

**МИНОБРНАУКИ РОССИИ**

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«МИРЭА – Российский технологический университет»**

|  |
| --- |
| **РТУ МИРЭА** |
|  |
| **Институт кибербезопасности и цифровых технологий (ИКБ)** |
|  |
| КБ-2 «Информационно-аналитические системы кибербезопасности» |

**ОТЧЕТ О ВЫПОЛНЕНИ ЛАБОРАТОРНОЙ РАБОТЫ №5**

**В РАМКАХ ДИСЦИПЛИНЫ «МЕТОДЫ И СРЕДСТВА КРИПТОРГАФИЧЕСКОЙ ЗАЩИТЫ ИНФОРМАЦИИ»**

Выполнил:

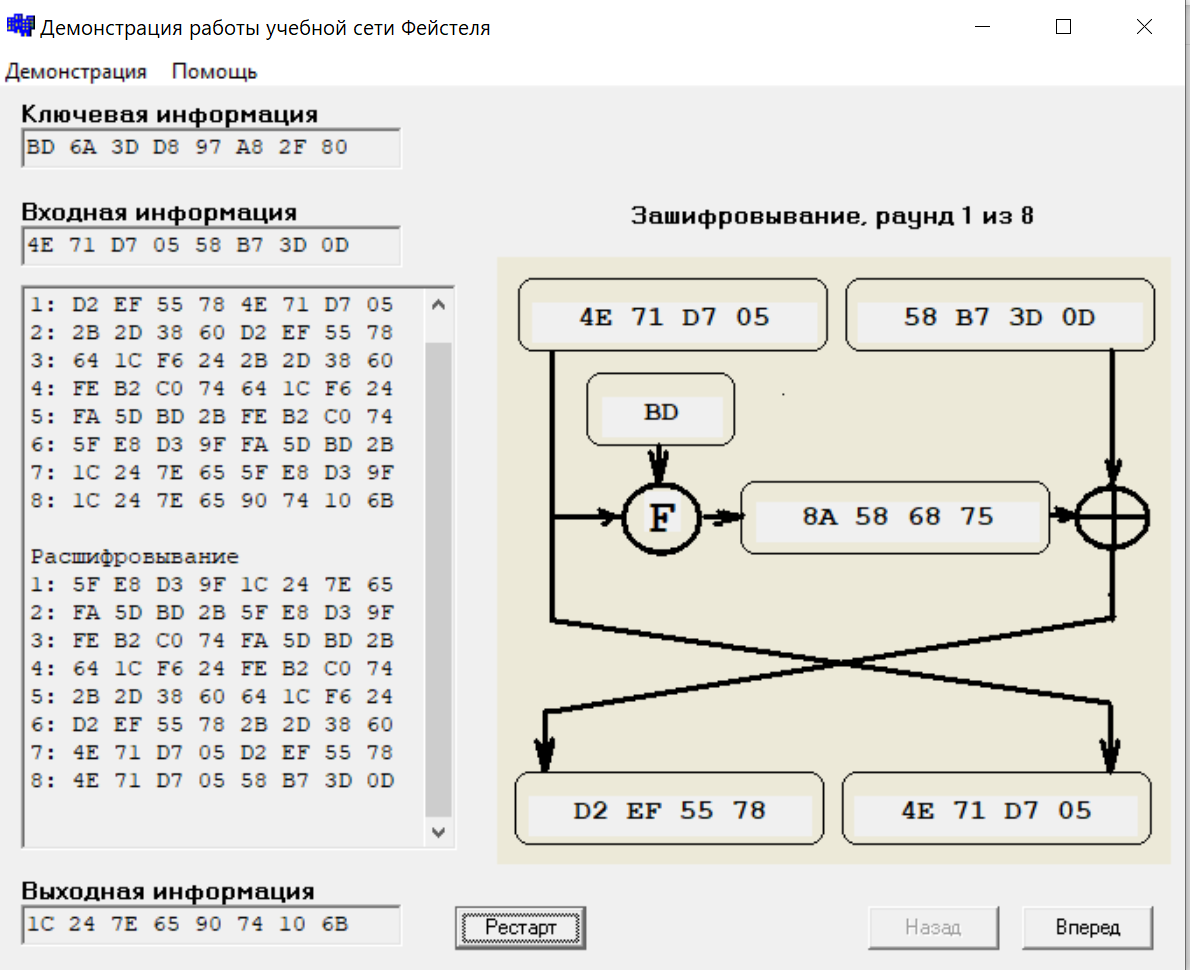
Студент 3-ого курса

Учебной группы БИСО-02-22

Зубарев В.С.

