Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Факультет безопасности информационных технологий

Дисциплина:

«Криптографические методы обеспечения информационной безопасности»

ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ №3

«Основные структурные элементы алгоритма AES»

Выполнил:

Полевцов Артем Сергеевич, студент группы N34511

(подпись)

Проверил:

Волков Александр Григорьевич, инженер ФБИТ

(отметка о выполнении)

(подпись)

Санкт-Петербург
2023

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель работы - изучить основные принципы работы алгоритмы AES.

В задачи данной лабораторной работы:

— Проанализировать эмуляцию алгоритма AES и примитивных атак на шифр, используя Cryptool 2. Выделить основные необходимые настройки шифра и требуемые ограничения на параметры;

— Выполненить 1 цикла раундовой функции алгоритма AES вручную. Отразить промежуточные результаты шифрования после всех этапов алгоритма AES. Дать математическое обоснование для каждой операции. Также для подробного изучения шифра может быть использована программная реализация 1 раунда (или полной системы) AES в режиме отладки с выводом промежуточных значений шифрования;

— Проанализировать принципы использования криптосистемы в современных приложениях на примере библиотеки openssl;

1 ОСНОВНЫЕ СТРУКТУРНЫЕ ЭЛЕМЕНТЫ АЛГОРИТМА AES

1.1 Ход работы

1.1.1 Изучение алгоритма шифрования AES при помощи AES Visualization в Cryptool

Я оставил ключ и открытый текст по умолчанию:

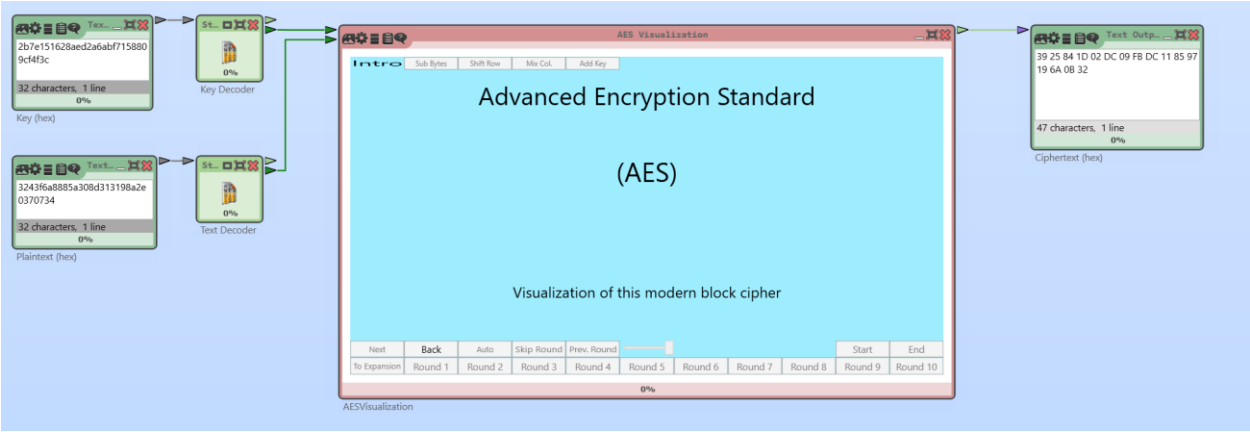

Рисунок 1 - AES Visualization в Cryptool 2

Расширение ключа:

Из исходных байтов ключа была построена матрица 4 на 4

Рисунок 2 – State matrix

Здесь мы взяли последний столбец матрицы и сдвинули все байты в этом столбце на один вверх так, чтобы последний стал первым, предпоследний – последним и так далее.

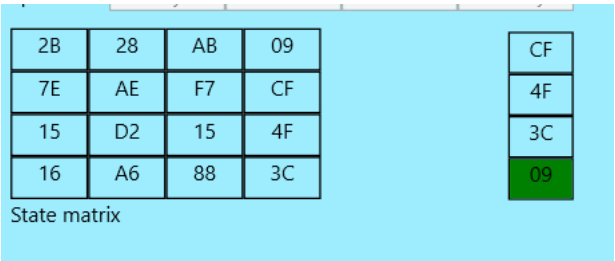

Рисунок 3 – State matrix

Далее по матрице s-box, в которой название столбца – это левая часть байта, а строки – правая.

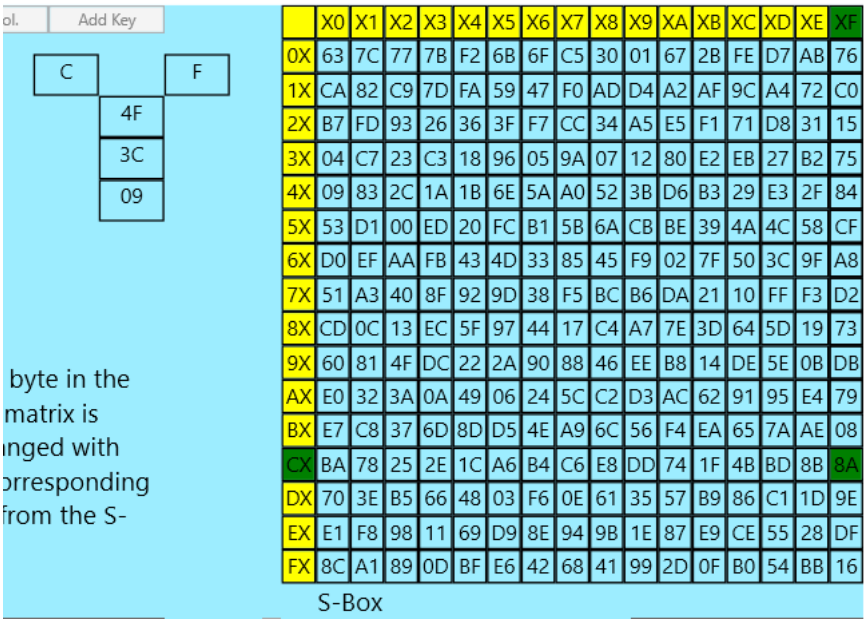

Рисунок 4 – Преобразование столбца матрицы

Таким образом, получаем новый столбец из байтов исходного:

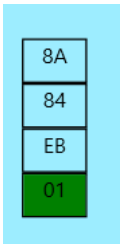

Рисунок 5 – Новый столбец

Далее к полученному столбцу применяется операция сложения с константой, которая соответствует номеру этапа расширения ключа.

