**BC/NW 2022 № 1 (39):8.1**

**ВОССТАНОВЛЕНИЕ ИСХОДНОГО КОДА И СТРУКТУРЫ ПРОГРАММЫ ПО ИСПОЛНЯЕМОМУ ФАЙЛУ**

(Методическое пособие)

**Абросимов Л.И., Орлов Д.А., Качалин М.Н.**

**Введение**

В настоящее время сторонние разработчики в целях информационной безопасности большую часть программного обеспечения поставляют без исходного кода на языке высокого уровня. Из-за этого осложняется выявление рисков использования таких приложений, так как ни бинарный код, ни код полученный в результате дизассемблирования не даёт возможность оценить взаимосвязь элементов и идентифицировать стандартные алгоритмические конструкции, используя разумные трудозатраты. С такими проблемами также можно столкнуться при создании аналогов зарубежного ПО для внутреннего рынка. Однако восстановленная программа упрощает преодоление вышеуказанных трудностей. Для помощи специалистам в области информационной безопасности, то есть повышения уровня абстракции представления, может использоваться декомпилятор.

Под декомпилятором необходимо понимать инструмент, который получая программу написанную на языке ассемблер, выдаёт эквивалент данной программе, но на языке высокого уровня.

Существует одна проблема, при компиляции из языка высокого уровня используется отображение «многие к одному» концепций языка высокого уровня к концепциям языка ассемблера, что приводит к невозможности однозначного восстановления программы. Это приводит к необходимости работать в паре декомпилятор с аналитиком. Роль аналитика, в таком случае, состоит в том, чтобы управлять процессом декомпиляции, например, выявление подпрограмм и возвращаемых ими значений или восстановления типов данных. Эти задачи сложны, трудоёмки и программно невыполнимы, на нынешнее время не существует декомпилятора, способного самостоятельно работать и справляться с восстановлением программы до какого-либо языка высокого уровня. Для облегчения решения задач возможно использовать набор декомпиляторов в некоторой композиции.

Так же любители могут столкнуться с такой необходимостью при желании поиграть в старые игры, которые не могут быть запущены на современных компьютерах. Или же когда онлайн игра в жанре ММОРПГ закрывается, а энтузиасты воссоздают и клиенты, и серверы данной игры на основе файлов клиента и образцов пакетов данных, отправляемых и принимаемых сервером.

В настоящей работе рассматривается пример обратной разработки на примере программы шифрования.

Так же необходимо сказать, что описанный далее процесс, а именно восстановление исходного кода затрагивает закон об интеллектуальной собственности, конечно в данной работе используется для рассмотрения только созданное мной программное обеспечение.

Однако, в целом создатели справедливо опасаются, что с восстановлением исходного кода будут раскрыты методы и алгоритмы работ их программ. Поэтому стоит упомянуть, что если целью рассматриваемого процесса стоит уничтожение защиты от копирования, то это нарушение закона, однако само по себе восстановление исходников не является незаконным. Если бы ситуация была бы обратной, это напоминало бы запрет открытия капота вашего автомобиля для его ремонта.

Постановка задачи

Целью работы является создание дидактического пособия для углубленного изучения студентами и бакалаврами технологий ассемблирования и дизассемблирования.

Для достижения цели настоящей научно-исследовательской работы необходимо выполнить ряд задач:

1. Разработать алгоритм и исходную программу по известному методу шифрования способом сложения с ключом и получить исходный исполняемый файл (для последующего восстановления).

2. Выбрать инструмент дизассемблирования (ИДА) и реализация настройки и запуска, выбранного ИДА, разработать базу данных функционирующего ИДА (для проведения анализа используемых команд).

3. Провести дизассемблирование и получить восстановленную структуру программы и узнать заложенный ключ.

4. Создать программу «Шифрование сложение с ключом» работающую по найденной методике на языке C++.

5. Провести тестирование на цифровом примере и сохранить результаты в базе данных.

6. Выполнить сравнение результатов работы, исходной и восстановленной программ, при одинаковых исходных данных.

7. Создать дидактические материалы с пояснением настроек и работы ИДА при выполнении заданий в учебных целях для углубленного изучения технологий дизассемблирования.

Уточнённая постановка задачи

Необходимо для примера написать исходную программу, использующую метод шифрования способом сложения с ключом, на языке Assembler с использованием MS-DOS. Программа шифрования методом сложения с ключом имеет простой алгоритм и структуру. Ключ для шифрования будет задан внутри кода программы, чтобы можно было проверить себя или студента на правильность выполнения анализа исполняющего файла.

