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 09.03.04 – «Программная инженерия»

Дисциплина: «Защита информации»

Профиль: «Разработка программно-информационных систем»

Семестр 6

ОТЧЕТ

по лабораторной работе №4

Тема: «Блочные шифры»

Выполнил: студент группы АСУ-19-1б

Шеретов М.А. \_\_\_\_\_\_\_\_\_\_\_

Проверил: старший преподаватель

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

Дата \_\_\_\_\_\_

Пермь, 2021

**ЦЕЛЬ РАБОТЫ**

Получить практические навыки по созданию и применению блочных шифров.

**ЗАДАНИЕ**

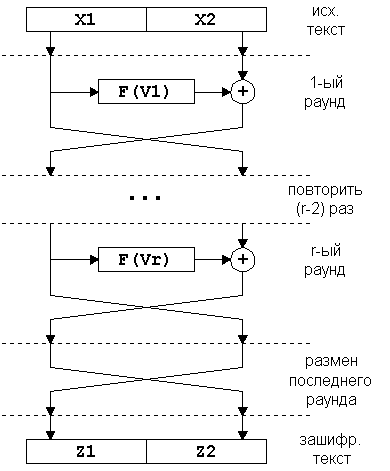
Реализовать шифрование текстового файла, методом блочного шифрования, используя блоки длиной 64 бит, ключ длиной 32 бит, реализуя в алгоритме шифрования методику DES.

**ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

**Сеть Фейштеля**

Суть метода - смешивания текущей части шифруемого блока с результатом некоторой функции, вычисленной от другой независимой части того же блока. Эта методика получила широкое распространение, поскольку обеспечивает выполнение требования о многократном использовании ключа и материала исходного блока информации.

Классическая сеть Фейштеля имеет следующую структуру:

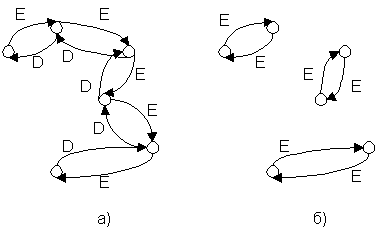
  
**Рис.1.**

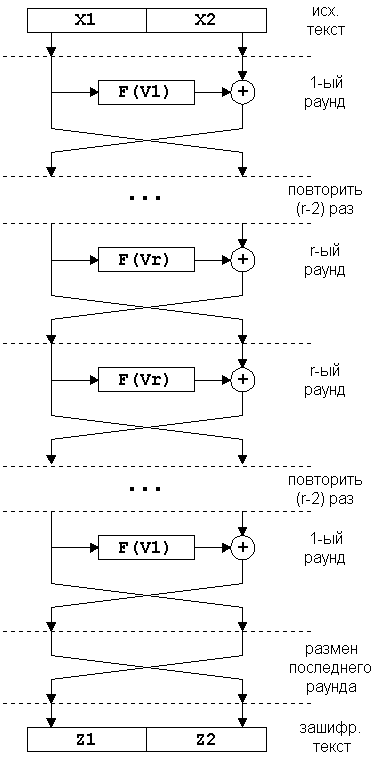
Независимые потоки информации, порожденные из исходного блока, называются ветвями сети. В классической схеме их две. Величины Vi именуются параметрами сети, обычно это функции от материала ключа. Функция F называется образующей. Действие, состоящее из однократного вычисления образующей функции и последующего наложения ее результата на другую ветвь с обменом их местами, называется циклом или раундом (англ. round) сети Фейштеля. Оптимальное число раундов K – от 8 до 32. Важно то, что увеличение количества раундов значительно увеличивает криптоскойстость любого блочного шифра к криптоанализу. Возможно, эта особенность и повлияла на столь активное распространение сети Фейштеля – ведь при обнаружении, скажем, какого-либо слабого места в алгоритме, почти всегда достаточно увеличить количество раундов на 4-8, не переписывая сам алгоритм. Часто количество раундов не фиксируется разработчиками алгоритма, а лишь указываются разумные пределы (обязательно нижний, и не всегда – верхний) этого параметра.

Сеть Фейштеля обладает тем свойством, что даже если в качестве образующей функции F будет использовано необратимое преобразование, то и в этом случае вся цепочка будет восстановима. Это происходит вследствие того, что для обратного преобразования сети Фейштеля не нужно вычислять функцию F-1.

