МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №2

“Изучение шифра DES”

Студент гр. 8382 Мирончик П.Д.

Преподаватель Племянников А.К.

Санкт-Петербург

2021

ЦЕЛЬ РАБОТЫ

Исследовать шифры DES, 3DES, а также другие модификации шифра DES: DESX, DESL, DESXL и получить практические навыки работы с ними, в том числе с использованием приложения Cryptool 1 и 2.

1. ИССЛЕДОВАНИЕ ПРЕОБРАЗОВАНИЙ DES

1.1. Задание

1. Изучить преобразования шифра DES с помощью демонстрационного приложения из Cryptool 1.

   a. Indiv.Procedures-> Visualization…-> DES…

2. Выполнить вручную преобразования первых двух раундов и вычисление раундовых ключей при следующих исходных данных:

   a. Открытый текст (не более 64 бит) – фамилия_имя (транслитерация латиницей)

   b. Ключ (56 бит) – номер зачетной книжки и инициал отчества (всего 7 символов)

3. Выполнить вручную обратное преобразование зашифрованного сообщения

4. Убедиться в совпадении результатов

1.2. Описание DES

DES шифрует информацию блоками по 64 бита с помощью 64-битного ключа шифрования. Шифрование выполняется следующим образом:

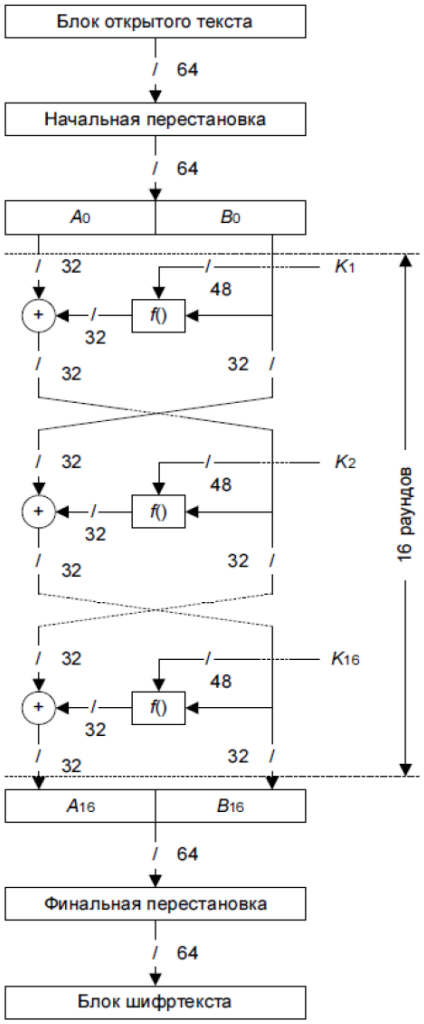

Рис.1.1 – шифр DES

1. Над 64-битными блоками производится начальная перестановка, задаваемая таблично

2. После начальной перестановки блок делится на 2 субблока по 32 бита (A_0 и B_0), над которыми производятся 16 раундов преобразований:

$$A_i = B_{i-1}$$

$$B_i = A_{i-1} \oplus f(B_{i-1}, K_i)$$

Где i – номер текущего раунда, K_i – ключ раунда, $\oplus$ - логическая операция XOR.

Схема работы функции раунда $f()$ выглядит следующим образом (Рис. 1.2):

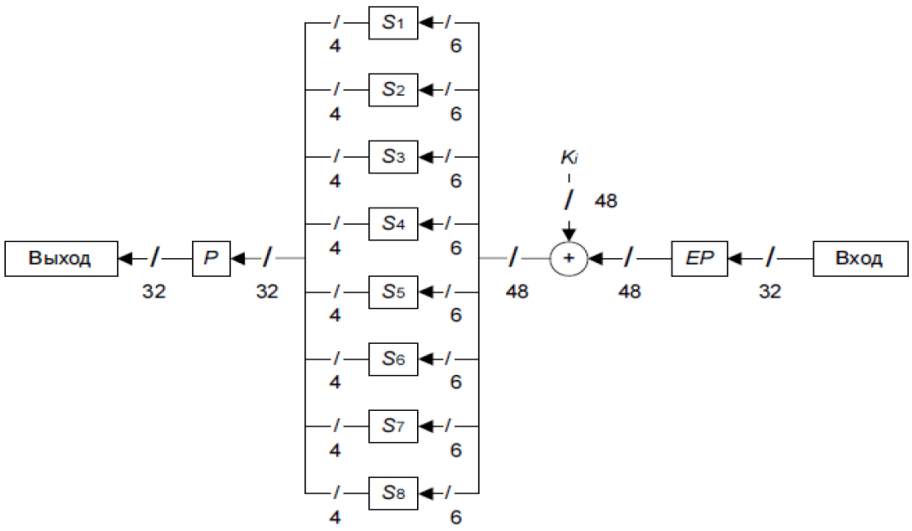

Рис.1.2 – Работа функции раунда DES

a) Выполняется расширяющая перестановка EP, преобразующая входные 32 бита в 48 бит.

b) Полученные 48 бит складываются с K_i операцией XOR.

c) Результат сложения разбивается на 8 блоков по 6 битов. Каждый блок обрабатывается соответствующей таблицей замен:

110110 001010 110110 010100 000100 100110 101001 010011

B[1] B[2] B[3] B[4] B[5] B[6] B[7] B[8]

10 = 1x2^1+0*2^0 = 2 1011 = 1*2^3+0*2^2+1*2^1+1*2^0 = 11

S-box 1:

row \ column	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	3

Рис.1.3 – Работа с таблицей замен в функции раунда

d) Над полученными 32 битами выполняется перестановка (обозначенная P на Рис. 1.2)

В итоге полученные субблоки A_{16} и B_{16} образуют 64-битный блок, над которым производится конечная перестановка и в итоге получается результирующий блок шифротекста.

Рассмотрим процедуру генерации раундовых ключей (Рис. 1.4):

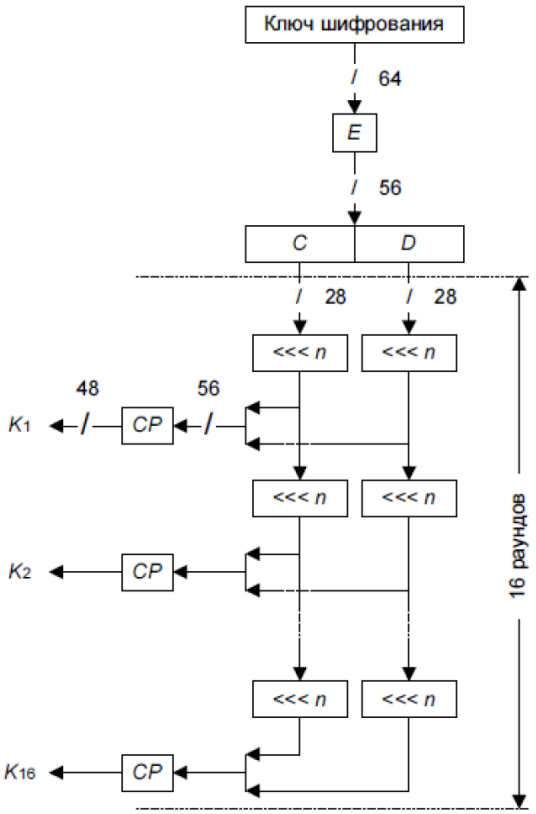

Рис. 1.4 – Генерация раундовых ключей

a) Выполняется операция E сжатия ключа и перестановка.

b) Полученные 56 бит делятся на 2 субблока C и D по 28 бит.

c) На каждом шаге C и D циклически сдвигаются на определенное количество бит, объединяются в 56-битное значение, к которому применяется сжимающая перестановка. В итоге получаются 48-битные ключи.