Цель работы: выяснить какие преобразования происходят в раундовой функции F.

Для лабораторной работы предоставлена программа Feistel.exe. Основное окно программы показано на рисунке.



Описание программы

Учебная сеть Фейстеля

Разрядность блока данных - 64 бита,

то есть две половинки по 32 бита.

8 раундов преобразования.

Разрядность ключа - 64 бита

Используется по 8 бит на каждый раунд

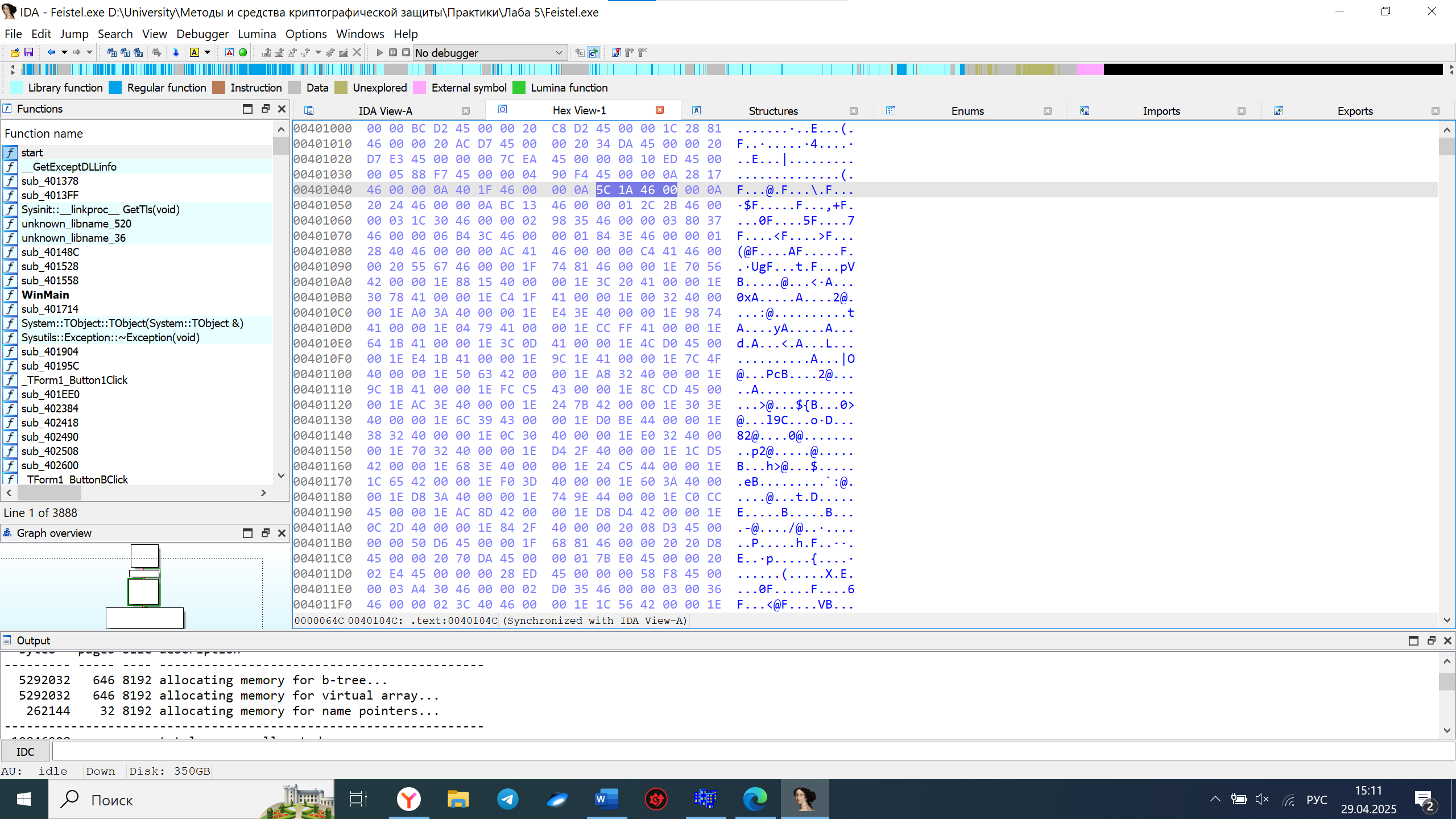
Порядок работы программы

Вначале случайным образом генерируются ключ и входная информация, затем проходит 8 раундов зашифровывания и следом 8 раундов

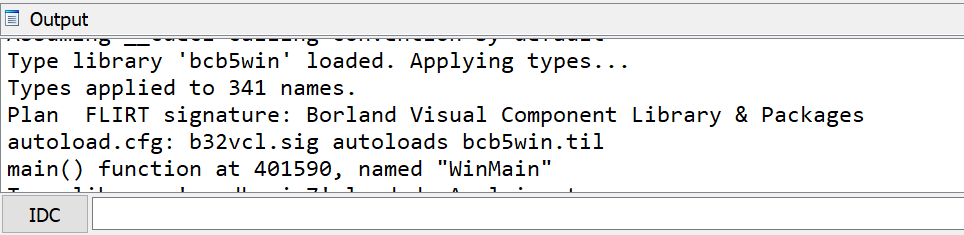
расшифровывания, при этом промежуточные результаты заносятся в протокол. Последняя строка (раунд 8) в протоколе зашифровывания представляет собой шифротекст, а последняя строка в протоколе расшифровывания - снова открытый текст.

Для вскрытия функции предположим, что сама функция использует ограниченный набор действий: побитовые сдвиги вправо и влево на определенное значение, операцию перестановки, операцию XOR (). Для выяснения алгоритма раундовой функции будем использовать дизассемблирование и декомпилирование кода программы. Для этого будем использовать программное обеспечение IDA Pro (v 7.7.3).