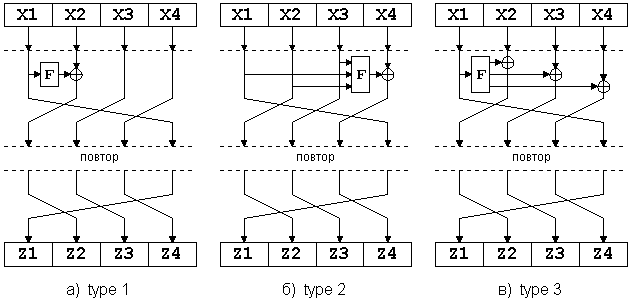
Более того, как нетрудно заметить, сеть Фейштеля симметрична. Использование операции XOR, обратимой своим же повтором, и инверсия последнего обмена ветвей делают возможным раскодирование блока той же сетью Фейштеля, но с инверсным порядком параметров Vi. Заметим, что для обратимости сети Фейштеля не имеет значение является ли число раундов четным или нечетным числом. В большинстве реализаций схемы, в которых оба вышеперечисленные условия (операция XOR и уничтожение последнего обмена) сохранены, прямое и обратное преобразования производятся одной и той же процедурой, которой в качестве параметра передается вектор величин Vi либо в исходном, либо в инверсном порядке.

С незначительными доработками сеть Фейштеля можно сделать и абсолютно симметричной, то есть выполняющей функции шифрования и дешифрования одним и тем же набором операций. Математическим языком это записывается как "Функция EnCrypt тождественно равна функции DeCrypt". Если мы рассмотрим граф состояний криптоалгоритма, на котором точками отмечены блоки входной и выходной информации, то при каком-то фиксированном ключе для классической сети Фейштеля мы будем иметь картину, изображенную на рис.2а, а во втором случае каждая пара точек получит уникальную связь, как изображено на рис. 2б. Модификация сети Фейштеля, обладающая подобными свойствами приведена на рисунке 3. Как видим, основная ее хитрость в повторном использовании данных ключа в обратном порядке во второй половине цикла. Необходимо заметить, однако, что именно из-за этой недостаточно исследованной специфики такой схемы (то есть потенциальной возможности ослабления зашифрованного текста обратными преобразованиями) ее используют в криптоалгоритмах с большой осторожностью.

  
**Рис.2.**

  
**Рис.3.**

А вот модификацию сети Фейштеля для большего числа ветвей применяют гораздо чаще. Это в первую очередь связано с тем, что при больших размерах кодируемых блоков (128 и более бит) становится неудобно работать с математическими функциями по модулю 64 и выше. Как известно, основные единицы информации обрабатываемые процессорами на сегодняшний день – это байт и двойное машинное слово 32 бита. Поэтому все чаще и чаще в блочных криптоалгоритмах встречается сеть Фейштеля с 4-мя ветвями. Самый простой принцип ее модификации изображен на рисунке 4а. Для более быстрого перемешивания информации между ветвями (а это основная проблема сети Фейштеля с большим количеством ветвей) применяются две модифицированные схемы, называемые «type-2» и «type-3». Они изображены на рисунках 4б и 4в.

  
**Рис.4.**

Сеть Фейштеля надежно зарекомендовала себя как криптостойкая схема произведения преобразований, и ее можно найти практически в любом современном блочном шифре. Незначительные модификации касаются обычно дополнительных начальных и оконечных преобразований (англоязычный термин – whitening) над шифруемым блоком. Подобные преобразования, выполняемые обычно также либо «исключающим ИЛИ» или сложением имеют целью повысить начальную рандомизацию входного текста. Таким образом, криптостойкость блочного шифра, использующего сеть Фейштеля, определяется на 95% функцией F и правилом вычисления Vi из ключа. Эти функции и являются объектом все новых и новых исследований специалистов в области криптографии.

**Пример блочного шифра - Блочный шифр DES**

Стандарт шифрования DES (Data Encryption Standard) был разработан в 1970-х годах, он базируется на алгоритме DEA.