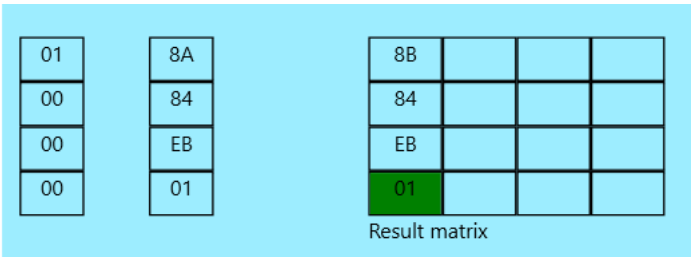

Рисунок 6 – Формирование результирующей матрицы

Далее мы берем первый столбец матрицы и применяем операцию xor с получившимся в предыдущем шаге столбцом.

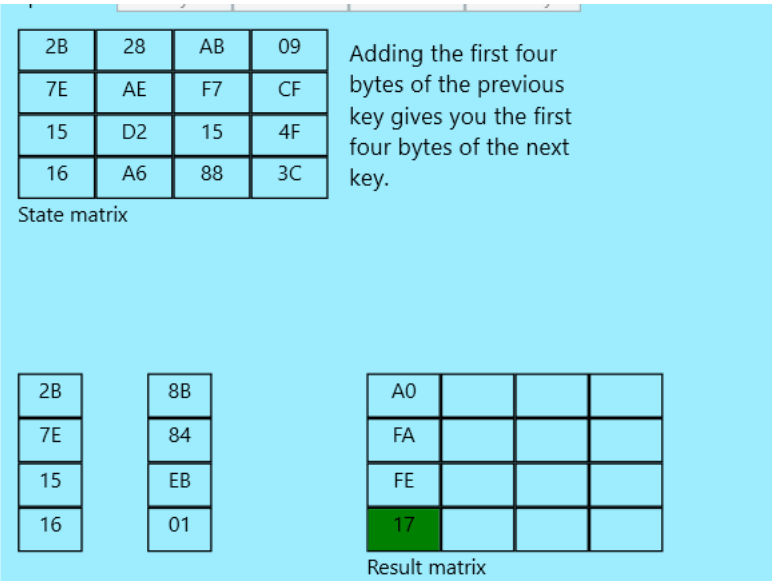

Рисунок 7 – Формирование результирующей матрицы

Далее применяем операцию xor для следующего столбца исходной матрицы и крайнего полученного столбца результирующей матрицы.

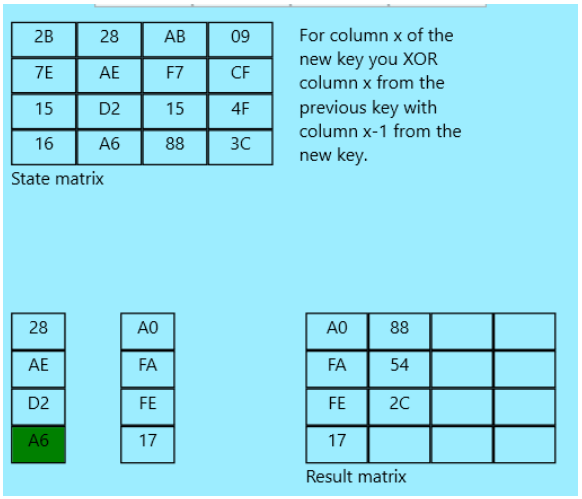

Рисунок 8 – Формирование результирующей матрицы

И так повторяем для всех столбцов, в итоге получаем новую матрицу 4 на 4:

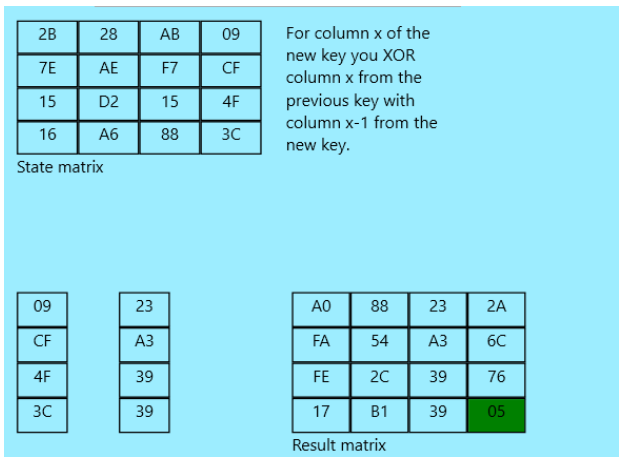

Рисунок 9 – Формирование результирующей матрицы

На следующем этапе проделываем те же самые операции, только полученная на предыдущем этапе матрица становится исходной:

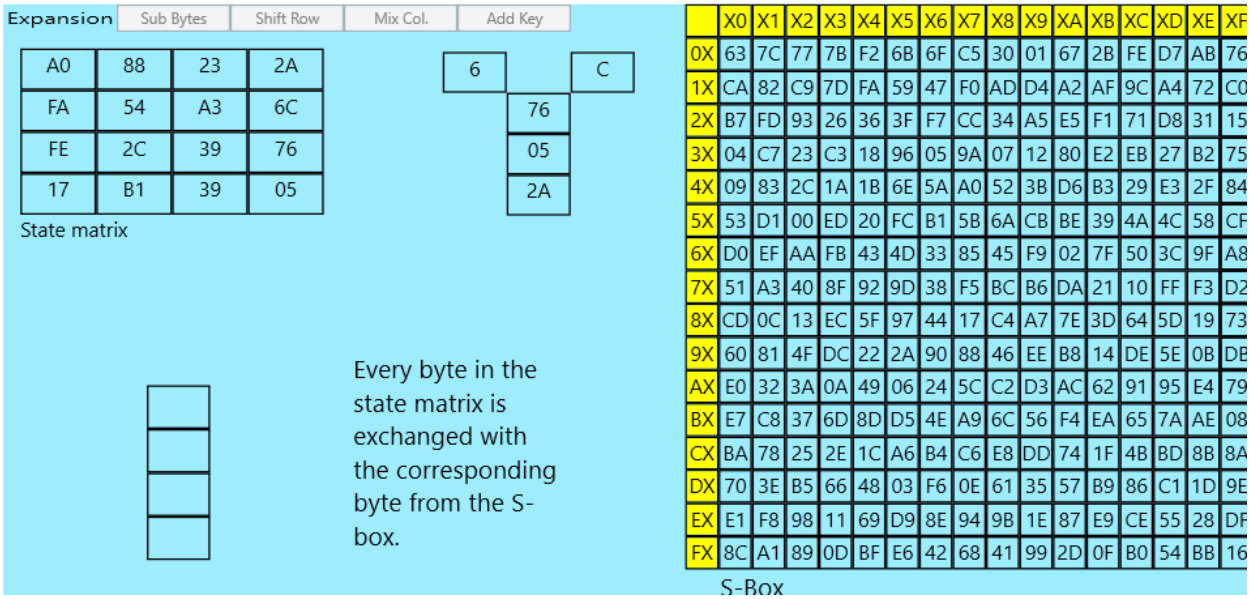

Рисунок 10 – Начало второго раунда

На первом этапе шифрования мы построчно слева-направо сверху-вниз применяем операцию xor для каждого байта исходного сообщения и ключа.

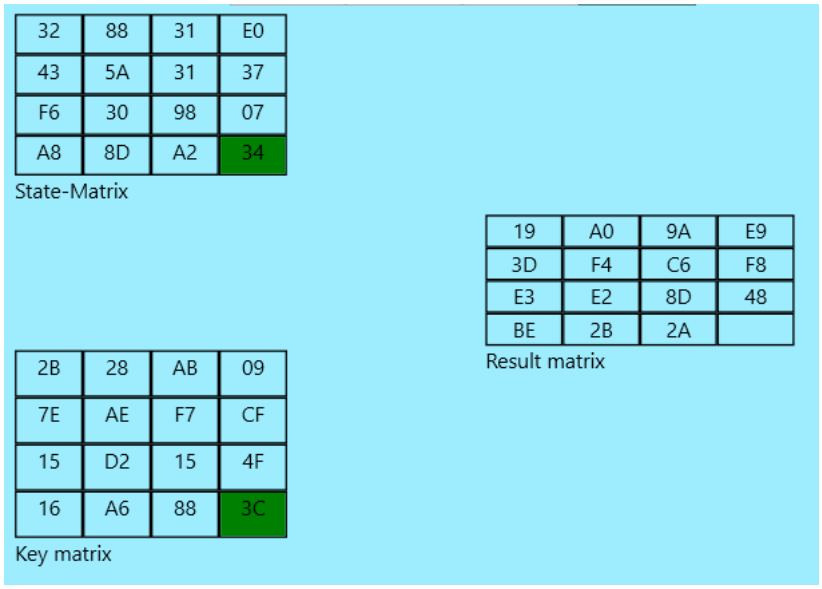

Рисунок 11 – Операция xor

Далее на этапе Sub Bytes мы так же идем слева-направо и сверху-вниз по строкам получившейся на предыдущем шаге матрицы и заменяем каждый байт по матрице S-Box

