Министерство образования Республики Беларусь Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ
И РАДИОЭЛЕКТРОНИКИ

КАФЕДРА ИНФОРМАТИКИ

Отчёт по лабораторной работе №1

По теме «Симметричная криптография. Стандарт шифрования ГОСТ 28147-89»

Выполнила: студентка гр. 053501 Шурко Т.А.

Проверил: ассистент кафедры информатики Лещенко Е. А.

Минск 2023

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Шифрование — обратимое преобразование информации в целях сокрытия от неавторизованных лиц, с предоставлением, в это же время, авторизованным пользователям доступа к ней. Главным образом, шифрование служит задачей соблюдения конфиденциальности передаваемой информации. Важной особенностью любого алгоритма шифрования является использование ключа, который утверждает выбор конкретного преобразования из совокупности возможных для данного алгоритма [1].

Пользователи являются авторизованными, если они обладают определенным аутентичным ключом. Вся сложность и, собственно, задача шифрования состоит в том, как именно реализован этот процесс.

В целом, шифрование состоит из двух составляющих — зашифрование и расшифрование.

С помощью шифрования обеспечиваются три состояния безопасности информации. Первой является конфиденциальность: шифрование используется для скрытия информации от неавторизованных пользователей при передаче или при хранении. Второе — целостность. Шифрование используется для предотвращения изменения информации при передаче или хранении. Последнее это идентифицируемость. Шифрование используется для аутентификации источника информации и предотвращения отказа отправителя информации от того факта, что данные были отправлены именно им.

Для выполнения лабораторной работы 1 необходимо реализовать программное средство зашифрования и расшифрования текстовых файлов при помощи стандартов шифрования ГОСТ 28147-89 в режиме простой замены.

1 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

1.1 Симметричная криптография. Стандарт шифрования ГОСТ 28147-89

ГОСТ 28147-89 «*Системы обработки информации. Защита криптографическая. Алгоритм криптографического преобразования*» — государственный стандарт СССР (а позже межгосударственный стандарт СНГ), описывающий алгоритм симметричного блочного шифрования и режимы его работы [2].

ГОСТ 28147-89 — блочный шифр с 256-битным ключом и 32 циклами (называемыми раундами) преобразования, оперирующий 64-битными блоками. Основа алгоритма шифра — сеть Фейстеля.

Выделяют четыре режима работы ГОСТ 28147-89:

- простой замены;
- гаммирование;
- гаммирование с обратной связью;
- режим выработки имитовставки.

Рассмотрим более подробно основной шаг криптопреобразования.

На первом шаге необходимо подготовить данные: разделить на блоки данных, каждый из которых должен иметь размер 64 бита. Далее этот блок разделяем на 2 блока N1 и N2 размером 32 бита.

На шаге 2 необходимо сложить по модулю 32 блок N1 с соответствующим элементом ключа.

На последующем шаге необходимо произвести замену: для этого необходима поблочная замена. 32-битовое значение, полученное на предыдущем шаге, интерпретируется как массив из восьми 4-битовых блоков кода: $S=(S_0, S_1, S_2, S_3, S_4, S_5, S_6, S_7)$, причем S_0 содержит 4 самых младших, а S_7 – 4 самых старших бита S. Далее значение каждого из восьми блоков заменяется новым, которое выбирается по таблице замен следующим образом: значение блока S_i меняется на S_i-тый по порядку элемент (нумерация с нуля) i-того узла замены (т.е. i-той строки таблицы замен, нумерация также с нуля). Другими словами, в качестве замены для значения блока выбирается элемент из таблицы замен с номером строки, равным номеру заменяемого блока, и номером столбца, равным значению заменяемого блока как 4-битового целого неотрицательного числа [1].

На 4 шаге результат шага 3 сдвигается циклически на 11 бит в сторону старших разрядов и передается на следующий шаг.

Далее побитовое сложение: значение, полученное на шаге 4, с блоком N2. Полученное значение преобразуемого блока возвращается как результат выполнения алгоритма основного шага криптопреобразования.