Расшифровывание данных алгоритмом DES происходит при прохождении всех шагов алгоритма в обратном порядке.

1.3. Ручной расчет субблоков и раундовых ключей шифра для первых двух раундов.

В качестве шифруемого текста использовалось *mironchi*, ключ - *838208d*. Было выполнено два раунда шифрования.

Для зашифрования и расшифрования алгоритмом DES была написана программа (Приложение А).

Рассмотрим генерацию ключей процесс генерации ключей. Сначала ключ побитно записывается построчно в матрицу 8×7:

0	0	1	1	1	0	0
0	0	0	1	1	0	0
1	1	0	0	1	1	1
0	0	0	0	0	1	1
0	0	1	0	0	0	1
1	0	0	0	0	0	0
1	1	1	0	0	0	0
1	1	0	0	1	0	0

Затем матрица дополняется нулями справа (правильно было бы дополнять битами так, чтобы каждый байт содержал нечетное число единиц, однако в нашем случае это ни на что не повлияет):

0	0	1	1	1	0	0	0
0	0	0	1	1	0	0	0
1	1	0	0	1	1	1	0
0	0	0	0	0	1	1	0
0	0	1	0	0	0	1	0
1	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0
1	1	0	0	1	0	0	0

К сформированной матрице применяется сжимающая перестановка E:

1	1	1	0	0	1	0
0	1	1	0	0	0	1
0	0	0	1	0	1	0
0	0	1	0	0	0	0
0	0	0	1	1	1	0
0	0	0	0	0	1	1
0	0	1	0	0	0	0
1	1	1	0	0	1	1

Далее формируются две матрицы C и D из 1-4 и 5-8 строк полученной матрицы:

C

1	1	1	0	0	1	0
0	1	1	0	0	0	1
0	0	0	1	0	1	0
0	0	1	0	0	0	0

D

0	0	0	1	1	1	0
0	0	0	0	0	1	1
0	0	1	0	0	0	0
1	1	1	0	0	1	1

Эти матрицы будут использоваться для формирования раундовых ключей описанном в п.1.2 способом. Сначала обе матрицы C и D сдвигаются на несколько элементов влево (в случае с первыми двумя раундами это 1 элемент), затем объединяются и к ним применяется сжимающая перестановка CP. В результате для первого и второго раундов были получены следующие ключи:

Round key 1

0	0	1	0	0	0
1	0	1	0	1	1
0	1	0	1	0	0
0	1	0	0	0	1
0	1	0	0	0	0
0	0	1	0	1	0
0	0	0	1	1	1
1	1	0	0	1	0

Round key 2

0	0	1	0	0	1
1	0	0	0	0	0
0	0	1	1	1	1
1	0	0	1	0	0
0	1	0	0	1	0
1	0	0	0	0	0
1	0	0	0	1	1
0	0	0	1	1	0

На этом процесс генерации ключей заканчивается. Перейдем к рассмотрению процесса зашифрования текста:

```
0  1  1  0  1  1  0  1
0  1  1  0  1  0  0  1
0  1  1  1  0  0  1  0
0  1  1  0  1  1  1  1
0  1  1  0  1  1  1  0
0  1  1  0  0  0  1  1
0  1  1  0  1  0  0  0
0  1  1  0  1  0  0  1
```

На первом шаге над битами исходного текста выполняется начальная перестановка IP:

```
1  1  1  1  1  1  1  1
0  0  0  0  0  1  0  0
0  0  0  1  1  0  0  1
1  0  1  0  1  0  1  1
0  0  0  0  0  0  0  0
1  1  1  1  1  1  1  1
1  1  0  1  1  0  1  1
0  0  1  1  1  1  0  0
```

И из полученной матрицы формируются субблоки A_0 и B_0:

```
          A0
-  -  -  -  -  -  -  -
1  1  1  1  1  1  1  1
0  0  0  0  0  1  0  0
0  0  0  1  1  0  0  1
1  0  1  0  1  0  1  1
-  -  -  -  -  -  -  -
          B0
-  -  -  -  -  -  -  -
0  0  0  0  0  0  0  0
1  1  1  1  1  1  1  1
1  1  0  1  1  0  1  1
0  0  1  1  1  1  0  0
-  -  -  -  -  -  -  -
```

Эти субблоки вместе с сформированными ранее ключами будут использованы в раундах шифрования (см. формулу в п.1.2). В итоге субблоки A и B имели следующие значения после выполнения раундов:

```
          Round 1
            A1
-  -  -  -  -  -  -  -
0  0  0  0  0  0  0  0
1  1  1  1  1  1  1  1
1  1  0  1  1  0  1  1
0  0  1  1  1  1  0  0
-  -  -  -  -  -  -  -
            B1
-  -  -  -  -  -  -  -
0  1  1  0  0  1  0  0
0  0  1  0  0  0  1  1
0  0  1  0  0  0  1  1
0  1  0  1  1  0  1  1
-  -  -  -  -  -  -  -
          Round 2
            A2
-  -  -  -  -  -  -  -
0  1  1  0  0  1  0  0
0  0  1  0  0  0  1  1
0  0  1  0  0  0  1  1
0  1  0  1  1  0  1  1
-  -  -  -  -  -  -  -
            B2
-  -  -  -  -  -  -  -
0  1  1  0  0  0  1  0
0  0  1  0  0  0  1  0
1  1  1  0  1  1  1  1
0  0  1  1  0  1  0  1
-  -  -  -  -  -  -  -
```

Блоки A_2 и B_2 далее были объединены и к полученному блоку применена конечная перестановка IP^{-1}:

```
0  0  0  1  1  1  1  1
1  0  1  1  1  1  0  1
0  1  0  0  1  0  1  0
0  0  0  0  1  0  0  1
0  0  0  0  0  0  1  1
1  1  1  1  1  1  1  0
1  1  0  0  1  0  0  1
0  0  0  0  1  0  0  0
```

или `31  189  74  9  3  254  201  8` в числовом представлении – это и есть зашифрованный текст.