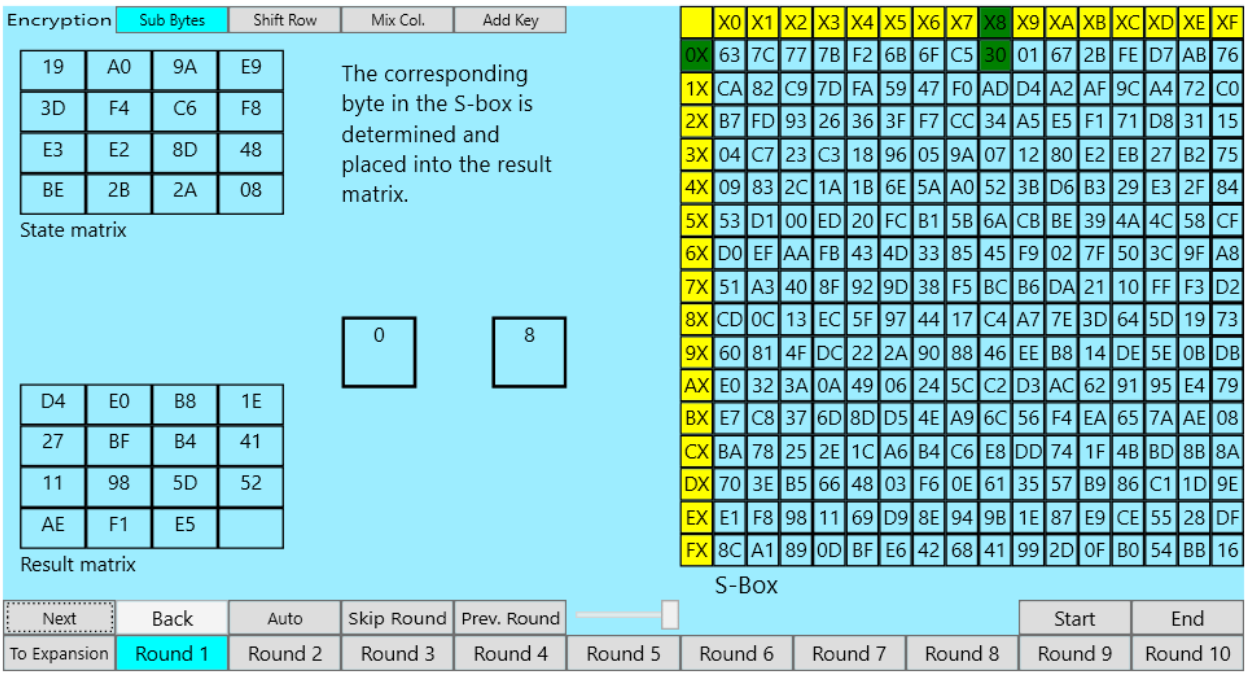

Рисунок 12 – Этап Sub Bytes

На этапе Shift Row сначала вторая строка сдвигается один раз влево. Затем третья строка дважды сдвигается влево и, наконец, четвертая строка сдвигается три раза влево. Перекрывающиеся байты переносятся вправо, образуя матрицу 4 x 4.

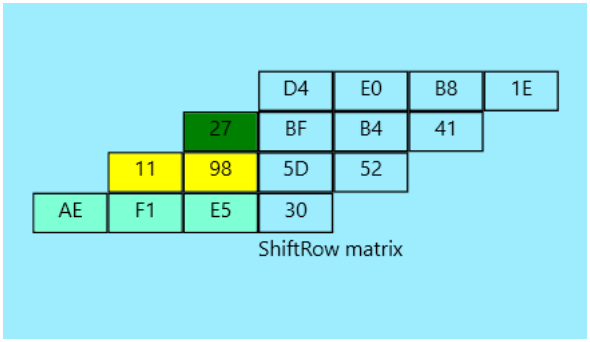

Рисунок 13 - Этап Shift Row

Рисунок 14 - Этап Shift Row

На этапе Mix Col мы выполняем умножение каждого столбца, полученной на предыдущем этапе матрицы на специальную матрицу:

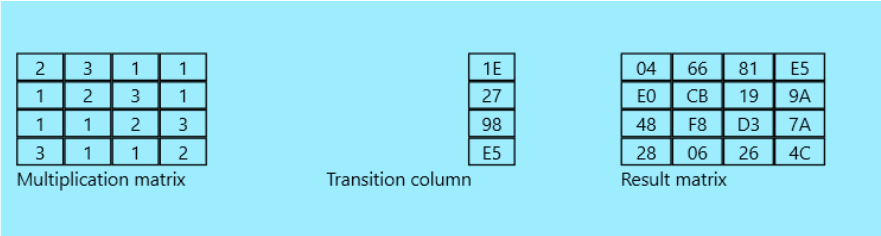

Рисунок 15 - Этап Mix Col

Далее на этапе Add key мы меняем в полученной на прошлом этапе матрице столбцы и строки местами и производим операцию xor для каждого байта этой матрицы с каждым байтом ключа:

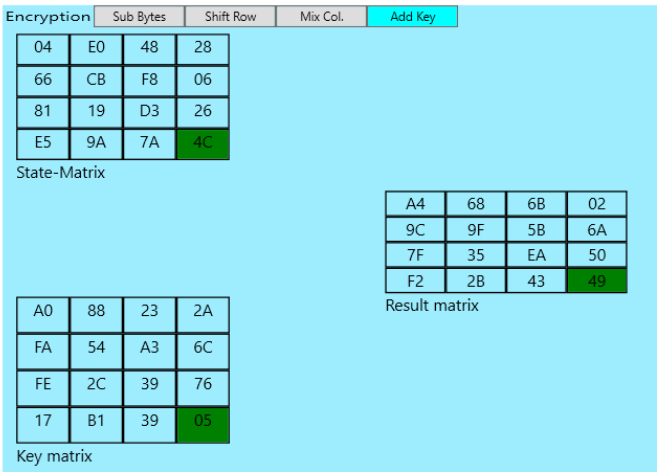

Рисунок 16 - Этап Add Key

1.1.2 Демонстрация лавинного эффекта:

Для демонстрации лавинного эффекта я изменил случайный бит входного сообщения:

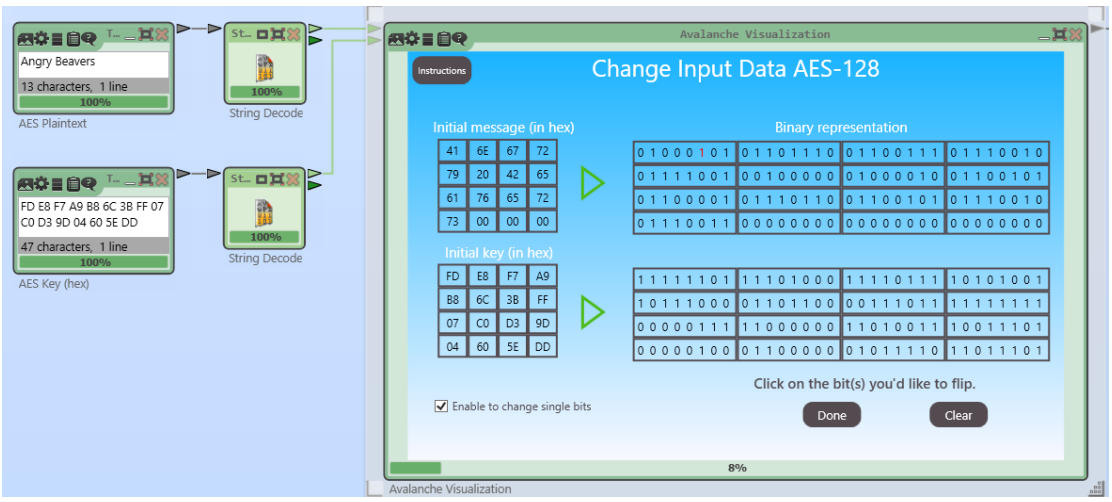

Рисунок 17 – Демонстрация лавинного эффекта

Далее, пройдя десять раундов, мы можем видеть, что на третьем раунде произошел скачок с 7% искаженных байтов до 49,2% и далее это значение остается примерно одинаковым, максимальный процент искаженных бит мы можем видеть на восьмом раунде 57%.