1.2 Блок-схема алгоритма

Блок-схема основного шага алгоритма изображена на рисунке 1. Всего таких циклов 32 для зашифрования и расшифрования, блок-схемы которых указаны на рисунках 2 и 3 соответственно [3].

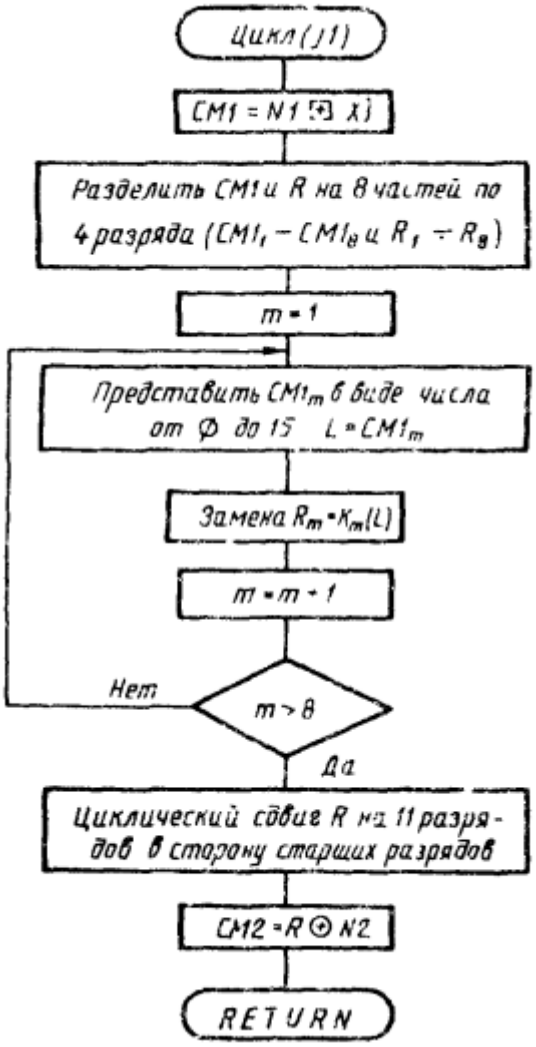

Рисунок 1 — Блок-схема одного цикла шифрования

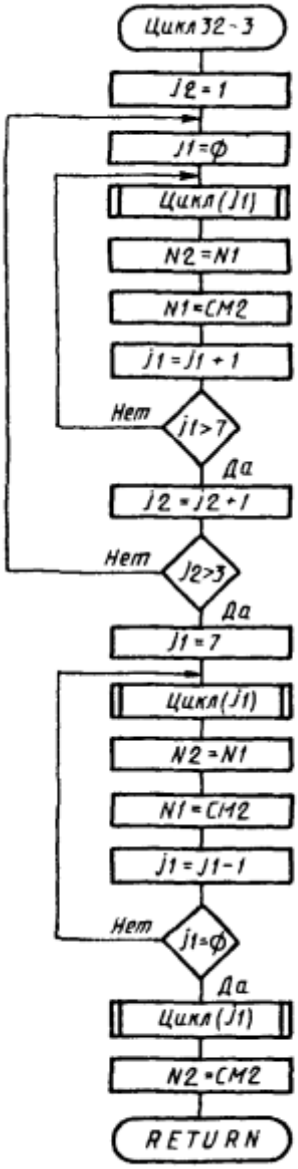

Рисунок 2 — Блок-схема 32-х циклов зашифрования

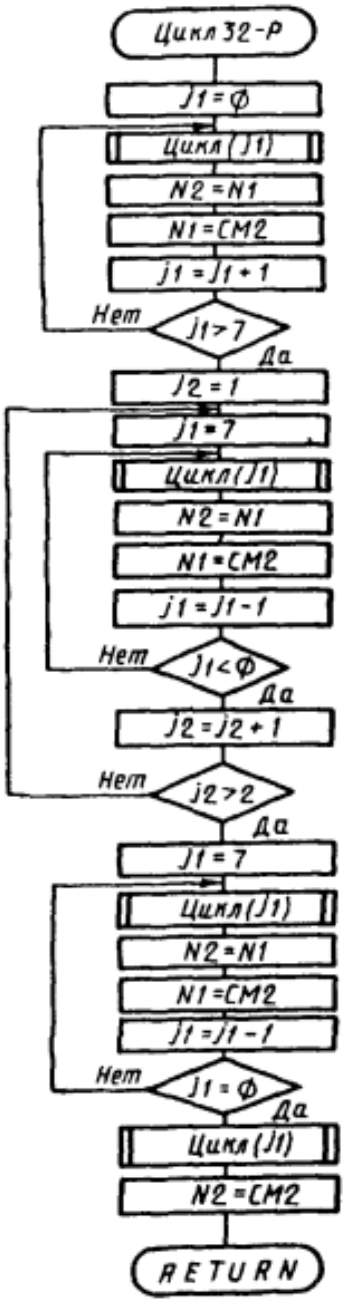

Рисунок 3 — Блок-схема 32-х циклов расшифрования

2 РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ ЗАДАЧИ

В результате выполнения задачи был реализован алгоритм простой замены на языке программирования C#. Текст для шифрования указан в приложении Б, текст после применения алгоритма в приложении В. Текст исполняемой программы в приложении А.