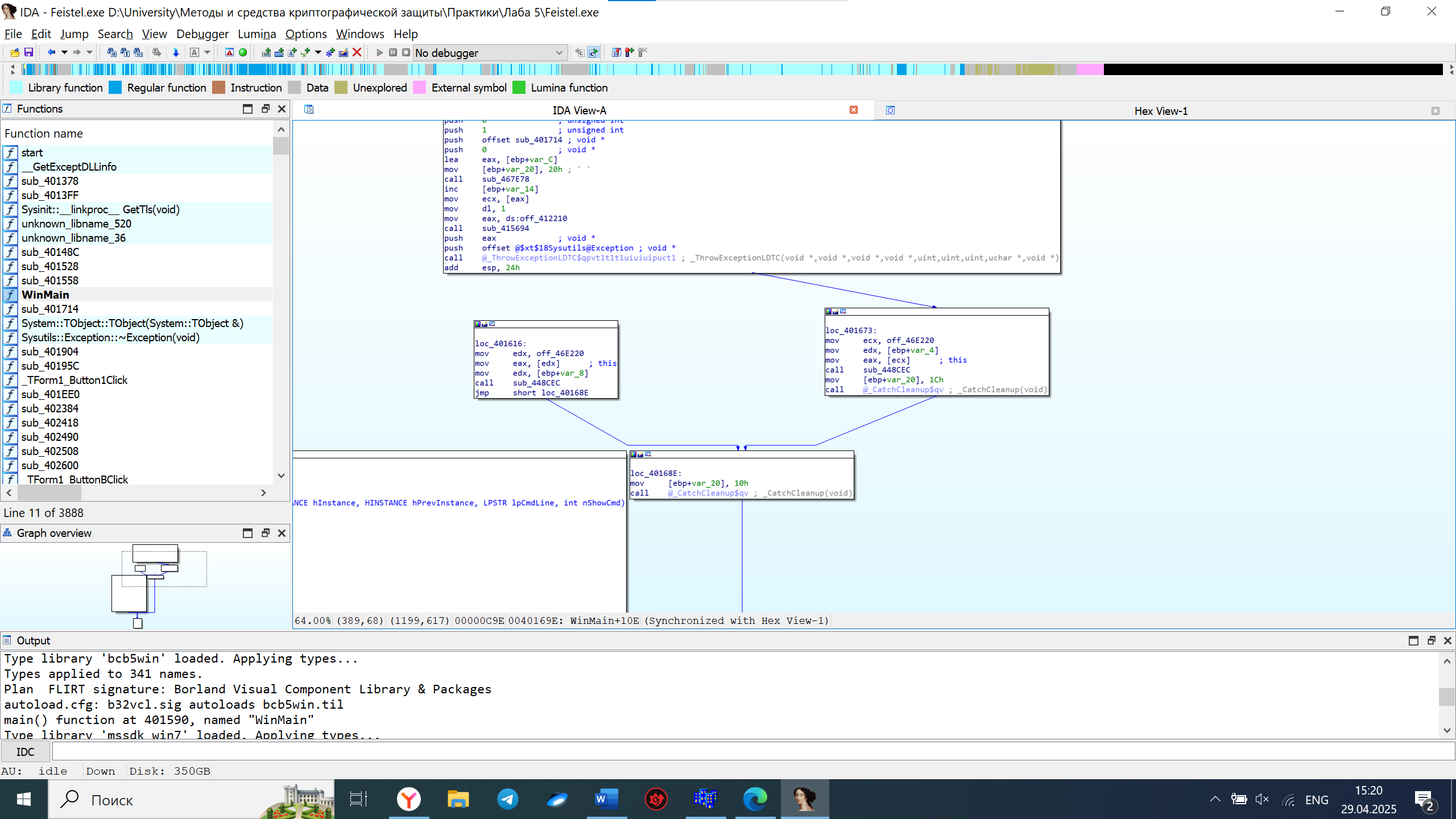
После загрузки исполняемого файла, программа проведет стандартный анализ бинарного кода и покажет ассемблерный вид (как на рисунке).



В окне output, можно увидеть, что программа определила языковой пакет, на котором написан исходник.