Исходные идеи алгоритма шифрования данных DEA (data encryption algorithm) были предложены компанией IBM еще в 1960-х годах и базировались на идеях, описанных Клодом Шенноном в 1940-х годах. Первоначально эта методика шифрования называлась lucifer (разработчик Хорст Фейштель, название dea она получила лишь в 1976 году. Lucifer был первым блочным алгоритмом шифрования, он использовал блоки размером 128 бит и 128-битовый ключ. По существу этот алгоритм являлся прототипом DEA. В 1986 в Японии (NIT) разработан алгоритм FEAL(Fast data Encipherment ALgorithm), предназначенный для использования в факсах, модемах и телефонах (длина ключа до 128 бит). Существует и ряд других разработок.

DEA  оперирует с блоками данных размером 64 бита и использует ключ длиной 56 бит. Такая длина ключа соответствует 1017 комбинаций, что обеспечивало до недавнего времени достаточный уровень безопасности. В дальнейшем можно ожидать расширения ключа до 64 бит (например, LOKI) или вообще замены DES другим стандартом, например Советский шифр ГОСТ – 28147-89, алгоритм работы которого очень схож с DEC использует так же 64-битные блоки, но 256-битный ключ.

Входной блок данных, состоящий из 64 бит, преобразуется в выходной блок идентичной длины. Ключ шифрования должен быть известен, как отправляющей так и принимающей сторонам. В алгоритме широко используются перестановки битов текста.

Вводится функция f, которая работает с 32-разрядными словами исходного текста (А) и использует в качестве параметра 48-разрядный ключ (J). Схеме работы функции f показана на рис. 5. Сначала 32 входные разряда расширяются до 48, при этом некоторые разряды повторяются. Схема этого расширения показана ниже (номера соответствуют номерам бит исходного 32-разрядного кода).

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

Для полученного 48-разрядного кода и ключа выполняется операция исключающее ИЛИ (XOR). Результирующий 48-разрядный код преобразуется в 32-разрядный с помощью S-матриц. На выходе S-матриц осуществляется перестановка согласно схеме показанной ниже (числа представляют собой порядковые номера бит).

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

Преобразование начинается с перестановки бит (**IP – Initial Permutation)** в 64-разрядном блоке исходных данных. 58-ой бит становится первым, 50-ый – вторым и т.д. Схема перестановки битов показана ниже.

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

Полученный блок делится на две 32-разрядные части L0 и R0. Далее 16 раз повторяются следующие 4 процедуры:

Преобразование ключа с учетом номера итерации i (перестановки разрядов с удалением 8 бит, в результате получается 48-разрядный ключ)

Пусть A=Li, а J – преобразованный ключ. С помощью функции f(A,J) генерируется 32-разрядный выходной код.  
Выполняется операция XOR для Ri f(A,J), результат обозначается Ri+1.  
Выполняется операция присвоения Li+1=Ri.

Инверсная перестановка разрядов предполагает следующее размещение 64 бит зашифрованных данных (первым битом становится 40-ой, вторым 8-ой и т.д.).

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

S-матрицы представляют собой таблицы содержащие 4-ряда и 16 столбцов. Матрица S(1) представлена ниже (эта матрица, также как и те, что приведены в ссылке 2, являются рекомендуемыми).

No. 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15  
0 14 4 13 1 2 15 11 8 3 10 6 12 5 9 0 7  
1 0 15 7 4 14 2 13 1 10 6 12 11 9 5 3 8  
2 4 1 14 8 13 6 2 11 15 12 9 7 3 10 5 0  
3 15 12 8 2 4 9 1 7 5 11 3 14 10 0 6 13

Исходный 48-разрядный код делится на 8 групп по 6 разрядов. Первый и последний разряд в группе используется в качестве адреса строки, а средние 4 разряда – в качестве адреса столбца. В результате каждые 6 бит кода преобразуются в 4 бита, а весь 48-разрядный код в 32-разрядный (для этого нужно 8 S-матриц). Существуют разработки, позволяющие выполнять шифрование в рамках стандарта DES аппаратным образом, что обеспечивает довольно высокое быстродействие.

Преобразования ключей Kn (n=1,…,16; Kn = KS(n,key), где n – номер итерации) осуществляются согласно алгоритму, показанному на рис. 7.

Для описания алгоритма вычисления ключей Kn (функция KS) достаточно определить структуру «Выбора 1» и «Выбора 2», а также задать схему сдвигов влево. «Выбор 1» и «Выбор 2» представляют собой перестановки битов ключа. При необходимости биты 8, 16,…, 64 могут использоваться для контроля четности.