Для запуска программного продукта необходимо подключить класс Encryption, а также указать файл, из которого берется текст и файл, в который будет помещаться результат работы (Рисунок 4). Как параметр можно передать флаг false, который будет указывать на то, что необходимо дешифрование файла. По умолчанию флаг установлен на шифрование.

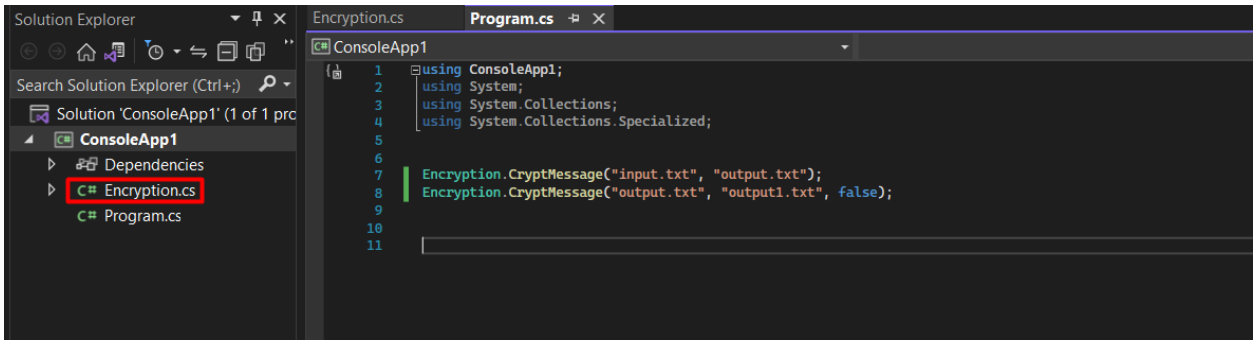

Рисунок 4 — Запускаемая программа

ЗАКЛЮЧЕНИЕ

При выполнении лабораторной работы был изучен стандарт шифрования ГОСТ 28147-89 в режиме прямой замены, рассмотрены и составлены блок-схема алгоритма, произведена его реализация на языке программирования C#, а также проверена работоспособность программного продукта.