Обратная перестановка отличается только формулой раунда и очередностью выбора ключей (раундовые ключи берутся в обратном порядке):

$$A_{i-1} = B_i \oplus f(A_i, K_i)$$

$$B_{i-1} = A_i$$

В результате текст был успешно расшифрован.

2. ИССЛЕДОВАНИЕ DES В РЕЖИМАХ ECB И CBC

2.1. Задание

1. Создать картинку со своими ФИО (формат bmp).

2. Зашифровать картинку шифром DES в режиме ECB.

3. Зашифровать картинку шифром DES в режиме CBC с тем же ключом.

4. Сохранить скриншоты картинок для отчета.

5. Сжать исходную и 2 зашифрованных картинки средствами CrypTool. Зафиксировать размеры полученных файлов в таблице.

6. Выбрать случайный текст на английском языке (не менее 1000 знаков) и зашифровать его DES в режиме ECB.

7. Для одного и того же шифротекста оценить время проведения атаки «грубой силы» в случаях, когда известно n-4, n-6, n-8,.., 2 байт секретного ключа. Зафиксировать результаты измерений в таблице.

8. Повторить подобные измерения для DES в режиме CBC.

2.2. Основные параметры шифров

Рис. 2.1 – описание режима ECB

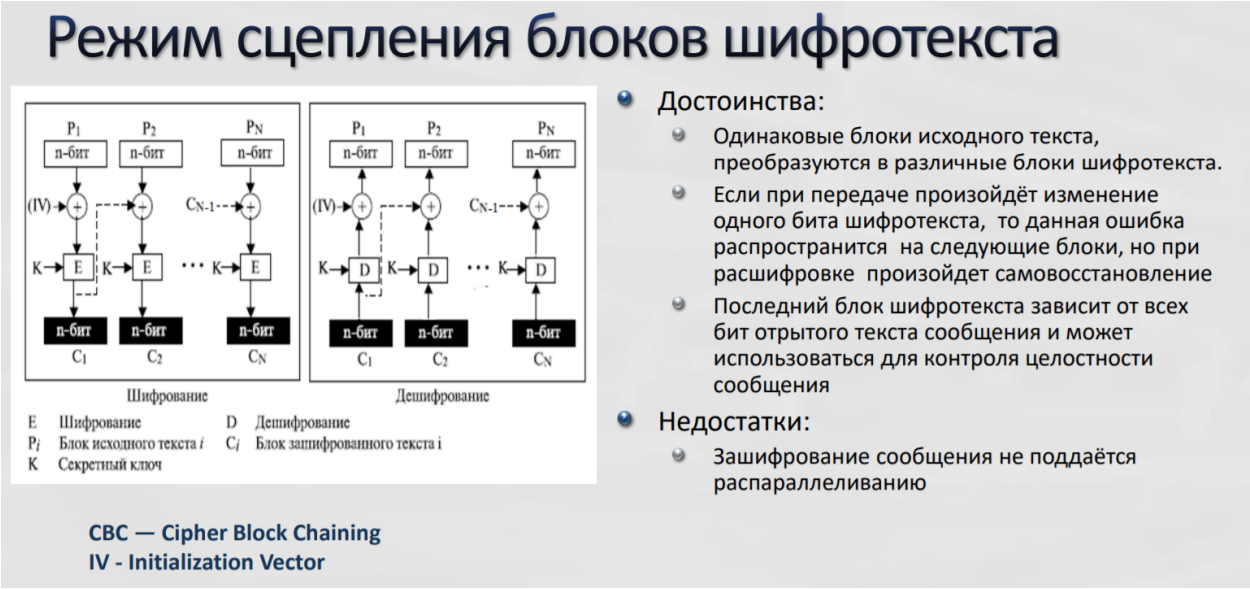

Рис. 2.2 – описание режима CBC

2.3. Обработка изображений

Обработаем изображение, представленное на рисунке 2.3.

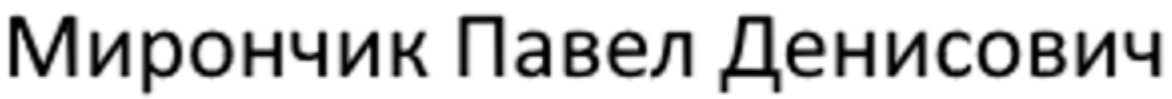

Рис. 2.3 – Исходное изображение

Используем ключ шифрования `08 03 08 02 00 08 89 17`. Результаты шифрования представлены на рис. 2.4 – ECB, и 2.5 - CBC.

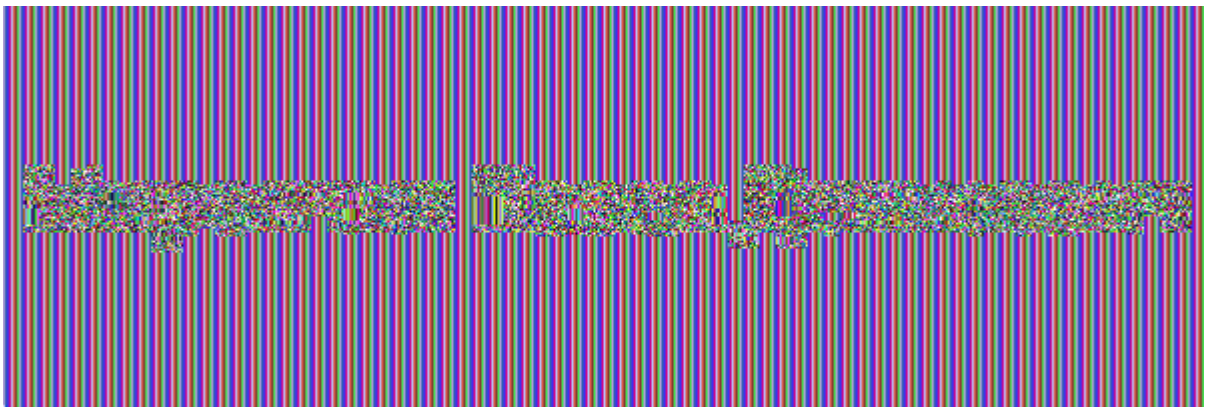

Рис. 2.4 – Зашифрованное в режиме ECB изображение

Рис.2.5 – Зашифрованное в режиме CBC изображение

Заметно, что в режиме ECB получается гораздо меньше шума, и появляется возможность примерно определить содержимое исходного изображения.

Попробуем теперь сжать изображения средствами CtypTool 1.

Исходное изображение удалось сжать в 10 раз – эффективность сжатия 90%.

Эффективность сжатия изображения в режиме ECB составила 86% - это связано с тем, что шифрование ECB привело к созданию множества идентичных блоков, которые можно эффективно сжать.

В то же время, результат зашифрования режима CBC сжать не удалось совсем – эффективность сжатия составила 0%.

