = cd;

      keys.add(roundKey);

      // print('roundKey = $roundKey');

      for (var b = 0; b < blocksBinary.length; b++) {

        var tmpBlock = \_encryptOnce(blocksBinary[b], roundKey);

        blocksBinary[b] = \_normalizeToSize(tmpBlock, blockLength);

      }

    }

    for (var i = 0; i < blocksBinary.length; i++) {

      blocksBinary[i] = permutation(blocksBinary[i], Utils.permutationMatrix16);

    }

    key = binaryToString(cd);

    return CryptoResult(blocksBinary.join(), key);

  }

  CryptoResult decrypt(String input, String encryptedKey) {

    var blocksBinary = \_binaryToBlocks(input);

    for (var i = 0; i < blocksBinary.length; i++) {

      blocksBinary[i] = permutation(blocksBinary[i], Utils.permutationMatrix1);

    }

    for (var r = 0; r < rounds; r++) {

      var roundKey = keys[rounds - r - 1];

      // print('roundKey = $roundKey');

      for (var b = 0; b < blocksBinary.length; b++) {

        var tmpBlock = \_decryptOnce(blocksBinary[b], roundKey);

        blocksBinary[b] = \_normalizeToSize(tmpBlock, blockLength);

      }

    }

    for (var i = 0; i < blocksBinary.length; i++) {

      blocksBinary[i] = permutation(blocksBinary[i], Utils.permutationMatrix16);

    }

    return CryptoResult(blocksBinary.join(), '');

  }

}

**Листинг А2 – Класс Utils**

class Utils {

  static const permutationMatrix1 = [

    [58, 50, 42, 34, 26, 18, 10, 2],

    [60, 52, 44, 36, 28, 20, 12, 4],

    [62, 54, 46, 38, 30, 22, 14, 6],

    [64, 56, 48, 40, 32, 24, 16, 8],

    [57, 49, 41, 33, 25, 17, 9, 1],

    [59, 51, 43, 35, 27, 19, 11, 3],

    [61, 53, 45, 37, 29, 21, 13, 5],

    [63, 55, 47, 39, 31, 23, 15, 7]

  ];

  static const permutationMatrix16 = [

    [40, 8, 48, 16, 56, 24, 64, 32],

    [39, 7, 47, 15, 55, 23, 63, 31],

    [38, 6, 46, 14, 54, 22, 62, 30],

    [37, 5, 45, 13, 53, 21, 61, 29],

    [36, 4, 44, 12, 52, 20, 60, 28],

    [35, 3, 43, 11, 51, 19, 59, 27],

    [34, 2, 42, 10, 50, 18, 58, 26],

    [33, 1, 41, 9, 49, 17, 57, 25]

  ];

  static const keyPermutationMatrix1 = [

    [57, 49, 41, 33, 25, 17, 9],

    [1, 58, 50, 42, 34, 26, 18],

    [10, 2, 59, 51, 43, 35, 27],

    [19, 11, 3, 60, 52, 44, 36],

    [63, 55, 47, 39, 31, 23, 15],

    [7, 62, 54, 46, 38, 30, 22],

    [14, 6, 61, 53, 45, 37, 29],

    [21, 13, 5, 28, 20, 12, 4]

  ];

  static const expandMatrix = [

    [32, 1, 2, 3, 4, 5],

    [4, 5, 6, 7, 8, 9],

    [8, 9, 10, 11, 12, 13],

    [12, 13, 14, 15, 16, 17],

    [16, 17, 18, 19, 20, 21],

    [20, 21, 22, 23, 24, 25],

    [24, 25, 26, 27, 28, 29],

    [28, 29, 30, 31, 32, 1]

  ];

  static const pPermutationMatrix = [

    [16, 7, 20, 21, 29, 12, 28, 17],

    [1, 15, 23, 26, 5, 18, 31, 10],

    [2, 8, 24, 14, 32, 27, 3, 9],

    [19, 13, 30, 6, 22, 11, 4, 25]

  ];

  static const cdPermutationMatrix = [

    [14, 17, 11, 24, 1, 5, 3, 28, 15, 6, 21, 10, 23, 19, 12, 4],

    [26, 8, 16, 7, 27, 20, 13, 2, 41, 52, 31, 37, 47, 55, 30, 40],

    [51, 45, 33, 48, 44, 49, 39, 56, 34, 53, 46, 42, 50, 36, 29, 32]