Режим шифрования простой заменой имеет свои особенности. Так как блоки данных шифруются независимо друг от друга и от их позиции в массиве данных, при зашифровании двух одинаковых блоков открытого текста получаются одинаковые блоки шифртекста и наоборот. Отмеченное свойство позволит криптоаналитику сделать заключение о тождественности блоков исходных данных, если в массиве зашифрованных данных ему встретились идентичные блоки, что является недопустимым для серьезного шифра. Вторая особенность заключается в длине шифруемого текста, если она не кратна 8 байтам или 64 битам, возникает проблема, чем и как дополнять последний неполный блок данных массива до полных 64 бит. Эта задача не так проста, как кажется на первый взгляд. Очевидные решения типа «дополнить неполный блок нулевыми битами» или, более обще, «дополнить неполный блок фиксированной комбинацией нулевых и единичных битов» могут при определенных условиях дать в руки криптоаналитика возможность методами перебора определить содержимое этого самого неполного блока, и этот факт означает снижение стойкости шифра. Кроме того, длина шифртекста при этом изменится, увеличившись до ближайшего целого, кратного 64 битам, что часто бывает нежелательным [4][5].

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Библиотека rolc.ru [Электронный ресурс]: Библиотека rolc.ru — Алгоритм шифрования ГОСТ 28147-89, его использование и программная реализация для компьютеров платформы Intel x86. Режим доступа: https://kaf401.rloc.ru/Criptfiles/gost28147/GOST28147.htm (дата обращения: 10.09.2023).
[2] Википедия [Электронный ресурс]: Википедия — ГОСТ 28147-89. Режим доступа: https://ru.wikipedia.org/wiki/ГОСТ_28147-89 (дата обращения: 10.09.2023).
[3] Системы обработки информации. Защита криптографическая [Электронный ресурс]: Datasheet / АНО МЦК. — Режим доступа : https://files.stroyinf.ru/Data2/1/4294826/4294826631.pdf (дата обращения: 10.09.2023).
[4] Habr.com [Электронный ресурс]: Habr.com — ГОСТ 28147-89 (Часть 1. Введение и общие принципы). Режим доступа: https://kaf401.rloc.ru/Criptfiles/gost28147/GOST28147.htm (дата обращения: 10.09.2023).
[5] Habr.com [Электронный ресурс]: Habr.com — ГОСТ 28147-89 (Часть 2. Режим простой замены). Режим доступа: https://kaf401.rloc.ru/Criptfiles/gost28147/GOST28147.htm (дата обращения: 10.09.2023).