Для вычисления очередного значения ключа таблица делится на две части С0 и D0. В С0 войдут биты 57, 49, 41,…, 44 и 36, а в D0 – 63, 55, 47,…, 12 и 4. Так как схема сдвигов задана C1,D1; Cn, Dn и так далее могут быть легко получены из C0 и D0. Так, например, C3 и D3 получаются из C2 и D2 циклическим сдвигом влево на 2 разряда

|  |  |
| --- | --- |
| **PC-1 (Выбор 1)** | **PC-2 (Выбор 2)** |
| 57 49 41 33 25 17 9 | 14 17 11 24 1 5 |
| 1 58 50 42 34 26 18 | 3 28 15 6 21 10 |
| 10 2 59 51 43 35 27 | 23 19 12 4 26 8 |
| 19 11 3 60 52 44 36 | 16 7 27 20 13 2 |
| 63 55 47 39 31 23 15 | 41 52 31 37 47 55 |
| 7 62 54 46 38 30 22 | 30 40 51 45 33 48 |
| 14 6 61 53 45 37 29 | 44 49 39 56 34 53 |
| 21 13 5 28 20 12 4 | 46 42 50 36 29 32 |

|  |  |
| --- | --- |
| **Номер итерации** | **Число сдвигов влево** |
| 1 | 1 |
| 2 | 1 |
| 3 | 2 |
| 4 | 2 |
| 5 | 2 |
| 6 | 2 |
| 7 | 2 |
| 8 | 2 |
| 9 | 1 |
| 10 | 2 |
| 11 | 2 |
| 12 | 2 |
| 13 | 2 |
| 14 | 2 |
| 15 | 2 |
| 16 | 1 |

**ХОД РАБОТЫ**

На рисунке 1 представлено основное окно приложения.

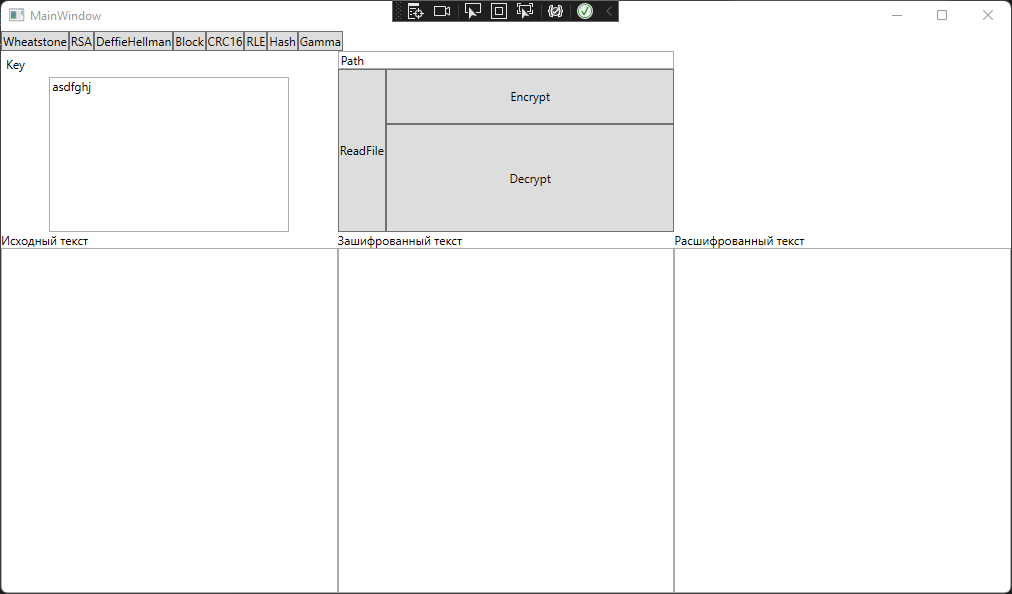
****

Рисунок 1. Окно приложения

В первое поле вводится путь к файлу, который надо считать. Файл считывается в виде массива байтов, а затем преобразуется в двоичный код (в данном примере файл считывается не весь, а только первые 100 байт, т.к. он очень большой).

В поле Key вводится ключ длиной 56 бит, как указано в DES. В данном случае вводится набор букв из 7 символов, который в дальнейшем преобразуется в двоичный код.