  ];

  static const s1 = [

    [14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7],

    [0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8],

    [4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0,],

    [15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13],

  ];

  static const s2 = [

    [15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10],

    [3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5],

    [0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15],

    [13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9]

  ];

  static const s3 = [

    [10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4, 2, 8],

    [13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1],

    [13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7],

    [1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12]

  ];

  static const s4 = [

    [7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15],

    [13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 9],

    [10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4],

    [3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14]

  ];

  static const s5 = [

    [2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9],

    [14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6],

    [4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14],

    [11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3],

  ];

  static const s6 = [

    [12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5, 11],

    [10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8],

    [9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6],

    [4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8, 13]

  ];

  static const s7 = [

    [4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1],

    [13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6],

    [1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2],

    [6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12]

  ];

  static const s8 = [

    [13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7],

    [1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14, 9, 2],

    [7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8],

    [2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11]

  ];

  static List<List<int>> selectS(int sIdx) {

    return [s1, s2, s3, s4, s5, s6, s7, s8][sIdx];

  }

  static List<String> splitToChunks(String s, int chunks) {

    var l = <String>[];

    for (var i = 0; i < s.length; i += chunks) {

      l.add(s.substring(i, (i + chunks).clamp(0, s.length)));

    }

    return l;

  }

  static int charsCount(String char, String str) {

    int c = 0;

    for (var i = 0; i < str.length; i++) {

      if (str[i] == char) c++;

    }

    return c;

  }

}

**Листинг А3 – Интерфейс**

import 'dart:convert';

import 'dart:math';

import 'package:flutter/material.dart';

import 'package:zi6/crypto.dart';

void main() {

  runApp(const MyApp());

}

class MyApp extends StatelessWidget {

  const MyApp({Key? key}) : super(key: key);

  @override

  Widget build(BuildContext context) {

    return MaterialApp(

      title: 'Pass hash',

      theme: ThemeData(

        primarySwatch: Colors.blue,

      ),

      home: HomePage(),

    );

  }

}

class HomePage extends StatefulWidget {

  HomePage({Key? key}) : super(key: key);

  @override

  State<HomePage> createState() => \_HomePageState();

}

class \_HomePageState extends State<HomePage> {

  final inputController = TextEditingController();

  final hashController = TextEditingController();

  final crypto = Crypto();

  String bytesArray = '';

  bool useBase64 = false;

  List<int> bytesResult = [];

  final \_chars =

      'AaBbCcDdEeFfGgHhIiJjKkLlMmNnOoPpQqRrSsTtUuVvWwXxYyZz1234567890';

  final \_rnd = Random();

  String getRandomString(int length) => String.fromCharCodes(Iterable.generate(

      length, (\_) => \_chars.codeUnitAt(\_rnd.nextInt(\_chars.length))));

  void getHash() {

    if (inputController.text.isEmpty) return;

    var psw = inputController.text;

    if (psw.length > 16) {

      inputController.text = psw.substring(0, 16);

    } else if (psw.length < 16) {

      inputController.text = psw + getRandomString(16 - psw.length);

    }

    psw = inputController.text;

    var pswKey = psw.length > 7 ? psw.substring(0, 7) : psw;

    var binary = crypto.stringToBinary(inputController.text);

    var res = crypto.encrypt(binary, pswKey);

    bytesResult = crypto.binaryToBytes(res.value);

    setHash();

    setState(() {

      bytesArray = bytesResult.join(' ');

    });

  }

  void setHash() {

    if (useBase64) {

      hashController.text = base64.encode(bytesResult);

    } else {

      hashController.text = String.fromCharCodes(bytesResult);

    }

  }

  @override

  Widget build(BuildContext context) {

    return Scaffold(

      body: SizedBox.expand(

        child: Column(

          mainAxisAlignment: MainAxisAlignment.center,

          children: [

            SizedBox(

              width: 500,

              child: TextField(

                controller: inputController,

                decoration: const InputDecoration(labelText: 'Password'),

              ),

            ),

            TextButton(onPressed: getHash, child: const Text("Get hash")),

            SizedBox(

              width: 500,

              child: Row(

                children: [

                  Expanded(

                    child: TextField(

                      controller: hashController,

                      decoration: const InputDecoration(labelText: 'Hash'),

                    ),

                  ),

                  Checkbox(

                      value: useBase64,

                      onChanged: (v) {

                        setState(() {

                          useBase64 = v!;

                          setHash();

                        });

                      }),

                  const Text('use base64')

                ],

              ),

            ),

            Text(bytesArray)

          ],

        ),

      ),

    );

  }

}