В данной работе выбран MS-DOS так как, на его примере проще показать основы работы процессоров архитектуры x86 и принципы написания программ на Ассемблере, когда необходимо запускать старые, устаревшие программы на новых платформах, не поддерживающих систему MS-DOS.

Для анализа исполняющего файла необходимо выбрать Интерактивный Дизассемблер (ИДА). Для решения поставленной задачи, была выбрана программа IDA PRO 7.5 SP3, на следующих основаниях:

• IDA Pro поддерживает много форматов, одними из которых являются MS DOS и EXE File [6], позволяющие программировать и анализировать рассматриваемый пример

• IDA Pro поддерживает очень большое количество процессоров и их модификаций [6]

• Частое обновление и поддержка[6]

Известно множество способов шифрования, и это большой пласт программ. Для реализации примера возьмём лишь один способ как пример: метод шифрования сложения с ключом

Выбранный метод шифрования заключается в сложении кодов символов ключа и каждой строки длинны ключа из файла, дешифрирование заключается в том, чтобы вычесть из символов ключ.

Так же для просмотра информации о бинарном коде или микрокоде, нам может понадобиться программа FAR manager [8].

FAR Manager — консольный файловый менеджер для операционных систем семейств Microsoft Windows и Linux. Он в отличии от встроенных блокнота и других представлений, способен представлять информацию в шестнадцатеричном формате при открытии файла клавишей F3 и переводе его в шестнадцатеричный формат клавишей F4[8].

1 **Разработка исходной программы и анализ исполняющего файла**

1.1 Создание программы шифрования

Для выполнения ВКР создана исходная программа шифрования на assembler.

Созданная программа принимает, в качестве исходных данных, название входного файла длинной до 20 символов в формате \*\*\*.txt и выходного файла длинной до 20 символов в формате \*\*\*.txt. После этого пытается открыть исходный файл для чтения, если это не удаётся выводиться сообщение об ошибке и программа прекращается. В ином случае программа пробует открыть второй файл для записи

Шифрование методом сложения с ключом состоит в том, чтобы складывать каждый i символ из исходного файла с i mod key\_length ключа, то есть происходит сложение каждой строки длины ключа из файла с ключом посимвольно.

Этот метод похож на шифр цезаря, которым некоторые дети пользовались в школе, но его намного труднее подобрать так как сдвиг от символа к символу меняется, и не зная длинны ключа будет сложно найти закономерность.

Пример 1

Входные данные:

namefld\_in =’aaa.txt’

namefld\_out =’abb.txt’

изначальное содержание файла aaa.txt

abc

полученное содержание файла abb.txt

МЗЬ

Дополнительная информация по тестам в параграфе 2.3

1.1.1 Разработка структуры программы (Описание переменных, процедур и макросов, используемых в исходной программе)

Список переменных, которые использовались в исходной программе шифрования на assembler с их описанием приведены в таблице 1.1

Таблица 1.1. Список переменных.

|  |  |  |
| --- | --- | --- |
| Имя | Метка | Примечание |
| requst\_in\_file | db | Сообщение. Приглашение к вводу входного файла |
| requst\_out\_file | db | Сообщение. Приглашение к вводу выходного файла |
| err\_in\_file\_m | db | Сообщение. Ошибка открытия входного файла |
| err\_out\_file\_m | db | Сообщение. Ошибка открытия выходного файла |
| err\_key\_m | db | Сообщение. Ошибка при вводе ключа |
| mess\_done | db | Сообщение. Оповещение о выполнении |
| new\_str | db | Символы перевода каретки на новую строку. |
| buff | db | Переменная для хранения считываемых данных. |
| handle\_in | dw | Переменная для хранения файлового индекса входного файла |
| handle\_out | dw | Переменная для хранения файлового индекса выходного файла |
| key\_len | dw | Переменная для хранения значения количества символов ключа |
| count | dw | Переменная для хранения значения количества считанных байт. |

Список процедур и макросов, используемых в программе с их описанием приведены в таблице 1.2

Таблица 1.2. Список процедур, макросов и их назначение.