ПРИЛОЖЕНИЕ А
(обязательное)
Код программы

Encryption.cs

```
using System;
using System.Collections;
using System.ComponentModel;
using System.IO;

namespace ConsoleApp1 {
    public class Encryption {
        private static readonly byte[] key = {
                0x11, 0x12, 0x13, 0x14,
                0x22, 0x23, 0x24, 0x28,
                0x30, 0x35, 0x33, 0x39,
                0x62, 0x6e, 0x33, 0x73,
                0x79, 0x67, 0x61, 0x20,
                0x74, 0x74, 0x67, 0x69,
                0x65, 0x68, 0x65, 0x73,
                0x73, 0x3d, 0x2C, 0x20
        };
        private static readonly byte[][] replacementTable =
        {
            new byte[] { 0xF,0x1,0x2,0x3,0x4,0x5,0x6,0x7,0x8,0x9,0xA,0xB,0xC,0xD,0xE,0x0 },
            new byte[] { 0xF,0x1,0x2,0x3,0x4,0x5,0x6,0x7,0x8,0x9,0xA,0xB,0xC,0xD,0xE,0x0 },
            new byte[] { 0xF,0x1,0x2,0x3,0x4,0x5,0x6,0x7,0x8,0x9,0xA,0xB,0xC,0xD,0xE,0x0 },
            new byte[] { 0xF,0x1,0x2,0x3,0x4,0x5,0x6,0x7,0x8,0x9,0xA,0xB,0xC,0xD,0xE,0x0 },
            new byte[] { 0xF,0x1,0x2,0x3,0x4,0x5,0x6,0x7,0x8,0x9,0xA,0xB,0xC,0xD,0xE,0x0 },
            new byte[] { 0xF,0x1,0x2,0x3,0x4,0x5,0x6,0x7,0x8,0x9,0xA,0xB,0xC,0xD,0xE,0x0 },
            new byte[] { 0xF,0x1,0x2,0x3,0x4,0x5,0x6,0x7,0x8,0x9,0xA,0xB,0xC,0xD,0xE,0x0 },
            new byte[] { 0xF,0x1,0x2,0x3,0x4,0x5,0x6,0x7,0x8,0x9,0xA,0xB,0xC,0xD,0xE,0x0 }
        };

        private static readonly int sizeOfAccumulator = 4;
        private static byte[] n1 = new byte[sizeOfAccumulator];
        private static byte[] n2 = new byte[sizeOfAccumulator];

        private static byte[] mod2_32( byte[] a, byte[] b) {
            byte[] res;

            UInt32 A = BitConverter.ToUInt32(a, 0);
            UInt32 B = BitConverter.ToUInt32(b, 0);

            A += B;

            res = BitConverter.GetBytes(A);
```

```
    return res;
}

private static BitArray CyclicShift11(BitArray bitArr) {
    for (int k = 0; k < 11; k++) {
        if (bitArr.Count > 1) {
            var tmp = bitArr[bitArr.Count - 1];

            for (var i = bitArr.Count - 1; i != 0; --i) bitArr[i] = bitArr[i - 1];

            bitArr[0] = tmp;
        }
    }

    return bitArr;
}

private static byte[] ConvertBitsArrayIntoByteArray(BitArray bitsArr) {
    int byteArraySize = bitsArr.Count / 8;
    byte[] byteArr = new byte[byteArraySize];

    for (int l = 0; l < byteArraySize; l++) {
        byte sByte = 0x00;
        int bitIndex = 0;

        for (int m = 0; m < 8; m++) {
            if (bitsArr[l*8 + m]) {
                sByte |= (byte)(1 << bitIndex);
            }
            bitIndex++;
        }
        byteArr[l] = sByte;
    }

    return byteArr;
}
private static byte Convert4BitsIntoByte(BitArray bitsArr, int bitArrayIndex) {
    byte sByte = 0x00;
    int bitIndex = 0;

    for (int k = 0; k < sizeOfAccumulator ; k++) {
        if (bitsArr[bitArrayIndex + k]) {
            sByte |= (byte)(1 << bitIndex);
        }
        bitIndex++;
    }

    return sByte;
}
private static BitArray ConvertBitArrayToNormal(BitArray bitArr) {
```

```
        BitArray resultBitArr = new BitArray(bitArr.Count);

        for (int m = 0; m < resultBitArr.Count; m += (sizeOfAccumulator*2)) {
            for (int k = sizeOfAccumulator * 2 - 1; k >= 0; k--) {
                resultBitArr[m + sizeOfAccumulator * 2 - 1 - k] = bitArr[m + k];
            }

        }
        return resultBitArr;
    }

    private static void CryptBytes(BinaryWriter binaryWriter, byte[] n1, byte[] n2, bool status=true) {
        byte[] keyLine = new byte[sizeOfAccumulator];
        byte[] sum32 = new byte[sizeOfAccumulator];

        int keyIndex = 0;
        for (int i = 0; i < 32; i++) {
            if (status) {
                if (i == 24) keyIndex = 7;
            } else {
                if (i == 8) keyIndex = 7;
            }

            // срезаем нужную часть ключа, размером в 4 байта
            Array.Copy(key, keyIndex, keyLine, 0, sizeOfAccumulator);

            //суммируем по модулю
            sum32 = mod2_32(n1, keyLine);

                BitArray sum32BitsArray = new BitArray(sum32);
                //sum32BitsArray = ConvertBitArrayToNormal(sum32BitsArray);

                BitArray bitArrayResult = new BitArray(sum32BitsArray.Count, false);

                int index = 0, resultIndex = 0;
                int index_i = 0;

                for (int j = sum32BitsArray.Count - 1; j >= 0; j -= sizeOfAccumulator) {
                    // 7
                    index = (j + 1) / 4 - 1;
                    //размер 4 бита