2.4. Обработка текстовых данных

Возьмем в качестве исходного текста текст начала статьи *What's New in Flutter 2* (https://medium.com/flutter/whats-new-in-flutter-2-0-fe8e95ecc65),

первые 1000 символов. Для шифрования используем тот же ключ, что и в пункте 2.3: `08 03 08 02 00 08 89 17`.

Проведем атаку грубой силы на зашифрованный в режиме ECB текст. Полученные результаты записаны в табл. 2.1.

Табл. 2.1 – Оценка атаки грубой силы для ECB

Количество известных байт ключа	Примерное время атаки
2	300 дней
4	25 минут

Проведем аналогичные измерения для текста, зашифрованного в режиме CBC (табл. 2.2).

Табл. 2.2 – Оценка атаки грубой силы для CBC

Количество известных байт ключа	Примерное время атаки
2	1.3 года
4	41 минута

3. ИССЛЕДОВАНИЕ 3-DES

3.1. Задание

1. Выбрать случайный текст на английском языке (не менее 1000 знаков).

2. Создать бинарный файл с этим текстом, зашифровав и расшифровав его DES на 0-м ключе.

3. Снять и сохранить частотную и автокорреляционную характеристику этого файла.

4. Зашифровать бинарный файл шифром 3-DES в режиме ECB.

5. Снять и сохранить частотную и автокорреляционную характеристику файла с шифровкой.

6. Зашифровать исходный бинарный файл 3-DES в режиме CBC с тем же ключом.

7. Снять и сохранить частотную и автокорреляционную характеристику файла с шифровкой.

8. Определить экспериментальным путем по какой схеме работает реализация 3-DES в CrypTool. Сохранить подтверждающие скриншоты.

3.2. Основные параметры и обобщенная схема шифра

Основные параметры и обобщенная схема шифра представлены на рис. 3.1.

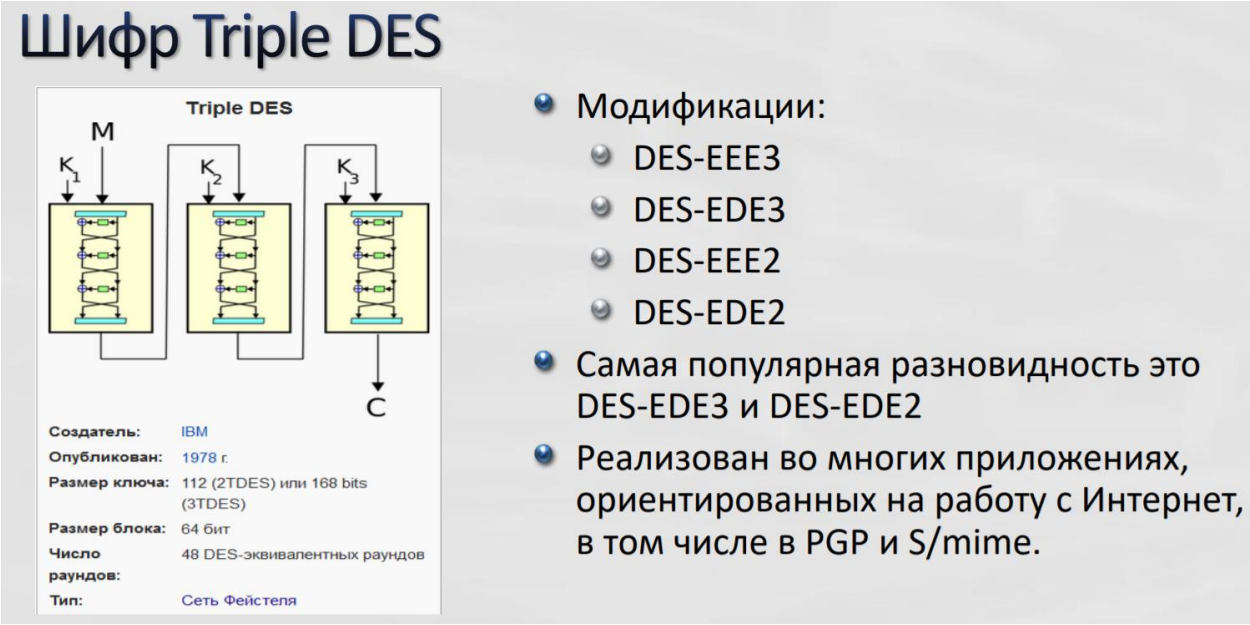

Рис. 3.1 – Основные параметры и обобщенная схема шифра

3.3. Примеры использования 3DES в CrypTool 1

В качестве исходного текста будем использовать тот же текст, что и в пункте 2 (рис. 3.2). Ключ для всех используемых режимов будет `01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16`.

```
Today, we...re pleased to announce the release of
 Flutter 2. It...s been a little more than two ye
ars since the Flutter 1.0 release, but in that sh
ort time, we...ve closed 24,541 issues and merged
 17,039 PRs from 765 contributors. Just since the
 Flutter 1.22 release in September, we...ve close
d 5807 issues and merged 4091 PRs from 298 contri
butors. Special thanks go out to our volunteer co
ntributors who generously give their spare time t
o improve the Flutter project. The top volunteer
contributors for the Flutter 2 release were xu-ba
olin with 46 PRs, al4n with 32 PRs that focused o
n bringing Flutter to null safety, and hamdikahlo
un with 20 PRs that improved a number of the Flut
ter plugins. But it...s not just coders that cont
ribute to the Flutter project; a great set of vol
unteer PR reviewers were also responsible for rev
iewing 1525 PRs, including hamdikahloun (again!),
 CareF and YaseedAlKhalaf (who...s only 16!). Flu
tter is truly a community effort and we couldn...
t have gotten to version 2 without .
```

Рис. 3.2 – Исходный текст

Зашифруем исходный текст с использованием ECB и CBC режимов 3DES шифра (Рис. 3.3 и Рис. 3.4).

```
o/al .....'.F.0r.....V.3....s\|dG.".b...#.S.~.&
.J..9U...V.p..$....j..N...Z..>[J.cc.[O\#d..q.aK@.
.P.l.BL.d-..........D.....7>...u.....|,.9....Z{.
W.46.F.AvB....M....:T..Y..M.}[f....HpJ.....'.I.d.
D..kO...l.L.*c-.(]&....D...ot.>2C..}....n+....b..
......Z..I8bj.*.[...^..n.;R.....b!...%.,......#|.
...l....6.Ux.Q...e.xKT...c.l..agH.....:..4...D...
o.F.G.Q.....w.v.............!{ln..?....Uf3TK..+
T..:.C....b....o...v..[P..l.86.....3k..s....ey...
9|t.:.....J?...,W}Z.u..c.lT...?..s..UX..IA..J..;.
.M.....E5%.R..U=...M$i.^...l+.~...K.Zh.x.l..s.T?e
5..........i..`.RN.w......%..?....P-..0E.....4..>
?]_.i........@.....s.8.u....g+X5.sM....D...B.}e.&
..+..:.`...:./....X.<v.iBJ.B..hs8..M:..U...),..K.
"(....^..%C..............7#...d. ......0...*?...*
r.4...kC....i.......l...Fl...uw....\.o..n}s.'d..9
J...M...}..4E...a:->._.A.[...$)K.-.~S..IK.3.1]IC.
>C.....v.f9...>*...lo.pgG..P.4....x..A*..)<.)..8.
u.<l..<q.[..7.w..-........6`j..WA..3.......7W....
....D+cG.w......=V..E...R..F...........3,..@.X..s
.......P....(.(.p.....L.2..K.._.....
```