Код, считанный из файла, записывается в поле File data. При нажатии на кнопку Encrypt он шифруется и записывается в следующее поле Encrypted data. При нажатии на кнопку Decrypt зашифрованный код расшифровывается и записывается в Decrypted data. На рисунке 1 видно, что в конце код отличается от первоначального – это из-за того, что он был разбит на блоки по 64 бита и расширен до этого размера.

Также код можно преобразовать к текст, но это особо не имеет смысла. Операция представлена на рисунке 2.

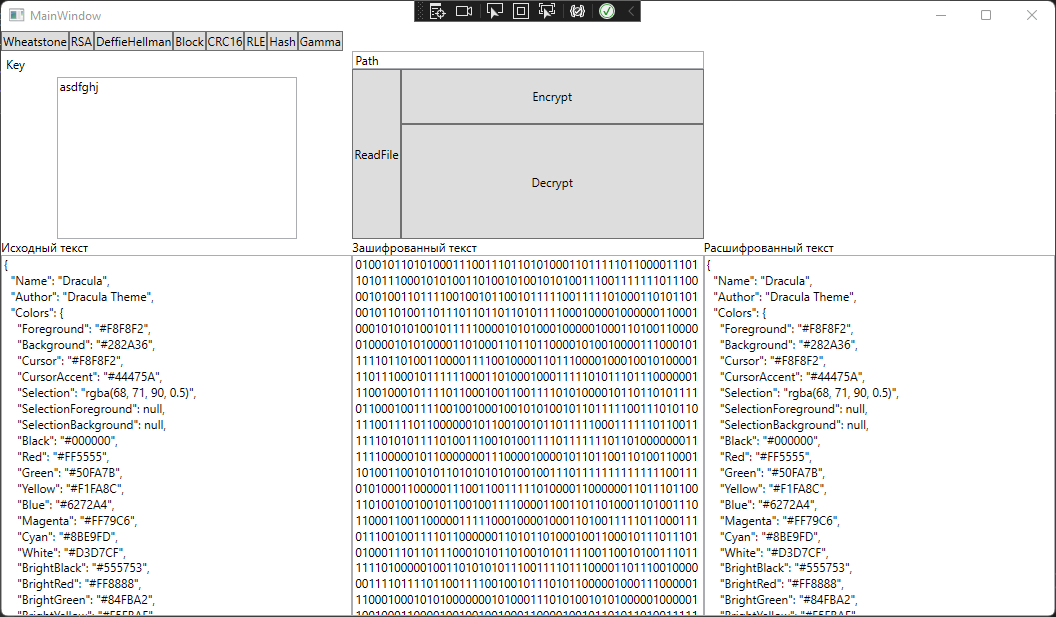


Рисунок 2. Преобразованный код

**ПРИЛОЖЕНИЕ А**

**Листинг А1 – Классы Crypto и CryptoResult**

import 'package:zi\_l4/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 = s.codeUnits;

    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);

  }

  /// 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 cd = generateKey(stringToBinary(key));

    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:io';

import 'package:flutter/material.dart';

import 'package:zi\_l4/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(

        theme: ThemeData(

          primarySwatch: Colors.blue,

        ),

        home: Home());

  }

}

class Home extends StatefulWidget {

  Home({Key? key}) : super(key: key);

  final crypto = Crypto();

  @override

  State<Home> createState() => \_HomeState();

}

class \_HomeState extends State<Home> {

  final keyFirstController = TextEditingController(text: 'Hkeyasd');

  final keySecondController = TextEditingController();

  final fileController = TextEditingController();

  final decodedController = TextEditingController();

  final encodedController = TextEditingController();

  final pathController =

    TextEditingController(text: r'D:\Steam\bin\gldriverquery.exe');

  String fileOrigin = '';

  String encodedOrigin = '';

  String decodedOrigin = '';

  bool fileCheck = false;

  bool encodedCheck = false;

  bool decodedCheck = false;

  String selectMask(String origin, bool isString) {

    return isString ? widget.crypto.binaryToString(origin) : origin;

  }

  void \_readFile() {

    var data = File(pathController.text).readAsBytesSync();

    fileOrigin = widget.crypto.bytesToBinary(data.getRange(0, 100).toList());

    fileController.text = selectMask(fileOrigin, fileCheck);