                    //s0
                    int bitArrayIndex = index * sizeOfAccumulator;
                    byte sByte = Convert4BitsIntoByte(sum32BitsArray, bitArrayIndex);

                    byte replacementValue = replacementTable[index_i][Convert.ToInt16(sByte)];

                    BitArray bitReplacementArray = new BitArray( new byte[] { replacementValue });
```

```
            //bitReplacementArray = ConvertBitArrayToNormal(bitReplacementArray);

            for (int k = 0; k < sizeOfAccumulator; k++) {
               bitArrayResult[resultIndex + k] = bitReplacementArray[k];
            }

            resultIndex += sizeOfAccumulator;
            index_i++;
         }

         BitArray bitsAfterShift = CyclicShift11(bitArrayResult);

         BitArray bitsFromN2 = new BitArray(n2);
        // bitsFromN2 = ConvertBitArrayToNormal(bitsFromN2);

         for (int k = 0; k < bitsAfterShift.Count; k++) {
            sum32BitsArray[k] = bitsAfterShift[k] ^ bitsFromN2[k];
         }

         sum32 = ConvertBitsArrayIntoByteArray(sum32BitsArray);

         if (i < 31) {
            n2 = n1;
            n1 = sum32;
         } else {
            // n1 сохраняет старое значение
            n2 = sum32;
         }

         if (status) {
            if (i < 24) {
               keyIndex++;
               if (keyIndex > 7) keyIndex = 0;
            } else {
               keyIndex--;
               if (keyIndex < 0) keyIndex = 7;
            }
         } else {
            if (i < 8) {
               keyIndex++;
               if (keyIndex > 7) keyIndex = 0;
            } else{
               keyIndex--;
               if (keyIndex < 0) keyIndex = 7;
            }
         }
      }
      binaryWriter.Write(n1);
      binaryWriter.Write(n2);
```

```
            return;
        }

        public static void CryptMessage(string fileInputName, string fileOutputName, bool status=true) {
            using (FileStream outputFile = File.OpenWrite(fileOutputName)) {
                using (FileStream inputFile = File.OpenRead(fileInputName)) {

                    long fileSize = inputFile.Length; // размер файла в байтах
                    Console.WriteLine(fileSize);
                    long fileSizeBlock = fileSize;
                    // делаем размер кратным 8 (чтобы можно было разделить на блоки)
                    if (fileSize % 8 != 0) {
                        fileSizeBlock = fileSize + (8 - fileSize % 8);
                    }

                    using (BinaryReader binaryReader = new BinaryReader(inputFile)) {
                        using (BinaryWriter binaryWriter = new BinaryWriter(outputFile)) {
                            for (int p = 0; p < fileSizeBlock / 8; p++) {
                                // заполнение накопителей
                                n1 = binaryReader.ReadBytes(sizeOfAccumulator);
                                n2 = binaryReader.ReadBytes(sizeOfAccumulator);


                                if (n1.Length < sizeOfAccumulator) {
                                    byte[] n1Copy = new byte[sizeOfAccumulator];
                                    for (int i = 0; i < n1.Length; i++) {
                                        n1Copy[i] = n1[i];
                                    }
                                    for (int i = n1Copy.Length - 1; i < sizeOfAccumulator; i++) {
                                        if (i < 0) {
                                            i = 0;
                                        }
                                        n1Copy[i] = 0x00;
                                    }
                                    n1 = n1Copy;
                                }

                                if (n2.Length < sizeOfAccumulator) {
                                    byte[] n2Copy = new byte[sizeOfAccumulator];
                                    for (int i = 0; i < n2.Length; i++) {
                                        n2Copy[i] = n2[i];
                                    }
                                    for (int i = n2Copy.Length - 1; i < sizeOfAccumulator; i++) {
                                        if (i < 0) {
                                            i = 0;
                                        }
                                        n2Copy[i] = 0x00;
                                    }
                                    n2 = n2Copy;
```

```
                    }
                    CryptBytes(binaryWriter, n1, n2, status);
                }
            }
        }
    }

    return;
  }
 }
}
```