|  |  |
| --- | --- |
| INITSRD | Макрос производит инициализацию регистров DS и ES. |
| input\_kbd | Макрос для вывода на экран приглашения для ввода данных с клавиатуры. |
| read | Макрос для выполнения чтения из файла. |
| write | Макрос для выполнения записи в файл. |
| init | Макрос для инициализации регистров. |
| err\_in\_file | Процедура выводящая сообщение об ошибке в открытии входного файла. |
| err\_out\_file | Процедура выводящая сообщение об ошибке в открытии выходного файла. |
| coding | Процедура выполняющая кодирование по методу сложения с ключом. |
| exit | Процедура закрывающая открытые файлы и завершающая программу. |

*1.1.2 Схема алгоритма исходной программы шифрования на assembler*

Далее приведены пояснения создания алгоритма данной программы, которая представлена ниже на рисунке 1.1.

В любой программе есть инициализация (блок 1 на Рис.1.1) которая необходима, так как без первоначальной настройки программа не сможет работать правильно.

После этого нам необходимо получить имена файлов входного и выходного (блок, отмеченный цифрой 2 на Рис1.1), далее надо проверить оба файла на открытие, и, если один из них не открылся, вывести отчёт об ошибке и завершить программу это является обработкой граничных или другими словами аномальных ситуаций, которые необходимо обработать, чтобы программа завершалась правильно, с сообщением об ошибке, а не экстренно прекратилось, пытаясь получить данные из неоткрытого файла. Их небольшой анализ приведён в таблице 1.3.

Если оба файла открыты и готовы к шифрованию начинаем шифровать до конца входного файла. (блок 5 на Рис 1.1)

После окончания процедуры закрываем оба файла, сообщаем пользователю что всё готово и завершаем программу, это входит в процедуру завершения, в которой надо вывести отчёт что программа завершена и завершить программу, вызвав функцию MS-DOS аналог return 0 и языка программирования C++.

Схема алгоритма представлена на рисунке 1.1:

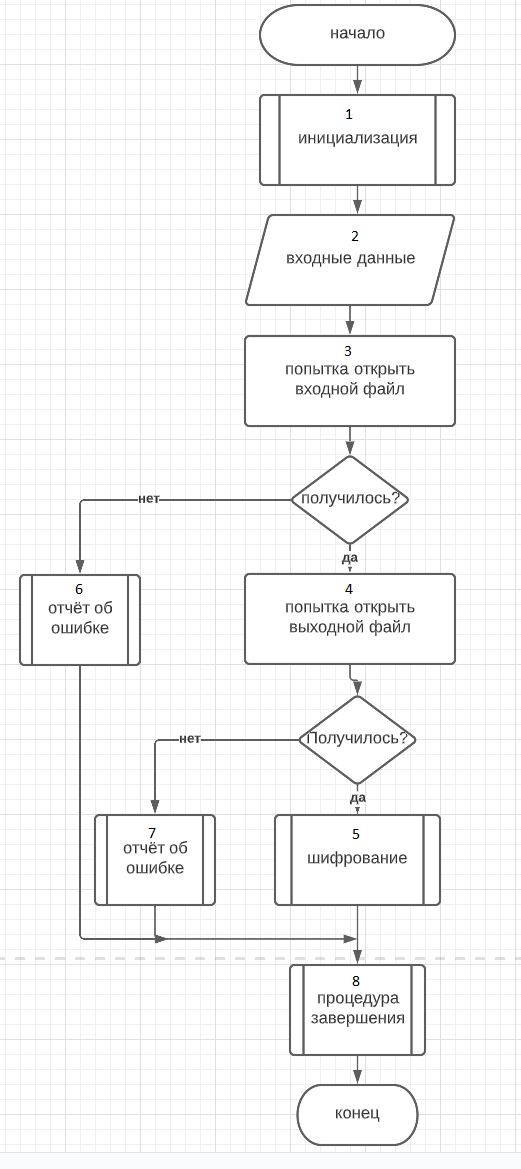


Рис.1.1 схема алгоритма исходной программы шифрования на assembler

Таблица 1.3. Аномальные ситуации

|  |  |  |
| --- | --- | --- |
| № | Описание | Сообщение или действие |
| 1 | Не удалось открыть входной файл или нет входного файла с таким именем или больше 20 символов в названии или неправильно введено название | error: cannot open sourse file |
| 2 | Не удалось открыть выходной файл или неправильно введено имя или больше 20 символов в названии | error: cannot open out file |
| 3 | Входной файл пуст | Выходной файл будет тоже пустым |

*1.1.3 Используемые в программе функции DOS*

Далее будут приведены используемые функции системы MS-DOS интерфейс шифрование файл макрос

**Для вывода строки на экран используется:**

Функция **09H** прерывания **INT 21H**

**Входные параметры**

* AH = 09H
* DS:DX = адрес строки, заканчивающейся символом '$' (ASCII 24H)

**Для ввода строки в буфер:**

Функция **0ah** прерывания **21h**.