Рис. 3.3 – текст, зашифрованный 3DES ECB

```
o/al .....A..H.....2.{.D$>...@.G....A.D.`....l..W
j..?9....l.....$....(.......TjY..5....#."~2..q.
qp..}....lQ...b|....U-C..?%....t...r.8..........
.YN.?..4@v7....5.Y......P;.8:.M...s`......,..+...
..)...6..4.......C.;."B|w.......7.a....'_..a.}^..
i..6ntdY..<..........OOO.*.2.d)\J...../.0!...G...
.yz.....E..Q..V........o..3...6.[....~Tq...|.NdO.
........_...H.A.......,.k.f....z.g.2....v.E..d..AV
.8.....F..O......l...$L.......\0 }.5...g...B6..@?
 ...#g.+.AX.d.........Y*R...X..}....4...,.....e.O.
.........0>.GK.\Z...LW..>....;B.:.........t4....Il
.j..w.....*.-7(..._.F.A.....Z(.&.dW3..t..O.H.....
.?.......z0G...`..h.../by,Y0...FJ..c....Tj$h._Q.@
....&*...jL........T.D.s.Q.M.<(.S....L_.4B....@.l
.u..L.t...`..).5..."5.7\.E....>f....>............
'.U.........d..NE....q.K.......f ...|.CR..;..B.=.
J...6q.P?.neKS4O&......|(.8r...H..]......5}...V...
M..j8Rrmt.LZo...Z4e.Fl....~...S...~e.B/}....b&A.T
._t...f.....4 M.. &..*.Kl.r.t..j.T...!Qo..e......
.P...<..L..5.../".V]<..i0../...\.,.T|.)...4n..5L.
..-.g..%........G.bX......KZ.....+.%R...*.H.
```

Рис. 3.4 – текст, зашифрованный 3DES CBC

Построим также частотные и автокорреляционные характеристики для зашифрованных текстов (Рис.3.5 – Рис.3.8).