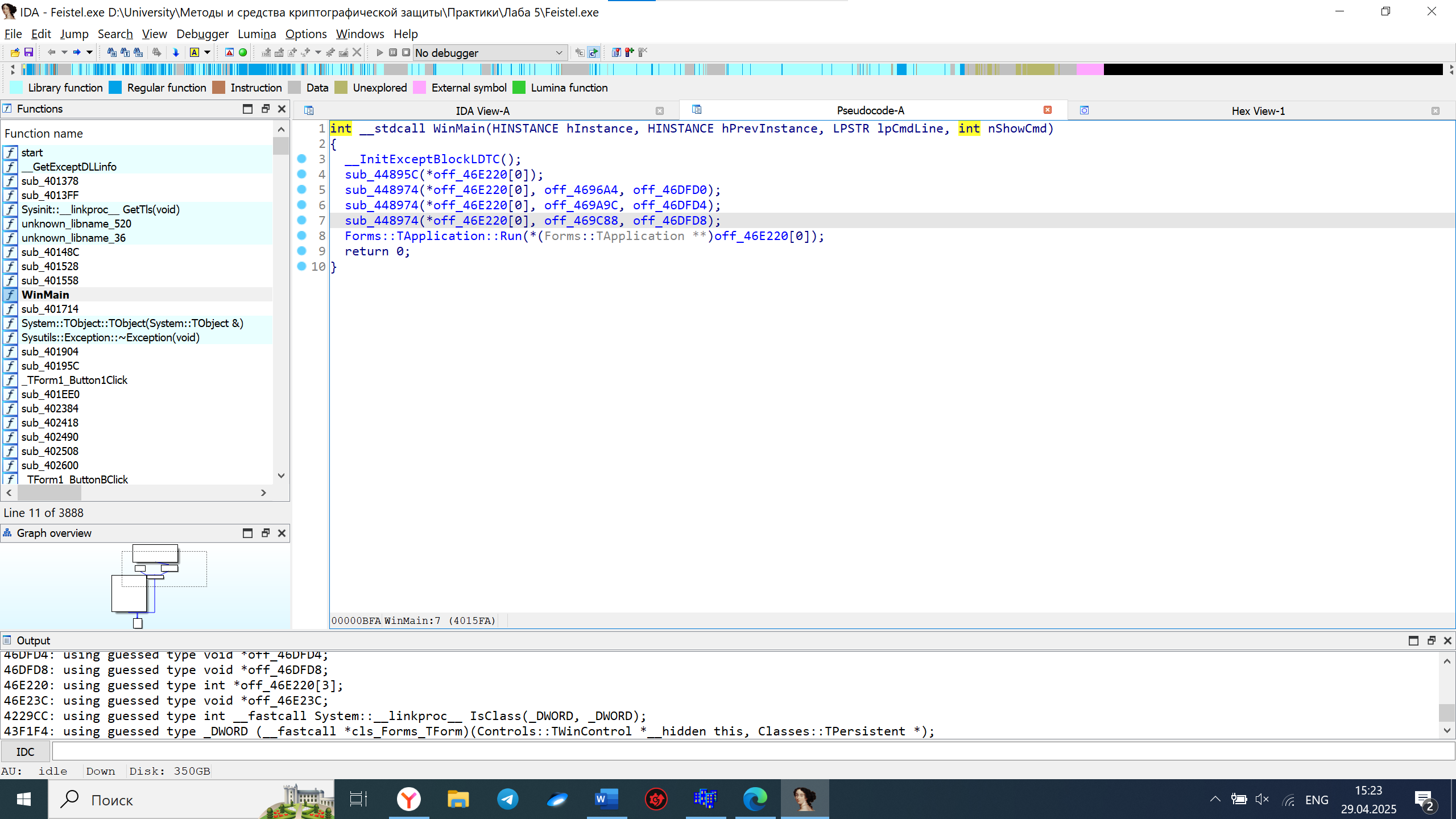


Данный модуль – это модуль визуализации для с, соотетсвенно сама программа так же написана на с. Это подтверждается также и тем что программа обнаружила функцию main (основная функция по умолчанию в с- подобных языках) и переименовала ее в WinMain.  
Во вкладке functions передем к этой функции и выведем более приемлимый формат функции main (вкладка IDA Vew-A).



На данном виде показана логика выполнения функции в виде графа (graph overview), однако читать ассемблерный код все еще не очень удобно. Для более понятной структуры есть возможность применить декомпиляцию диазассемблированного кода (по хоткею F5).