Program.cs

```
using ConsoleApp1;

Encryption.CryptMessage("input.txt", "output.txt");
Encryption.CryptMessage("output.txt", "output1.txt", false);
```

ПРИЛОЖЕНИЕ Б
(обязательное)
Текст для шифрования

The adults advised me to stop drawing snakes, from the inside or the outside. They told me that it was better to study geography, history, maths and grammar. That's why, at the age of six, I left a great career as a painter. I did it because my picture number one and picture number two were not successful when adults saw them. Adults never understand anything alone. And children are not happy when they have to always give them explanations.

So I had to choose another profession. I learnt to fly planes. I flew all over the world. And it's true that geography was very useful to me. I could see the difference between China and Arizona at first look. It is very useful if you are lost in the night.

During my life, I had a lot of contact with many serious people. I lived a lot among the adults. I could see them from a close distance. This experience did not improve my opinion of them much.

ПРИЛОЖЕНИЕ В
(обязательное)
Зашифрованный текст

!(єuntє¶Iъ¶_\X‹C:‹-й’ l "iO Г1ќчцЌ™Sa_л__Aљ0›,2_[й° _F©Ж?c[єй«{кk_§Ў_Ь,!ыqё_qGч*tO•sjДг¤_ШПt|ЎD‘ђюlы”,Ё›µ2_[cЖRA™§f ЧС” q_Ч{Ц’__>ЯЕ_*В_LHA—hKi_ш ыL_wO_Ци_К_ыЯГ;±_§Ё„ЭЄW°_ у2мЮЎЉ 01_м2Axyr_“U дKeЉЎ¬УфеЌЇц_
~Иўj¦‘ 7љ7§c'|Do2|лРфрогЯГ?>эR“u_O«’e¤‹c Ф_(н8}M_t u«©Ц9–{8©ъњXy h~НьмТ__…
O)_Ю: ђ•©~Лi_ЫI—;_l!Ёь / "=DePx>ыЄXwЯсч»dx5 гбIќ_0‡хш%__Nщ__ c_ OaГ7)Ћ№ыц№Ю0sиw&©ЪЋwъф›{яцn&P–
‹tЌ_‡V_@_CwT№бBD_){A°_Z¤92»ўћb${F{ъ
вЂCЃm·д”“/EI5_™э:ЃГ°љ’__Yko#¶г~_O¦”z_bJњЃЃ”0m„Ѓ…P_+
? --ЖOхљё)м__Я„_e < a
_ЂP‰Ї6v >”_ФdЃF0-ЎpЋZVJЊ«_ѓ_8___'Й‡{__k_oCT’_эЎљG;5 3f…Iз_rE,X_j□ °izб"и›_ЊCj„c„©Ou(ъqOGQ
Ниє¬/ NЛJ,i¬фPjEУ д__ъђ_¤|•pIyvl tI, i_(I†__cT±в¤АйГЭСЏР"в_·9’…ЧBtўtKtй№Ж,‘ъЎ_N3XЄkF_yгr¦_Й_Ѓ¤,ьщO_T ^&_oЁкеЁЮЛlЊ_y_T‡“K+7__†й"c©v16кб¶Д• Р†цр__#И-k_| ’_Ќ_эш1Лj
c¶Ю_[_TB
-НъTђ_<cЖ_I°»ФХщ p_c0ъ_>µb3л4ђ-c‹sФё_9ю"x_gjѓ_ЯQ6_M ЭЪ6ё ____»5к№‡&ЕЇ_@@и†_а• yvl tI,_'в†IчAzљ_®Г_8щG'gЪ°—
_µ{ъ~Ф_?мЏцDШ:·}z\Jм"«ю}»йђDYыr ^ яш…_Є_ЉД ~g_hч·$f}w
□_s_ ђ$K±_72)Э2Эњ