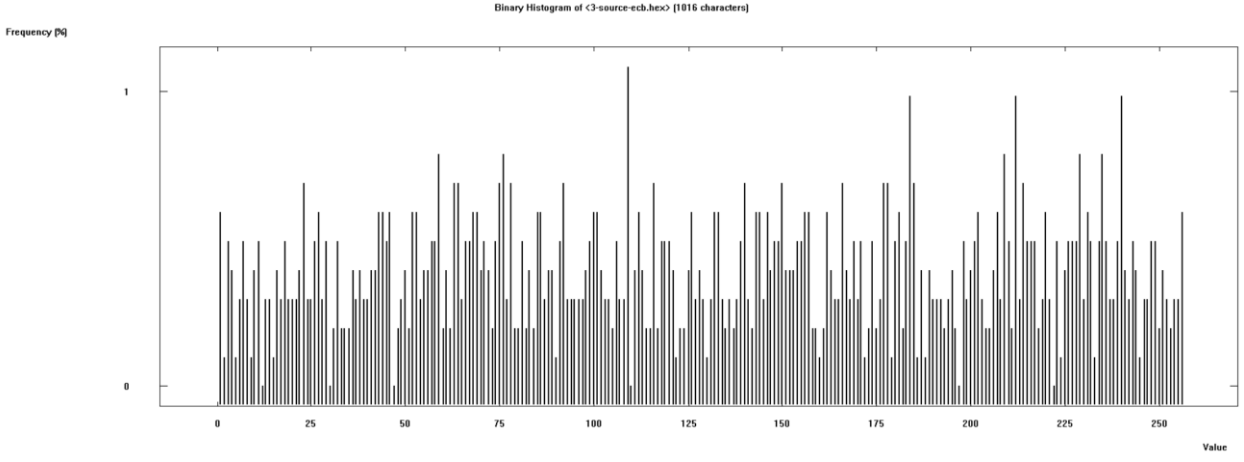

Рис. 3.5 – частотная характеристика 3DES ECB

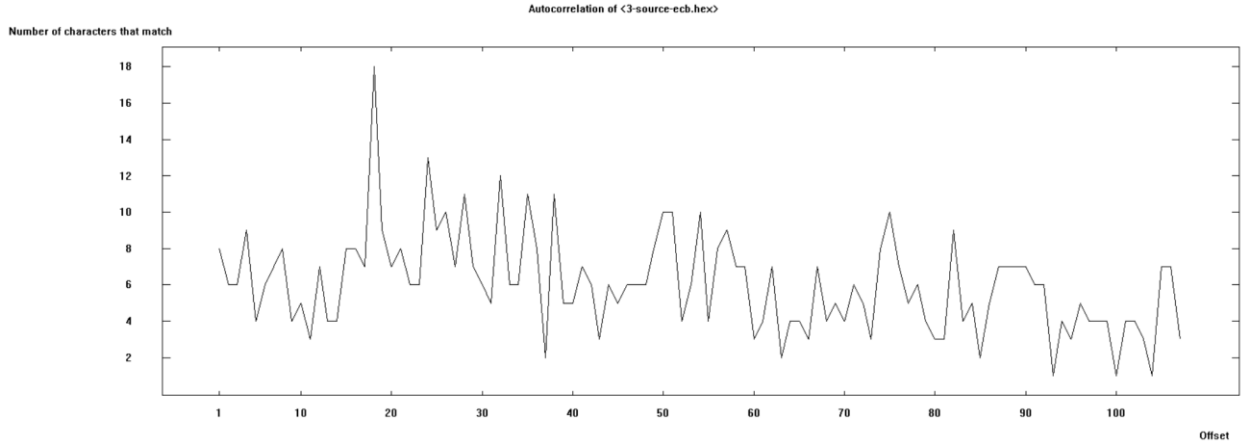

Рис. 3.6 – автокорреляционная характеристика 3DES ECB

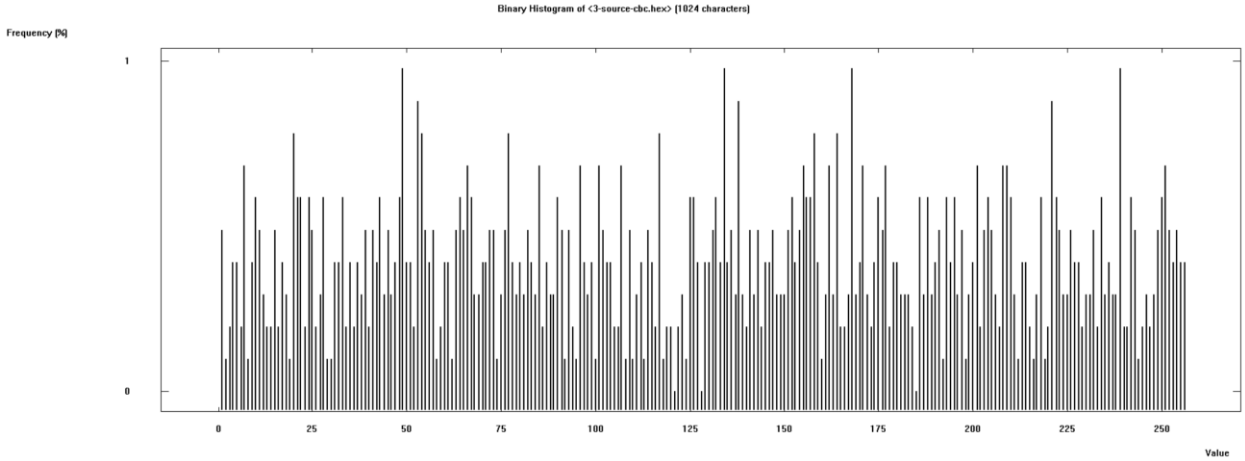

Рис. 3.7 – частотная характеристика 3DES CBC

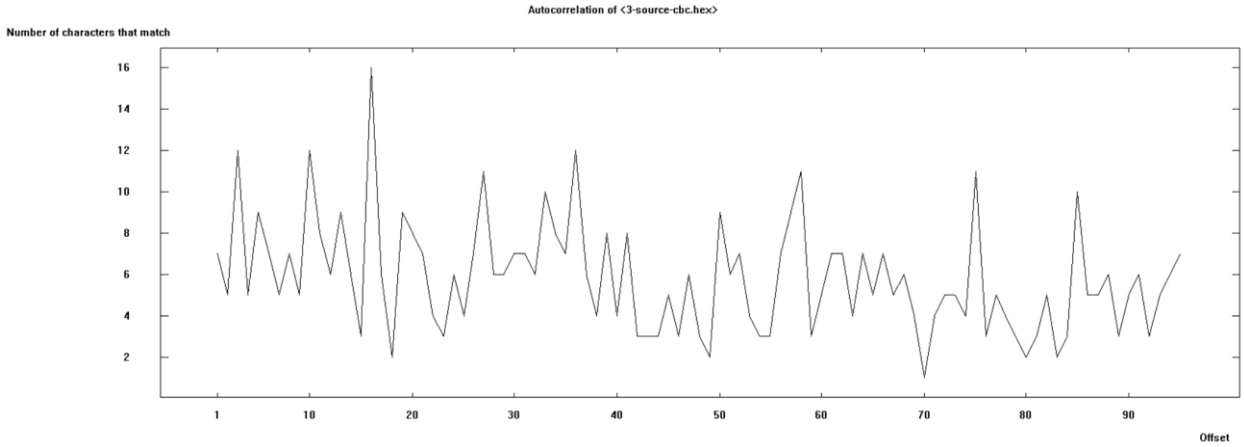

Рис. 3.8 – автокорреляционная характеристика 3DES CBC

3.4. Атака грубой силы

Проанализируем средствами CrypTool 1 зависимость времени атаки грубой силы от размера известной части ключа (табл. 3.1, табл. 3.2).

Табл. 3.1 – атака грубой силы для 3DES ECB

Количество известных байт ключа	Примерное время атаки
2	10^{17} лет
4	$6.3 * 10^{12}$ лет
6	$3.9 * 10^{8}$ лет
8	24000 лет
10	1.5 лет
12	46 минут

Табл. 3.2 – атака грубой силы для 3DES CBC

Количество известных байт ключа	Примерное время атаки
2	$1.2 * 10^{17}$ лет
4	$7.4 * 10^{12}$ лет
6	$4.5 * 10^{8}$ лет
8	27000 лет
10	1.7 лет
12	54 минуты

3.5. Схема реализации в CrypTool 1

Попробуем узнать, какой вариант реализации используется в CrypTool 1. Очевидно, что это либо EEE2, либо EDE2, т.к. размер ключа всего 16 байт. Далее попробуем вручную (т.е. зашифровать-расшифровать-зашифровать с использованием обычного DES) исходный текст (Рис. 3.9).

Видно, что зашифрованные вручную EDE и автоматически тексты сошлись, значит CrypTool 1 использует 3DES EDE2 реализаацию.

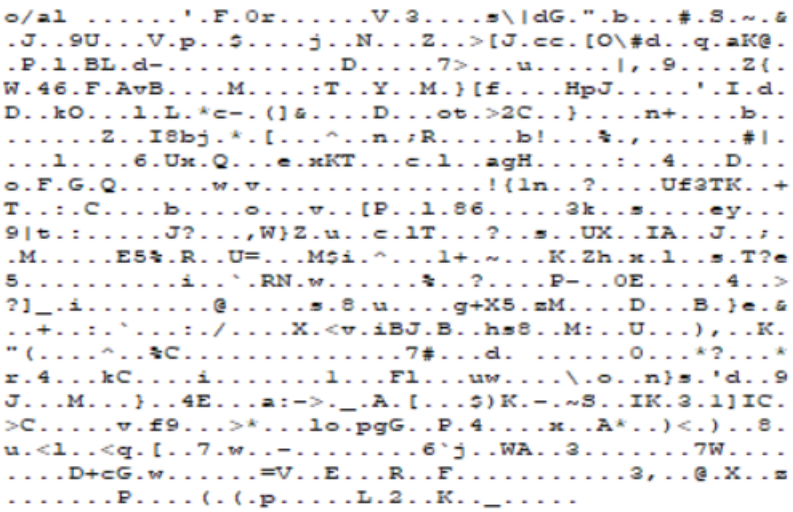

Рис. 3.9 – 3DES EDE2 ECB вручную

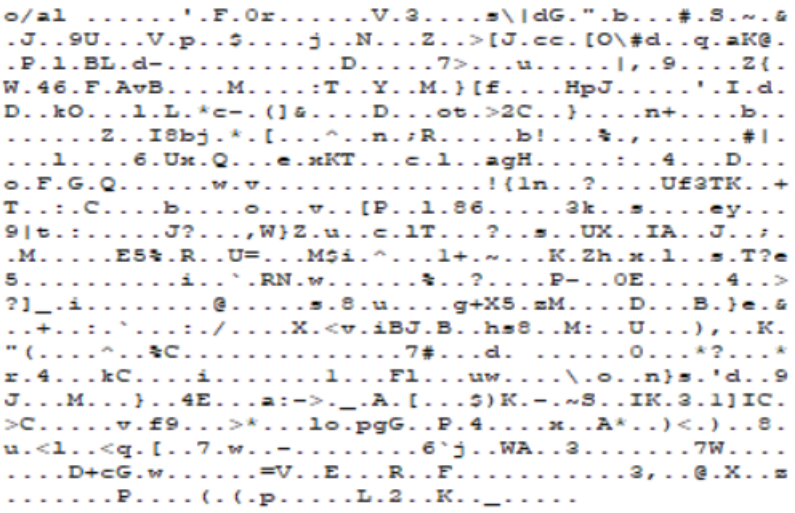

Рис. 3.10 – 3DES ECB автоматически

4. ИССЛЕДОВАНИЕ МОДИФИКАЦИЙ DESX, DESL, DESXL

4.1. Задание

1. Выбрать случайный текст на английском языке (не менее 1000 знаков).