  }

  void \_encrypt() {

    var res = widget.crypto.encrypt(fileOrigin, keyFirstController.text);

    keySecondController.text = res.key;

    encodedOrigin = res.value;

    encodedController.text = selectMask(encodedOrigin, encodedCheck);

  }

  void \_decrypt() {

    var res = widget.crypto.decrypt(encodedOrigin, keySecondController.text);

    decodedOrigin = res.value;

    decodedController.text = selectMask(decodedOrigin, decodedCheck);

    writeToFile(r'D:\Flutter\proj\zi\_l4\lib\res', widget.crypto.binaryToString(decodedOrigin));

  }

  void writeToFile(String filePath, String content){

    File(filePath).writeAsString(content);

  }

  @override

  Widget build(BuildContext context) {

    return Scaffold(

      body: Center(

        child: Padding(

          padding: const EdgeInsets.all(16.0),

          child: Column(

            crossAxisAlignment: CrossAxisAlignment.center,

            children: [

              buildSmallTextField(pathController, 'Path', w: 600),

              Row(

                mainAxisAlignment: MainAxisAlignment.center,

                children: [

                  Padding(

                    padding: const EdgeInsets.all(8.0),

                    child: buildButton(\_readFile, 'Read file'),

                  ),

                  Padding(

                    padding: const EdgeInsets.all(8.0),

                    child: buildButton(\_encrypt, 'Encrypt'),

                  ),

                  Padding(

                    padding: const EdgeInsets.all(8.0),

                    child: buildButton(\_decrypt, 'Decrypt'),

                  ),

                ],

              ),

              Row(

                mainAxisAlignment: MainAxisAlignment.spaceAround,

                children: [

                  buildSmallTextField(keyFirstController, 'Key',

                      keyField: true),

                  buildSmallTextField(keySecondController, 'Last round key',

                      keyField: true, readOnly: true),

                ],

              ),

              Row(

                mainAxisAlignment: MainAxisAlignment.spaceAround,

                children: [

                  Checkbox(

                      value: fileCheck,

                      onChanged: (v) => setState(() {

                            fileCheck = v!;

                            fileController.text = selectMask(fileOrigin, fileCheck);

                          })),

                  Checkbox(

                      value: encodedCheck,

                      onChanged: (v) => setState(() {

                            encodedCheck = v!;

                            encodedController.text =selectMask(encodedOrigin, encodedCheck);

                          })),

                  Checkbox(

                      value: decodedCheck,

                      onChanged: (v) => setState(() {

                            decodedCheck = v!;

                            decodedController.text =selectMask(decodedOrigin, decodedCheck);

                          })),

                ],

              ),

              Expanded(

                child: Row(

                  mainAxisAlignment: MainAxisAlignment.spaceAround,

                  children: [

                    Expanded(

                        child: buildTextField(fileController, 'File data')),

                    Expanded(

                        child:

                            buildTextField(encodedController, 'Encrypted data')),

                    Expanded(

                        child:

                            buildTextField(decodedController, 'Decrypted data')),

                  ],

                ),

              ),

              Text(Crypto.internals())

            ],

          ),

        ),

      ),

    );

  }

  Widget buildButton(VoidCallback action, String text) {

    return ElevatedButton(

      onPressed: action,

      child: Text(text),

      style: ElevatedButton.styleFrom(shape: const StadiumBorder()),

    );

  }

  Widget buildTextField(TextEditingController c, String text) {

    return Padding(

      padding: const EdgeInsets.all(8.0),

      child: TextField(

          textAlignVertical: TextAlignVertical.top,

          controller: c,

          maxLines: null,

          expands: true,

          decoration: InputDecoration(

              labelText: text,

              border:

                  OutlineInputBorder(borderRadius: BorderRadius.circular(20)))),

    );

  }

  Widget buildSmallTextField(TextEditingController c, String text,

      {bool readOnly = false, bool keyField = false, double w = 200}) {

    return SizedBox(

      // height: 100,

      width: w,

      child: TextField(

          controller: c,

          maxLength:

              keyField ? Crypto.keyLength ~/ Crypto.bitNums : null,

          readOnly: readOnly,

          enabled: !readOnly,

          decoration: InputDecoration(

              labelText: text,

              border:

                  OutlineInputBorder(borderRadius: BorderRadius.circular(20)))),

    );

  }

}