Можем сделать выводы, что такое значение является очень хорошим так как сильно искажает большое количество конечных данных при минимальном количестве изменений во входных.

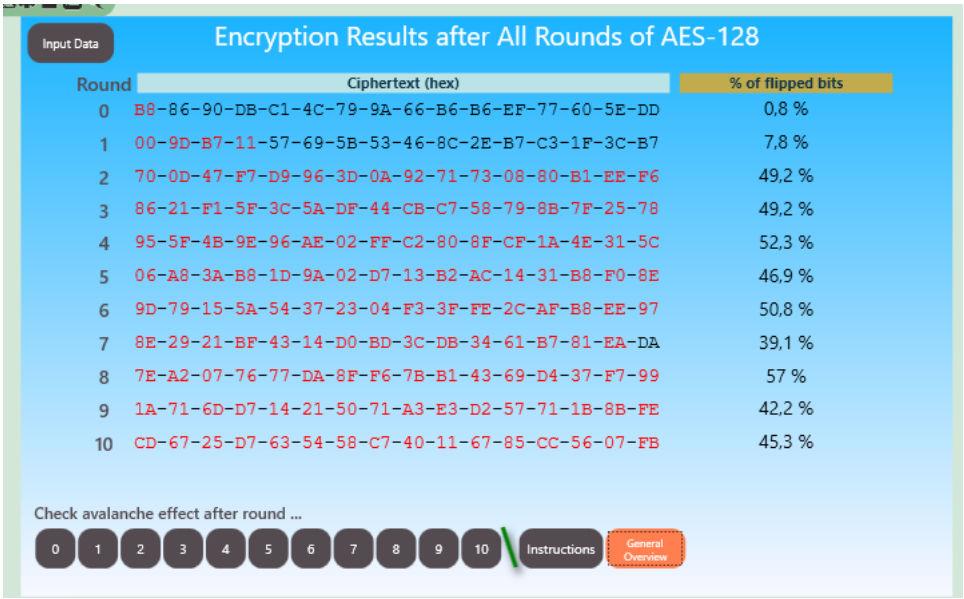

Рисунок 18 – Результат лавинного эффекта

1.1.3 Проведение атаки на основе известного открытого текста

Криптоанализ алгоритма AES с использованием компонента KeySearcher и известного фрагмента открытого текста (шпаргалки). KeySearcher получает на вход зашифрованный текст и пробует все ключи данного пространства ключей. Полученные открытые тексты затем сопоставляются с шпаргалкой. На выход отправляется открытый текст, который лучше всего соответствует шпаргалке.

Укажем в настройках анализа 13 байт ключа для более быстрого получения результата из 16 байт.

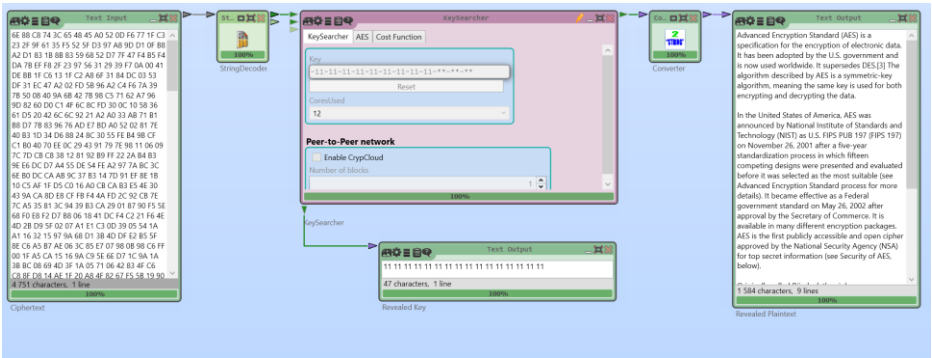

Рисунок 19 – Алгоритм KeySearcher

На 12 ядрах процесс подбора ключа занял около двух минут.

1.1.4 Проведение дифференциального криптоанализа

Если операция XOR используется для шифрования сообщения m с ключом k, зашифрованный текст c получается следующим образом:

$$m \oplus k = c$$

Рисунок 20 – Операция XOR

Основная идея дифференциального криптоанализа заключается в использовании эффекта свойства операции XOR путем двукратного использования ключа k. Если добавление ключа выполняется дважды, применяется следующее:

$$((m \oplus k) \oplus k) = c \oplus k = m$$

Рисунок 21 – Эффект свойства операции XOR

Мы получаем обратно исходное сообщение m.

Оператор XOR часто называют оператором разности. Это имя также используется в этом уроке.

Дифференциальный криптоанализ — это так называемая «атака с выбранным открытым текстом», которая означает, что криптоаналитик может выбирать сообщения с открытым текстом и шифровать их, чтобы иметь совпадающие пары открытый текст/зашифрованный текст.