**Входные параметры**

* AH = 0aH
* DS:DX = адрес входного буфера (смотри ниже)

**Вывод**

* нет = буфер содержит ввод, заканчивающийся символом CR (ASCII 0dH)

**Для открытия файла:**

Функция **3dH** прерывания **INT 21H**

**Входные параметры**

* AH = 3dH
* DS:DX = адрес строки ASCIIZ с именем файла
* AL = режим открытия

**Вывод**

* AX = код ошибки если CF установлен и файловый индекс если нет ошибки.

**Описание:**

DS:DX указывает на строку ASCIIZ в формате: "d:\путь\имяфайла",0. Если диск и/или путь опущены, они принимаются по умолчанию.

* файл должен существовать.
* файл открывается в выбранном режиме доступа / режиме открытия:
  + AL = 0 чтобы открыть для чтения
  + AL = 1 чтобы открыть для записи
  + AL = 2 чтобы открыть для чтения и записи

**Для создания файла:**

Функция **3сН** прерывания **INT 21Н**

**Входные параметры**

* AH = 3cH
* DS:DX = адрес строки ASCIIZ с именем файла
* CX = атрибут файла

**Вывод**

* AX = код ошибки если CF установлен и файловый индекс если ошибки нет

**Описание:**

DS:DX указывает на строку ASCIIZ в формате: "d:\путь\имяфайла",0. если диск и/или путь опущены, они принимаются по умолчанию.

* файл создается в указанном (или умалчиваемом) оглавлении
* файл открывается в режиме доступа чтение/запись
* вы должны сохранить файловый индекс (handle) для последующих операций
* если файл уже существует:
* при открытии файл усекается до нулевой длины
* если атрибут файла – только чтение, открытие

**Для чтения данных из файла:**

Функция **3FН** прерывания **INT 21Н**

**Входные параметры**

* AH = 3fH
* BX = файловый индекс
* DS:DX = адрес буфера для чтения данных
* CX = число считываемых байт

**Вывод**

* AX = код ошибки если CF установлен или число действительно прочитанных байт

**Описание:**

CX байт данных считываются из файла с файловым индексом, указанным в BX. данные читаются с текущей позиции указателя чтения/записи файла и помещаются в буфер вызывающей программы, адресуемый через DS:DX.

**Для записи в файл:**

Функция **40Н** прерывания **INT 21Н**

**Входные параметры**

* AH = 40H
* BX = файловый индекс
* DS:DX = адрес буфера, содержащего записываемые данные
* CX = число записываемых байт

**Вывод**

* AX = код ошибки если CF установлен
* AL = число реально считанных байт (лучший тест для ошибок)

**Описание:**

CX байт данных записывается в файл с файловым индексом, заданным в BX. данные берутся из буфера, адресуемого через DS:DX. данные записываются, начиная с текущей позиции указателя чтения/записи файла.

**Для закрытия файла:**

Функция **3Eh** прерывания **INT 21h**

**Входные параметры**

* AH = 3eH
* BX = файловый индекс

**Вывод**

* AX = код ошибки если CF установлен

**Описание:**

BX содержит файловый индекс (handle), возвращенный при открытии. файл, представленный этим индексом, закрывается, его буфера сбрасываются, и оглавление обновляется корректными размером, временем и датой.

*1.1.4 Пользовательский интерфейс*

Программа работает как консольное приложение, поэтому весь интерфейс представляет собой консоль с выводящимися на экран сообщениями. Сначала приглашающими подать входные данные потом либо отчёт об ошибке, либо же сообщение об удачном завершении программы.

Первым сообщением будет выводиться сообщение

sourse file is :

это приглашение напечать название входного файла

далее после ввода возможно два варианта развития, либо выведется сообщение

error: cannot open sourse file

это будет значит, что не удалось открыть входной файл. За этим будет следовать прекращение программы.