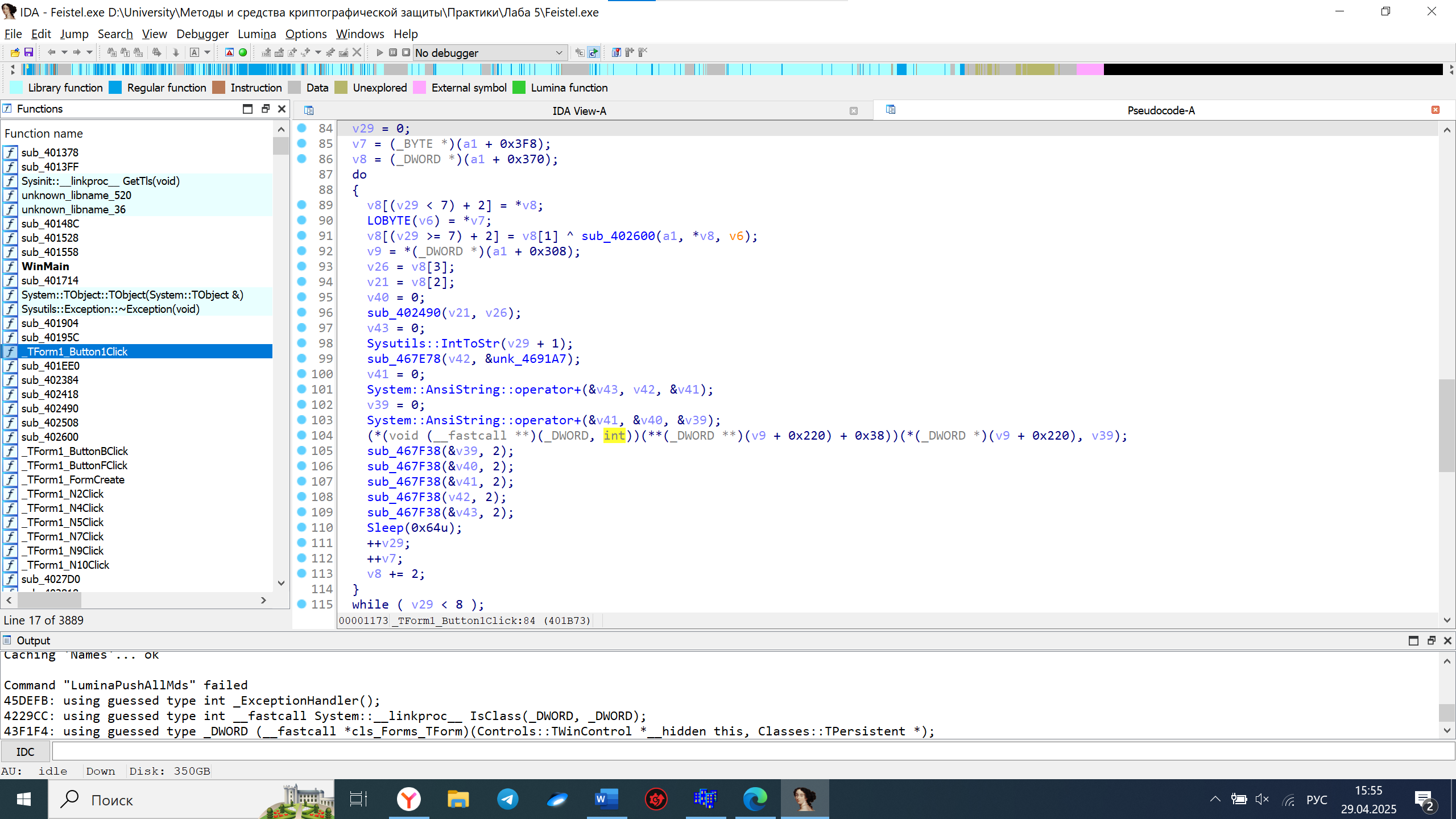
После выполнения декомпиляции появится вкладка Pseudocode-A. С-подобный псевдокод функции main представлен на рисунке ниже.



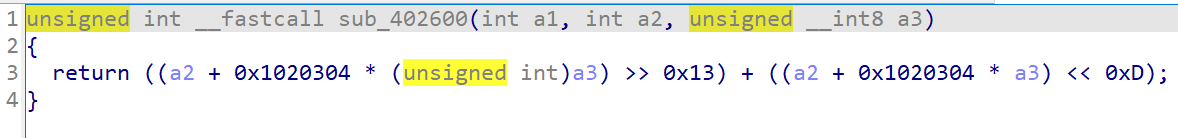
Теперь имея читабельное представление логики программы, перейдем к поиску раундовой функции. Так как программа шифрует открытый текст за 8 раундов и выводит промежуточные результаты, следовательно на один запуск приходится 16 вызовов раундовой функции (8 для шифрования и 8 для дешифрования). Для реализации цикла вызовов должны быть использованы циклы (while или for). Возможен вариант кода последовательно вызывается 8 раз раундовая функция. В таком случае необходимо понимать, то в модели Фейстеля преобразование блоков имеет вид

Тогда с точки зрения С -подобного песевдокода вид преобразования должен выглядеть как

Просмотрим функции на наличие похожих конструкций. Начнем с функций, которым программа дала осмысленное название. Например, \_TForm1\_ButtonClick. Очевидно, что данная функция вызывается по нажатию какой-то кнопки и меняет содержимое окна. Рассмотрим ее псевдокод. На определённом моменте вызываются два цикла while с постусловиями. Причем условием для выхода является превышение какой-то переменной значения 8.