Мы воспользуемся идеей использования ключа дважды, используя пары сообщений m1 и m2. Если теперь разница между зашифрованными текстами образовалась, справедливо следующее:

$$\begin{aligned} m_1 \oplus k &= c_1 \\ m_2 \oplus k &= c_2 \end{aligned}$$

Рисунок 22 – Разница между зашифрованными текстами

Следует:

$$c_1 \oplus c_2 = (m_1 \oplus k) \oplus (m_2 \oplus k) = m_1 \oplus m_2$$

Рисунок 23 – Операция XOR

Мы получаем разницу открытых текстов, вычисляя разницу зашифрованных текстов. Это можно использовать для получения информации о раундовых ключах в DCA.

Здесь мы применяем идею DCA к Chiffre 1. Структура шифра обычно общедоступна, поэтому известны все компоненты и их функциональность. Безопасность шифра никогда не должна основываться на секретности конструкции, а на секретности ключа. Это требование еще называют принципом Керкгофа.

Здесь мы применяем идею DCA к Chiffre 1. Структура шифра обычно общедоступна, поэтому показаны все компоненты и их функциональность. Безопасность шифра никогда не должна основываться на секретности конструкции, а также на секретности переключателя. Это требование по-прежнему называют принципом Керкгофа.

$$m \oplus k_0 = u$$

Рисунок 24 – Принцип Керкгофа

Промежуточный результат u затем заменяется S-блоками:

$$S(u) = v$$

Рисунок 25 – Замена S-блоками

Наконец, снова происходит добавление ключей (промежуточный результат v указан с добавлением k1):

$$v \oplus k_1 = c$$

Рисунок 26 – Добавление ключей

Внутренние переменные u и v шифра имеют большое значение в дальнейшем, поскольку мы используем эти промежуточные результаты для восстановления неизвестных ключей k1 и k0.

На следующем рисунке обобщен процесс шифрования, описанный выше на слайдах 11–16.

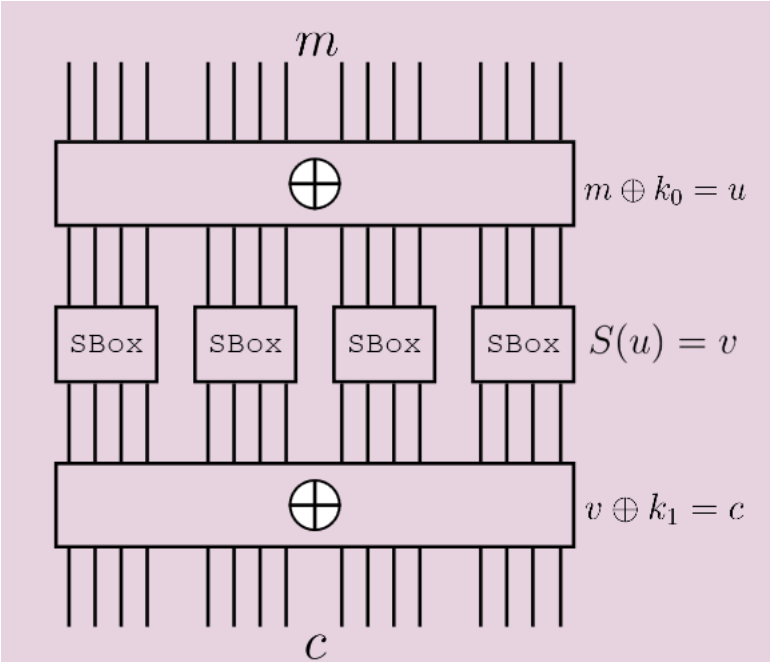

Рисунок 27 – Обобщенный процесс шифрования

Криптоаналитику известны параметры m и c, но не внутренние промежуточные значения u и v, поскольку k0 и k1 неизвестны. Однако криптоаналитик может вычислить разницу между двумя внутренними промежуточными значениями u на основе разницы между двумя открытыми текстами. Далее мы рассмотрим два сообщения m1 и m2 и построим их разницу:

$$m_1 \oplus m_2 = (m_1 \oplus k_0) \oplus (m_2 \oplus k_0) = u_1 \oplus u_2$$

Рисунок 28 – Разница двух сообщений

Эти знания можно использовать для расчета k1. Для этого мы рассмотрим две пары открытого текста и зашифрованного текста (m1, c1) и (m2, c2). Согласно уравнению (1), разность значений u равна разнице m1 и m2. Далее «советуется» k1 так, чтобы из

$$v \oplus k_1 = c$$

Рисунок 29 – Расчет c

Поскольку функциональность S-Box является общедоступной и, следовательно, обратимой, криптоаналитик может выполнять вычисления с помощью

$$S(u) = v$$

Рисунок 30 – Вычисления

$$S^{-1}(v_1) \text{ und } S^{-1}(v_2)$$

Рисунок 31 – Значения

Однако эти значения нельзя напрямую сравнивать с внутренними значениями u1 и u2, поскольку они неизвестны.

Из уравнения (1) следует, что если ключ раунда k1 угадан правильно, справедливо следующее:

$$u_1 \oplus u_2 = S^{-1}(v_1) \oplus S^{-1}(v_2)$$

Рисунок 32 – Уравнение

$$u_1 \oplus u_2$$

Рисунок 33 – Операция XOR

уже известно. Как указано на предыдущей странице, значения k1 угаданы. Здесь объясняется, как ограничить набор возможных правильных значений: Криптоаналитик пробует все значения t из k1, и если

$$S^{-1}(t \oplus c_1) \oplus S^{-1}(t \oplus c_2) = u_1 \oplus u_2$$

Рисунок 34 – Ограничение набора возможных правильных значений

верно, t считается кандидатом на k1. Если в конце остается более одного кандидата, атака повторяется с новыми парами открытого текста и зашифрованного текста.

После успешного восстановления k1 можно сразу вычислить k0. Это можно сделать, рассчитав уравнение

$$k_0 = S^{-1}(c \oplus k_1)$$

Рисунок 35 – Вычисление k0

Будет определено.