Или

out file is :

Если пользователь видит данное сообщение — это значит, что удалось открыть входной файл и программа просит ввести имя выходного файла, далее опять возможно два варианта после ввода. Если пользователь видит следующее сообщение

error: cannot open out file

это значит, что не удалось открыть выходной файл и результат будет как при прошлом сообщении об ошибке. Если это сообщение не получено пользователем будет выведено сообщение

done.

Это означает что приложение завершило работу и выводиться оно не взирая было ли выведено сообщение об ошибке или нет.

Далее будет приведён пример, когда пользователь ввел имя не существующего входного файла (Рис 1.2)

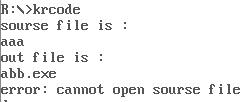


Рис.1.2 пример использования программы при неправильных входных данных

*1.1.5 Получение исполняющего файла*

Далее необходимо получить исполняющий файл в формате .exe ведь именно в таком формате получается пользователями программное обеспечение для пользования.

Использовав программу DOSBox для того чтобы взаимодействовать с файлами системы DOS, так же использовав ассемблер TASM получаем исполняющий файл

Выбор Ассемблера Turbo Assembler(TASM) обусловлен тем же что и выбор работы с DOS (дисковой операционной системы).

Для данных действий нам надо открыть DOSBox и проследовать в папку хранения файла с расширением, .asm в котором храниться код нашего исходного приложения

// изображение

Так же необходимо убедиться, что в этой же папке находиться все необходимые файлы ассемблер.

Далее двумя командами

Tasm имя\_файла

И

Tlink имя\_файла

После данных команд был получен файл с расширением .exe который можем используя DOSBox запустить введя команду имя\_файла.exe то есть просто запустив исполняющий файл

1.2. Анализ двоичного исполняющего файла

С помощью консольного файлового менеджера FAR откроем и посмотрим, как выглядит наш исполняющий файл в шестнадцатеричном формате

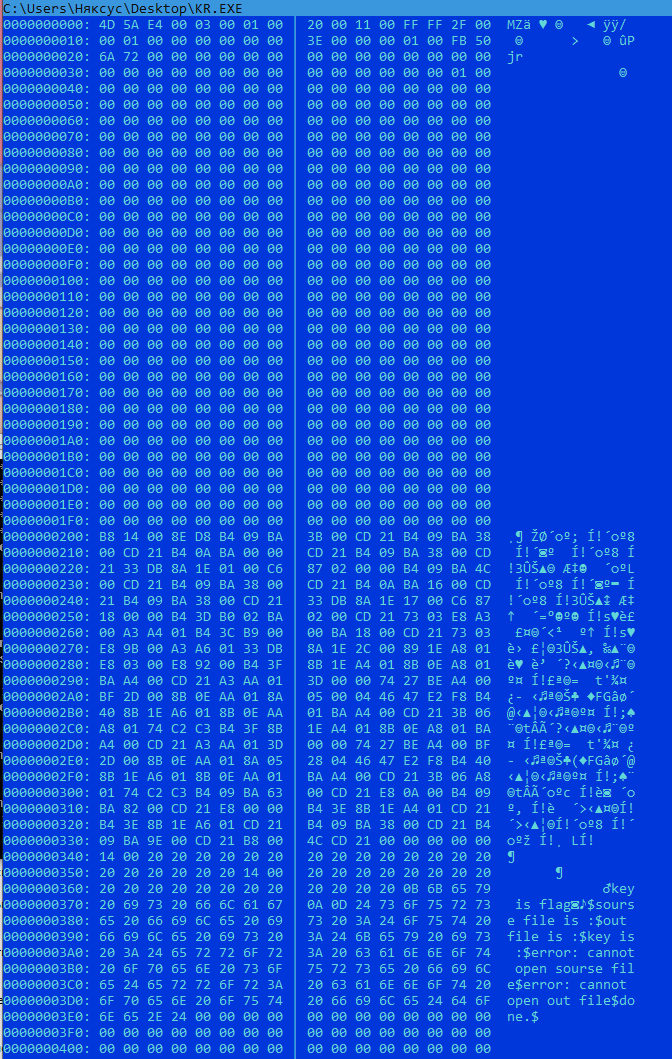


Рис. 1.3 представление исполняющего фала в программе FAR

На Рис. 1.3 можем определить начало и конец кода программы по адресу 200 до 33C и сегмент данных с адреса 37D до адреса 3E4.

Проводить анализ шестнадцатеричного представления сложно без привязки к командам, поэтому вернулись к IDA, которая покажет нам, какие коды соответствуют каким командам.