2. Создать бинарный файл с этим текстом, зашифровав и расшифровав его DES на 0-м ключе.

3. С помощью CrypTool зашифровать текст с использованием шифров DESX, DESL, DESXL.

4. Средствами CrypTool вычислить энтропию исходного текста и шифротекстов, полученных в итоге. Зафиксировать результаты измерений в таблице.

5. Средствами CrypTool оцените время проведения атаки «грубой силы» при полном отсутствии информации о секретном ключе.

4.2. Основные параметры и обобщенные схемы шифров

DESX использует на входе ключ длиной 184 бита, который делится на 3 56-битные части. Процесс шифрования происходит по следующей схеме:

$$\mathrm{DESX(M)} = \mathrm{K_2} \oplus \mathrm{DES_K(M} \oplus \mathrm{K_1)}$$

Если $K_1 = K_2 = 0$, то данный алгоритм сводится к стандартному DES.

Алгоритм DESL является облегченной версией алгоритма DES. Данный алгоритм был создан в 2006 году для RFID-меток. Алгоритм предполагает отказ от входной и выходной перестановки блока текста, т.к. они не несут криптографической сложности, а также 8 S-блоков заменяется на 1, но более стойкий чем все 8 стандартных блока DES.

Алгоритм DESXL использует те же оптимизации что и DESL, но производит шифрование по алгоритму DESX.

4.3. Таблица зависимости энтропии шифротекста от используемого шифра.

В качестве исходного текста будем использовать текст из пункта 2. Для DESX и DESXL используется ключ `01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24`, для DESL – `01 02 03 04 05 06 07 08`. Таблица зависимостей энтропии шифротекста от используемого шифра представлена в табл. 4.1.

Табл. 4.1 – зависимость энтропии от шифра

Шифр	Энтропия
DESX	7.83/8.0
DESXL	7.81/8.0
DESL	7.80/8.0

По табл. 1 видно, что энтропии шифров DESX, DESL и DESXL примерно совпадают, что говорит о примерно одинаковой устойчивости шифров.

4.4. Зависимость времени подбора ключа от используемого шифра

Проанализируем время подбора ключа для DESX, DESL и DESXL средствами CrypTool 1 (табл. 4.2 – табл. 4.4).

Табл. 4.2 – время подбора ключа для DESL

Количество известных байт ключа	Примерное время атаки
2	160 дней
4	13 минут

Табл. 4.3 – время подбора ключа для DESX

Количество известных байт ключа	Примерное время атаки
2	$2 * 10^{38}$ лет
4	$1.2 * 10^{34}$ лет
6	$7.4 * 10^{29}$ лет
8	$4.5 * 10^{25}$ лет
10	$6.9 * 10^{20}$ лет
12	$1.1 * 10^{16}$ лет
14	$1.6 * 10^{11}$ лет

16	$2.4 * 10^6$ лет
18	37 лет
20	4 часа

Табл. 4.4 – время подбора ключа для DESXL

Количество известных байт ключа	Примерное время атаки
2	$1.6 * 10^{38}$ лет
4	$9.9 * 10^{33}$ лет
6	$6 * 10^{29}$ лет
8	$3.7 * 10^{25}$ лет
10	$5.6 * 10^{20}$ лет
12	$8.5 * 10^{15}$ лет
14	$1.3 * 10^{11}$ лет
16	$2 * 10^6$ лет
18	30 лет
20	4 часа

ВЫВОДЫ

1. Исследование DES

Был исследован шифр DES. Рассмотрена схема шифрования (рис. 1.1), разобраны ее этапы: исходная перестановка, раунды, конечная перестановка. Отдельно рассмотрен процесс генерации ключей: исходная перестановка, исключающая каждый 8-й бит, деление на субблоки, сдвиг субблоков при генерации каждого следующего раундового ключа и объединение субблоков, из которых в результате сжимающей перестановки получаются раундовые ключи. Описана также схема реализации раундовой функции: расширяющая перестановка, сложение с раундовым ключем, замена получившихся 8 субблоков по таблице замен, завершающая перестановка.

Была разработана программа, осуществляющая шифрование и расшифрование блока текста по описанному выше алгоритму DES (приложение А), проведено ее тестирование на примере исходного текста *mironchi* и ключа *030208d*. В результате шифрования был получен фрагмент с кодами символов `31 189 74 9 3 254 201 8`, который затем был успешно расшифрован с применением той же программы. Процесс дешифрования был описан теоретически, но сами этапы преобразований расписаны не были, поскольку они полностью дублировали этапы преобразований для шифрования, за исключением реализации раундов (также потому, что получить полную информацию можно запуском программы). Процесс зашифрования описан максимально подробно (п. 1.3).

2. Режимы ECB и CBC шифра DES

Рассмотрены основные параметры режимов ECB и CBC.

ECB – шифрование в режиме DES по блокам, при этом следующий зашифрованный блок не зависит от предыдущего. ECB позволяет проводить шифрование блоков быстрее за счет параллельной обработки, а также благодаря независимости блоков обеспечивается устойчивость к ошибкам

(неправильное зашифрование одного блока не приведет к порче остальных блоков). Тем не менее, при зашифровании в режиме ECB одинаковые блоки будут одинаковыми и в зашифрованном виде, а также появляется уязвимость в виде возможности замены любого блока без повреждения других.

CBC – шифрование, в котором для каждого следующего блока выполняется сложение с зашифрованным предыдущим блоком. В отличие от ECB, ошибка в одном блоке распространяется на все следующие, однако при дешифровке происходит самовосстановление; одинаковые блоки будут в результате преобразованы в разные. Последний блок можно использовать в качестве контроля целостности сообщения. Тем не менее, CBC не позволяет провозить параллельное шифрование различных блоков.

На рис. 2.4 и рис. 2.5 наглядно показаны различия в шифровании одинаковых блоков в режимах ECB и CBC. CBC дает гораздо больше шума в сообщении, однако зашифрованное этим режимом сообщение практически невозможно сжать.