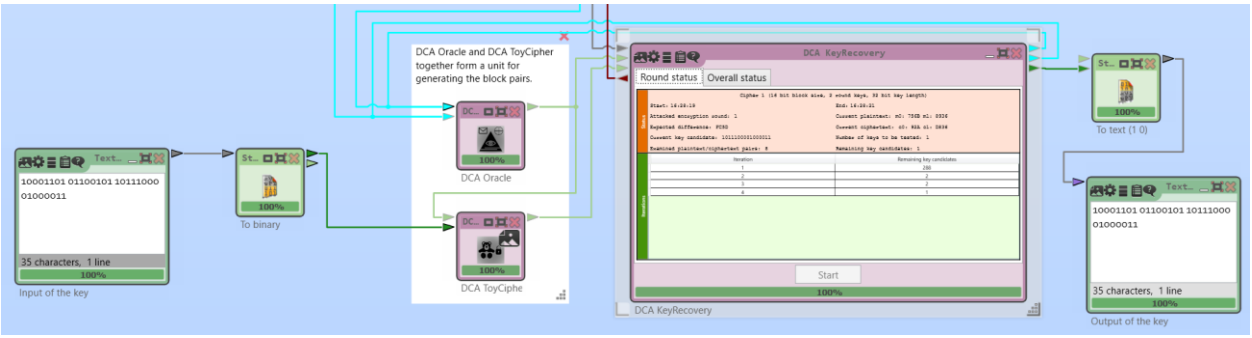

Рисунок 36 – Результат криптоанализа

1.1.5 Ручное выполнение 1 цикла раундовой функции

Входное сообщение: 81 e4 2a f6 a4 0b 5b 05 f2 cf 0a 96 f2 e2 33 e0

Ключ: df 9b 26 79 74 05 60 62 1d 9e 2d 39 05 a4 67 59

81	a4	f2	f2
e4	0b	cf	e2
2a	5b	0a	33
f6	05	96	e0

Рисунок 37 - Входное сообщение

DF	74	1d	05
9B	5	9e	a4
26	60	2d	67
79	62	39	59

Рисунок 38 - Ключ

Расширение ключа:

Берем последний столбец исходной матрицы, сдвигаем каждый байт на один вверх и заменяем через матрицу S-box:

05		a4		49
a4		67		85
67	>	59	>	CB
59		05		6B

Рисунок 40 – Промежуточные вычисления

Далее производим операцию xor с предыдущим столбцом, в данном случае с константой:

01		49		48
00	XOR	85	=	85
00		CB		CB
00		6B		6B

Рисунок 41 – Промежуточные вычисления

Далее берем первый столбец исходной матрицы и производим xor его с получившимся на прошлом шаге:

DF		48		97
9B	XOR	85	=	1e
26		CB		ed
79		6B		12

Рисунок 42 – Промежуточные вычисления

Проводим аналогичные действия, только со вторым столбцом исходной матрицы и крайним непустым столбцом результирующей матрицы:

74		97		e3
5	XOR	1e	=	1b
60		ed		8d
62		12		70

Рисунок 43 – Промежуточные вычисления

И так со всеми оставшимися столбцами исходной матрицы:

1d		e3		fe
9e	XOR	1b	=	85
2d		8d		a0
39		70		49

Рисунок 44 – Промежуточные вычисления

05		fe		fb
a4	XOR	85	=	21
67		a0		c7
59		49		10

Рисунок 45 – Промежуточные вычисления

В итоге получаем результирующую матрицу:

97	e3	fe	fb
1e	1b	85	21
ed	8d	a0	c7
12	70	49	10

Рисунок 46– Промежуточные вычисления

Побитовое сложение с ключом:

Применим операцию xor для каждого байта открытого текста и каждого байта ключа:

81	a4	f2	f2		df	74	1d	05		5e	d0	ef	f7
e4	0b	cf	e2	XOR	9b	05	9e	a4	=	7f	0e	51	46
2a	5b	0a	33		26	60	2d	67		0c	3b	27	54
f6	05	96	e0		79	62	39	59		8f	67	af	B9

Рисунок 47 – Промежуточные вычисления

Операция SubBytes:

В результате операции получаем матрицу:

58	70	df	68
d2	ab	d1	5a
fe	e2	cc	20
73	85	79	56

Рисунок 48 – Промежуточные вычисления

Операция ShiftRows:

Первая строка остается на месте, вторую сдвигаем на один элемент, третью -на два, четвертую на три:

			58	70	df	68
		d2	ab	d1	5a	
	fe	e2	cc	20		
73	85	79	56			

Рисунок 49 – Промежуточные вычисления

Получаем:

58	70	df	68
ab	d1	5a	d2
cc	20	fe	e2
56	73	85	79

Рисунок 50 – Промежуточные вычисления

Операция MixColumns:

b1		02	03	01	01		a1
b2	=	01	02	03	01	*	a2
b3		01	01	02	03		a3
b4		03	01	01	02		a4

Рисунок 51 – Промежуточные вычисления

Произведем операцию для каждой строки:

2	3	1	1		58		cc
1	2	3	1		ab		db
1	1	2	3		cc		30
3	1	1	2		56		26

Рисунок 52 – Промежуточные вычисления

2	3	1	1		70		0c
1	2	3	1		d1		da
1	1	2	3		20		f7
3	1	1	2		73		93

Рисунок 53 – Промежуточные вычисления

2	3	1	1		df		8a
1	2	3	1		5a		74
1	1	2	3		fe		f6
3	1	1	2		85		ee

Рисунок 54 – Промежуточные вычисления

2	3	1	1		68	23
1	2	3	1		d2	87
1	1	2	3		e2	cf
3	1	1	2		79	7a

Рисунок 55 – Промежуточные вычисления

В итоге получаем следующую последовательность байт:

cc	0c	8a	23
db	da	74	87
30	f7	f6	cf
26	93	ee	7a

Рисунок 56 – Промежуточные вычисления

1.1.6 Шифрование файла с помощью библиотеки OpenSSL

```
┌──(root㉿DESKTOP-JNTUATF)-[~]
└─# openssl enc -list | grep aes
-aes-128-cbc               -aes-128-cfb               -aes-128-cfb1
-aes-128-cfb8              -aes-128-ctr               -aes-128-ecb
-aes-128-ofb               -aes-192-cbc               -aes-192-cfb
-aes-192-cfb1              -aes-192-cfb8              -aes-192-ctr
-aes-192-ecb               -aes-192-ofb               -aes-256-cbc
-aes-256-cfb               -aes-256-cfb1              -aes-256-cfb8
-aes-256-ctr               -aes-256-ecb               -aes-256-ofb
-aes128                    -aes128-wrap               -aes192
-aes192-wrap               -aes256                    -aes256-wrap
-id-aes128-wrap            -id-aes128-wrap-pad        -id-aes192-wrap
-id-aes192-wrap-pad        -id-aes256-wrap            -id-aes256-wrap-pad
```

Рисунок 57 – Список всех алгоритмов, включающих aes

-aes-128-ofb:

```
┌──(root㉿DESKTOP-JNTUATF)-[~/lab3]
└─# openssl enc -aes-128-ofb -e -in plaintext.txt -out aes128ofb.enc
enter AES-128-OFB encryption password:
Verifying - enter AES-128-OFB encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.

┌──(root㉿DESKTOP-JNTUATF)-[~/lab3]
└─# cat aes128ofb.enc
Salted__=◆^◆}◆I◆Wd◆◆◆◆"K◆-◆
```

Рисунок 58 – AES-128-ofb

```
┌──(root㉿DESKTOP-JNTUATF)-[~/lab3]
└─# openssl enc -aes-128-ofb -d -in aes128ofb.enc -out aes128ofbdecrypted.txt
enter AES-128-OFB decryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.

┌──(root㉿DESKTOP-JNTUATF)-[~/lab3]
└─# cat aes128ofbdecrypted.txt
angry beavers
```

Рисунок 59 – AES-128-ofb

-aes-256-ecb:

```
┌──(root㉿DESKTOP-JNTUATF)-[~/lab3]
└─# openssl enc -aes-256-ecb -e -in plaintext.txt -out aes256ecb.enc -salt
enter AES-256-ECB encryption password:
Verifying - enter AES-256-ECB encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.

┌──(root㉿DESKTOP-JNTUATF)-[~/lab3]
└─# cat aes256ecb.enc
Salted__QBs*8   7␣y#␣e/'w◆◆◆r◆-
```

Рисунок 60 – AES-256-ecb

```
┌──(root㉿DESKTOP-JNTUATF)-[~/lab3]
└─# openssl enc -aes-256-ecb -d -in aes256ecb.enc -out aes256ecbdecrypted.txt
enter AES-256-ECB decryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.

┌──(root㉿DESKTOP-JNTUATF)-[~/lab3]
└─# cat aes256ecbdecrypted.txt
angry beavers
```

Рисунок 61 – AES-256-ecb

-aes-192-cfb:

```
┌──(root㉿DESKTOP-JNTUATF)-[~/lab3]
└─# openssl enc -aes-192-cfb -e -in plaintext.txt -out aes192cfb.enc -iter 10
enter AES-192-CFB encryption password:
Verifying - enter AES-192-CFB encryption password:

┌──(root㉿DESKTOP-JNTUATF)-[~/lab3]
└─# cat aes192cfb.enc
Salted__o◆Y ◆◆◆◆◆SI◆◆@◆s``
```

Рисунок 62 – AES-192-cfb

```
┌──(root㉿DESKTOP-JNTUATF)-[~/lab3]
└─# openssl enc -aes-192-cfb -d -in aes192cfb.enc -out aes192cfbdecrypted.txt -iter 10
enter AES-192-CFB decryption password:

┌──(root㉿DESKTOP-JNTUATF)-[~/lab3]
└─# cat aes192cfbdecrypted.txt
angry beavers
```

Рисунок 63 – AES-192-cfb

-salt	Use salt in the KDF (default)
-a	Base64 encode/decode, depending on encryption flag
-pbkdf2	Use password-based key derivation function 2 (PBKDF2)
-k val	Passphrase
-iter +int	Specify the iteration count and force the use of PBKDF2

ЗАКЛЮЧЕНИЕ

По результатам выполнения данной лабораторной работы я изучил, проанализировал и реализовал один цикл раундовой функции алгоритма AES. Также был произведен криптоанализ данного алгоритма. Были проанализированы принципы использования криптосистемы в современных приложениях на примере библиотеки openssl. В реализации одного цикла раундовой функции были отражены промежуточные результаты шифрования после всех этапов алгоритма AES.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Бабенко, Л. К. Современные алгоритмы блочного шифрования и методы их анализа / Л.К. Бабенко, Е.А. Ищукова. - М.: Гелиос АРВ, 2015. - 376 с.

2. Бабенко, Л.К. Современные интеллектуальные пластиковые карты / Л.К. Бабенко. - М.: Гелиос АРВ, 2015. - 921 с.

3. Болотов, А. А. Элементарное введение в эллиптическую криптографию. Протоколы криптографии на эллиптических кривых / А.А. Болотов, С.Б. Гашков, А.Б. Фролов. - М.: КомКнига, 2012. - 306 с.

4. Бузов, Геннадий Алексеевич Защита информации ограниченного доступа от утечки по техническим каналам / Бузов Геннадий Алексеевич. - М.: Горячая линия - Телеком, 2016. - 186 с.