На верхней панели есть открытый файл с именем Hex View-1, если его нет можно воспользоваться комбинацией клавиш alt+3 или с помощью верхнего меню во вкладке windows в выпадающем меню выбрать данный файл, который нам откроет следующее:

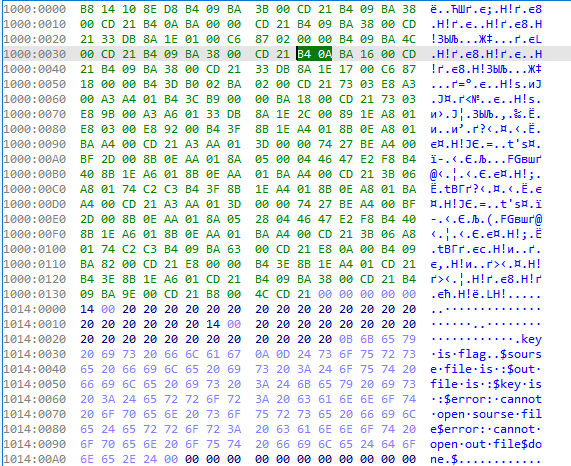


Рис. 1.4 представление исполняющего фала в программе IDA Pro

На рисунке 1.4 видно, что те же значения байтов что были на рисунке 5.1 находились с 200 до 410 адреса расположены уже с 1000:0000 до 1000:0145 адреса, и сегмент данных с 1015:002E по 1015:00A5

Выделяя команду в окне IDA view-A и переходя в окно Hex View-1, Будет выделяться соответствующие этой команде шестнадцатеричное представление, например:

B4 09 соответствует команде mov ah,9 и не важно в какой части программы она заложена.

Mov dx,3Ch отображается как BA 3C 00 из этого можно сделать вывод, что длина команд бывает разная.

Необходимо провести сравнение двух команд.

mov ah, 3Ch и mov ah,9

B4 3C B4 09

Можно заметить, что меняется только вторая половина команды причем именно на то, что необходимо сохранить в регистр ah. Это означает, что команда B4 – команда сохранения какой-либо информации и за ней будет следовать байт информации, который необходимо сохранить.

Попробуем сравнить другие команды, например,

mov dx, 38h и mov dx, 16h

BA 38 00 BA 16 00

Если пользоваться той же логикой, то бит BA отвечает за код команды соответствующий сохранению информации в регистр dx

Разберём подробнее команду mov ah,9

Её вид в hex формате B4 09 в двоичном будет

1011 0100 0000 1001

Первый байт всегда отвечает за код команды, Код операции может занимать от одного до трех байт. Для некоторых машинных команд часть битов кода операции может находиться в байте mod r/m.[7] (Рис. 5.3)

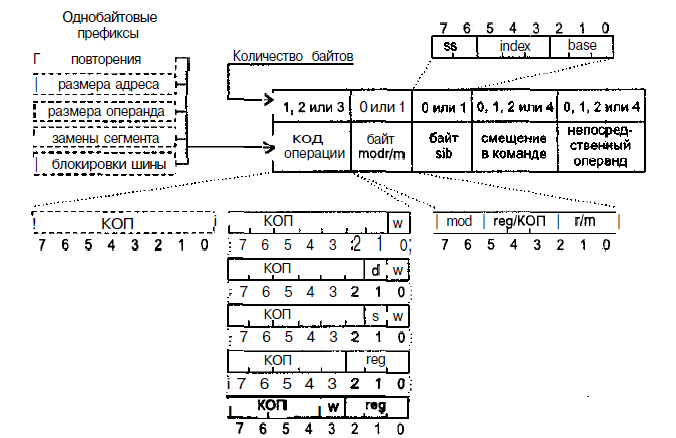
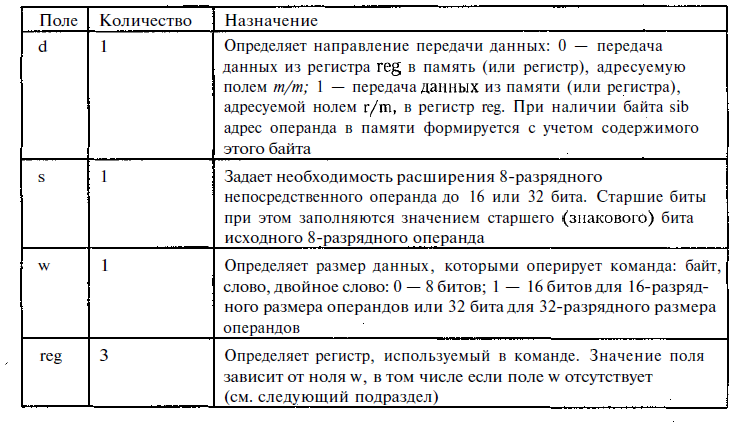


Рис1.5 формат машинной команды

Необходимо обратить внимание на самый нижний вариант представления второй половины первого байта (Рис. 1.5).