Так же заметим, что внутри исполняемого блока встречается конструкция v8[(v29 >= 7) + 2] = v8[1] ^ sub\_402600(a1, \*v8, v6), которая похожа раундововое преобразование. Тогда sub\_402600 – это раундовая функция. Перейдем к ее псевдокоду.



Разберем, что делает эта функция шаг за шагом:

Из описания вызова понимаем что a3 – раундовое значение ключа, так как значение хранится в 8 битном формате.

Умножение и сложение: Функция начинает с выполнения умножения и сложения. Рассмотрим, что происходит внутри каждого из выражений.

(a2 + 0x1020304 \* (unsigned int) a3) — это выражение представляет собой суммирование текущего значения a2 (32 бита) с результатом умножения ключа a3 (8 бит) на фиксированную константу 0x1020304.

Константа 0x1020304 представляется в десятичной системе как 16909060.

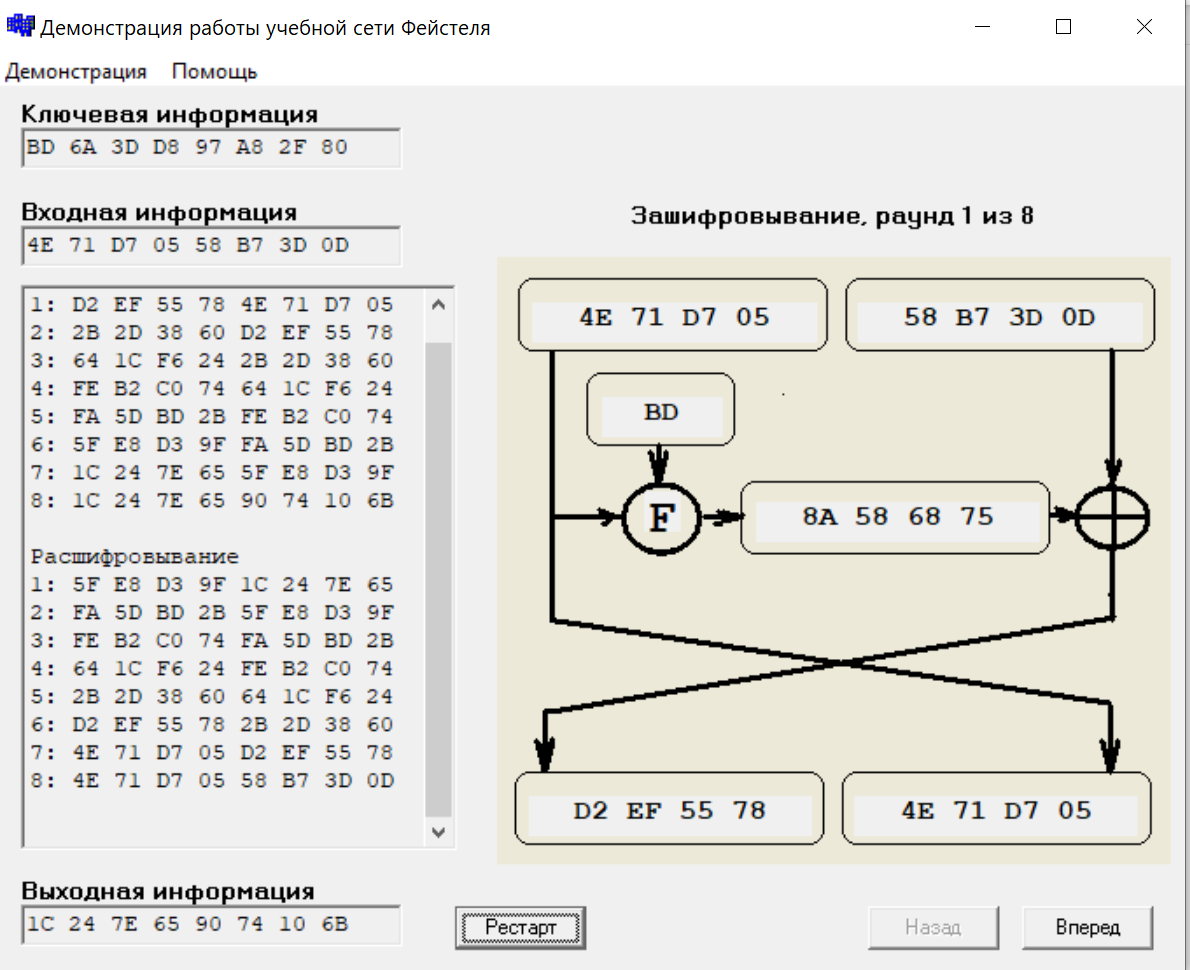
Сдвиги: далее функция выполняет два сдвига, чтобы обеспечить более сложные нелинейные преобразования:

>> 0x13 (сдвиг на 19 бит вправо): после того как мы выполнили сложение, результат сдвигается вправо на 19 бит.

<< 0xD (сдвиг на 13 бит влево): вторая часть этого выражения сдвигает результат влево на 13 бит.

Итоговая операция: эти два результата суммируются, создавая итоговое значение, которое будет использоваться для дальнейших шагов.

Проверим действительно ли данная функция является раундовой. Рассмотрим один шаг алгоритма. Вот как первый раунд подставлен в программе.



**Входные данные:**

**Левая часть данных (a2)**: 4E 71 D7 05 → 0x4E71D705

**Ключ (a3)**: BD → 0xBD (в десятичной системе это 189).

**Шаги выполнения:**

**Шаг 1: Умножение ключа на константу**

Константа: 0x1020304 = 16909060 (в десятичной системе).

Умножаем ключ a3 = 0xBD = 189 на константу 0x1020304:

189\*0x1020304=189\*16909060=3196178340189

**Шаг 2: Первая часть функции**

Теперь выполняем первую часть функции:

(a2+0x1020304\*(unsigned int)a3) >> 0x13

Подставляем значения:

a2=0x4E71D705, 0x1020304\*a3=3196178340

Сначала складываем:

0x4E71D705+3196178340=0xBD6A3DD8+3196178340=0x1141A10E5

Затем сдвигаем вправо на 19 бит:

0x1141A10E5 >> 0x13=0x8A

**Шаг 3: Вторая часть функции**

Теперь выполняем вторую часть функции:

(a2+0x1020304×a3) << 0xD

Подставляем значения:

a2=0x4E71D705, 0x1020304×a3=3196178340

Сначала складываем:

0x4E71D705+3196178340=0xBD6A3DD8+3196178340=0x1141A10E5

Затем сдвигаем влево на 13 бит:

0x1141A10E5 << 0xD=0x68A805000

**Шаг 4: Сложение частей**

Теперь складываем результат первой и второй частей:

0x8A+0x68A805000=0x68A80508A

**Шаг 5: Преобразование в 32 бита**

После того как мы вычислили результат работы раундовой функции, нам нужно получить только 32 бита из полученного значения. Это необходимо, поскольку мы работаем с блоками данных длиной 32 бита, и не можем использовать больше, чем 32 бита на каждом шаге.

**Подробное описание**

На шаге **сложения частей** мы получили промежуточный результат:

0x68A80508A

Этот результат в шестнадцатеричной системе представляет собой **40 бит**:

**Преобразование 40 бит в 32 бита**

1. **Запишем результат в двоичной системе**:

0x68A80508A=110100010101000000000101000010102

1. Поскольку мы работаем с 32 битами, мы просто отбрасываем старшие 8 бит:

* Наши 40 бит:  
  11010001010100000000010100001010 (это 40 бит).
* Чтобы оставить только 32 младших бита, мы игнорируем старшие 8 бит, оставляем последние 32 бита:  
  1010000000000101000010102

1. Переводим обратно в 16-тиричную систему:  
   0x8A 58 68 75.

Таким образом, мы получаем итоговый результат раундовой функции в 32 бита:

0x8A586875

Результат совпал с ожиданиями. Делаем вывод о том, что что найденная функция – раундовая.