Для ECB и CBC было проведено исследование зависимости времени атаки грубой силы от известной длины ключа. Выяснено, что при известных четырех байтах подбор занимает около получаса, для двух известных байт – уже около года. CBC показал при этом несколько лучшие результаты с точки зрения устойчивости.

3. 3-DES

3-DES – шифр DES, примененный трижды. Существуют различные модификации шифра 3DES, отличающиеся длиной ключа и средней операцией – шифрование или дешифрование.

Очевидно, что шифр 3-DES более устойчив к атаке грубой силы в сравнении с DES, т.к. имеет большую длину ключа – время подбора ключа атакой грубой силы показано в табл. 3.1 и табл. 3.2. Режим CBC опять показал несколько лучшие результаты в сравнении с режимом ECB.

Выяснено, что CrypTool 1 использует EDE2 реализацию шифра (см. п. 3.5).

4. DESX, DESL, DESXL

В заключение были рассмотрены шифры DESX, DESL и DESXL. Выяснено, что DESX представляет собой модифицированный вариант DES с 184-битным ключем, формула шифрования которого выглядит как

$$\mathrm{DESX(M)} = \mathrm{K_2} \oplus \mathrm{DES_K(M} \oplus \mathrm{K_1)}$$

DESL – облегченный вариант DES без начальной и конечной перестановок, а DESXL – комбинация DESX и DESL шифров.

Установлено, что все три шифра обладают одинаковой энтропией. Также для шифров проведен анализ атаки грубой силы, который показал, что DESX и DESXL примерно одинаково устойчивы к атакам грубой силы, а шифр DESL требует примерно вдвое меньше времени для подбора ключа, чем шифр DES. Вероятно, это связано с тем, что в шифровании DESL меньше операций, т.к. удаление операций начальной и конечной перестановок не влияют на криптостойкость шифра.

ПРИЛОЖЕНИЕ А

Программа для шифрования/дешифрования DES

```
import numpy as np

import warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)


IP = np.fromfile('tables/ip', sep=' ', dtype=np.int64) - 1
IP_1 = np.fromfile('tables/ip_1', sep=' ', dtype=np.int64) - 1
KEY_E = np.fromfile('tables/key_e', sep=' ', dtype=np.int64) - 1
KEY_CP = np.fromfile('tables/key_cp', sep=' ', dtype=np.int64) -
1
KEY_N = np.fromfile('tables/key_n', sep=' ', dtype=np.int64)
F_EP = np.fromfile('tables/f_ep', sep=' ', dtype=np.int64) - 1
F_S = np.fromfile('tables/f_s', sep=' ',
dtype=np.int64).reshape([4 * 8, 16])
F_P = np.fromfile('tables/f_p', sep=' ', dtype=np.int64) - 1


def num_to_bits(_num, _padding):
    return np.fromstring(np.binary_repr(_num).zfill(_padding),
dtype='S1').astype(int)


def bits_to_symbols(_bits):
    _bits = _bits.reshape([-1])
    assert(len(_bits) % 8 == 0)
    _i = 0
    _symbols = ""
    while _i < len(_bits):
        _num = 0
        for _ind in range(8):
            _num <<= 1
            _num |= _bits[_i]
            _i += 1
        _symbols += chr(_num)
    return _symbols


def symbols_to_bits(_symbols):
    _bits = []
    for _c in _symbols:
        _code = ord(_c)
        _code_bits = []
        for i in range(8):
            _code_bits.insert(0, _code & 1)
            _code >>= 1
        _bits += _code_bits
    return np.array(_bits, dtype=np.int64)
```

```
def generate_keys(_key, _rounds):
    print("Generating keys")
    _bits = symbols_to_bits(_key).reshape([8, 7])
    _bits = np.append(_bits, np.zeros([8,1], dtype=np.int64),
axis=1)
    print("Key bits\n", _bits)
    _bits = np.take(_bits.reshape([-1]), KEY_E).reshape(8, 7)
    print("Applying E\n", _bits)

    _C = _bits[:len(_bits)//2]
    _D = _bits[len(_bits)//2:]
    print("C\n", _C)
    print("D\n", _D)

    _keys = []
    for _round in range(_rounds):
        _C = np.roll(_C, KEY_N[_round])
        _D = np.roll(_D, KEY_N[_round])
        _round_key = np.append(_C, _D)
        _round_key = np.take(_round_key, KEY_CP)
        _keys.append(_round_key)
        print("Round key {0}\n".format(_round + 1),
_round_key.reshape(8, 6))

    return _keys


def f(_b, _key):
    _bits = np.take(_b.reshape([-1]), F_EP)
    _key = _key.reshape([-1])
    _bits = np.logical_xor(_bits, _key) * 1
    _bits = _bits.reshape([8, 6])
    _out_bits = np.array([], dtype=np.int64)

    for _i in range(8):
        _s = F_S[_i * 4:(_i + 1) * 4]
        _row = _bits[_i][0] * 2 + _bits[_i][5]
        _column = _bits[_i][1] * 8 + _bits[_i][2] * 4 +
_bits[_i][3] * 2 + _bits[_i][4]
        _out_bits = np.append(_out_bits,
num_to_bits(_s[_row][_column], 4))

    _bits = _out_bits
    _bits = np.take(_bits, F_P)

    return _bits


def encode(_symbols, _key, _rounds):
    _bits = symbols_to_bits(_symbols).reshape([-1])
    assert(len(_bits) == 64)
```

```
    print("Encoding {0}".format(_symbols))
    _keys = generate_keys(_key, _rounds)
    _bits = np.take(_bits, IP)
    _A = _bits[:len(_bits)//2]
    _B = _bits[len(_bits)//2:]

    for _round in range(_rounds):
        print("Round {0}".format(_round + 1))
        _A, _B = _B, np.logical_xor(_A, f(_B, _keys[_round])) *
1

    _bits = np.append(_A, _B)
    _bits = np.take(_bits, IP_1)

    return bits_to_symbols(_bits)


def decode(_symbols, _key, _rounds):
    _bits = symbols_to_bits(_symbols).reshape([-1])
    assert(len(_bits) == 64)
    print("Decoding {0}".format(_symbols))
    _keys = generate_keys(_key, _rounds)
    _bits = np.take(_bits, IP)
    _A = _bits[:len(_bits)//2]
    _B = _bits[len(_bits)//2:]

    for _round in reversed(range(_rounds)):
        print("Round {0}".format(_round + 1))
        _A, _B = np.logical_xor(_B, f(_A, _keys[_round])) * 1,
_A

    _bits = np.append(_A, _B)
    _bits = np.take(_bits, IP_1)

    return bits_to_symbols(_bits)


key = "838208d"
rounds = 2

encoded = encode("mironchi", key, rounds)
print([ord(c) for c in encoded], encoded)

print("-----------------------------------")

decoded = decode(encoded, key, rounds)
print([ord(c) for c in decoded], decoded)
```