Табл. 1.4 назначение дополнительных битов поля кода операции[7]



Бит w=0 определяет размер данных, с которыми работает данная команда и 0 означает 8 битов (Таблица 1.4)

Reg = 100 при w=0 это указывает на регистр AH (Таблица 1.5 и 1.6)

Табл.1.5 Значение кодов в поле reg(поле w присутствует)[7]

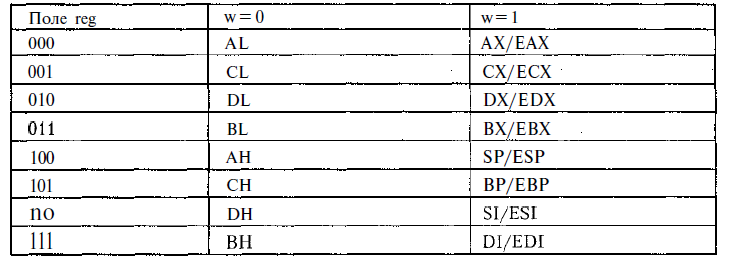
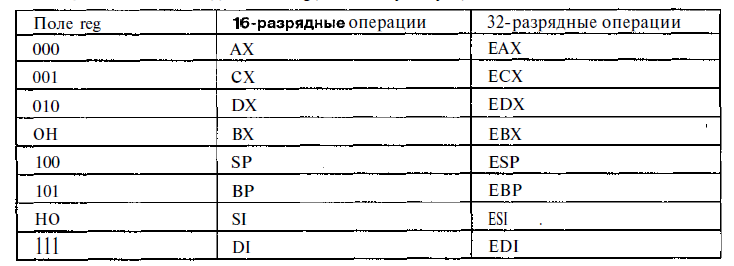


Табл. 1.6 Значение кодов в поле reg(поле w отсутствует)[7]



Из вышеперечисленного становится понятно, что:

10011 | 0 | 100 | 0000 1001

КОП | w | reg | байт который сохраняется (Табл. 5.2)

КОП- код операции

За процесс шифрования отвечают следующие байты

A3 A4 01 B4 3C B9 00 00 BA 18 00 CD 21 73 03

Также можно представить в виде кода:

mov word\_102E4, ax

mov ah, 3Ch

mov cx, 0

mov dx, 18h

int 21h

jnb short loc\_1007

Для удобства все соотношения представлены в таблице 1.7

Табл. 1.7 таблица соответствия байтов и команд основного цикла шифрования

|  |  |
| --- | --- |
| команда | байты |
| mov word\_102E4, ax | A3 A4 01 |
| mov ah, 3Ch | B4 3C |
| mov cx, 0 | B9 00 00 |
| mov dx, 18h | BA 18 00 |
| int 21h | CD 21 |
| jnb short loc\_10073 | 73 03 |

**2 Восстановление данных и структуры исходной программы, Создание восстановленной программы на с++**

2.1 Анализ исполняющего файла

Для анализа исполняющего файла был выбран интерактивный дизассемблер IDA PRO

При открытие данного приложения перед пользователем появляется окно (Рис 2.1), которое позволяет: начать дизассемблирование нового файла, открыть окно без проекта, вернуться к прошлому процессу или выбрать проект из списка.

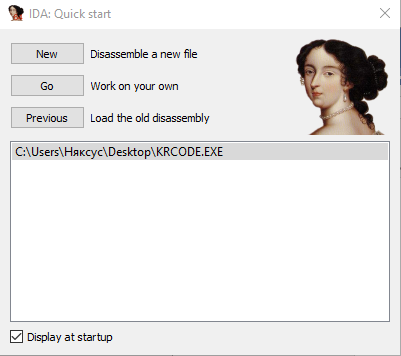


Рис. 2.1 меню быстрого старта программы IDA

После нажатия кнопки new и указания пути к исполняющему файлу формата .exe начинается автоматическая обработка и создание базы данных. В результате этого нам открывается окно для работы в IDA слева мы видим область где выписаны функции область 1(Рис 2.2), по центру выведена основная программа или точнее функция start которая единственная не теряет названия область 2(Рис 2.2), а остальные функции своё название теряют при ассемблировании, как и названия переменных. Под областью 1 видно всю структуру и то где на этой структуре находиться экран смотрящего.

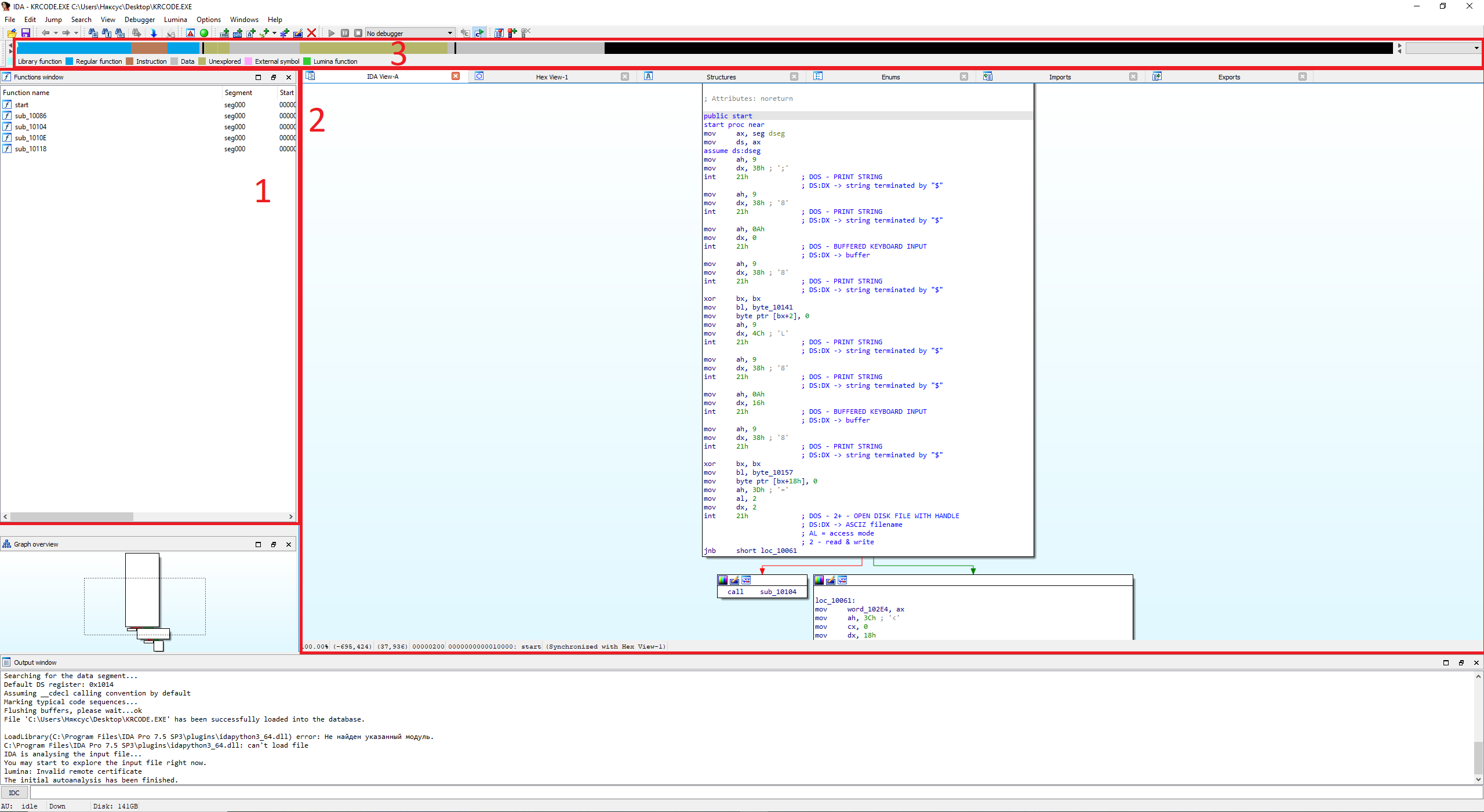


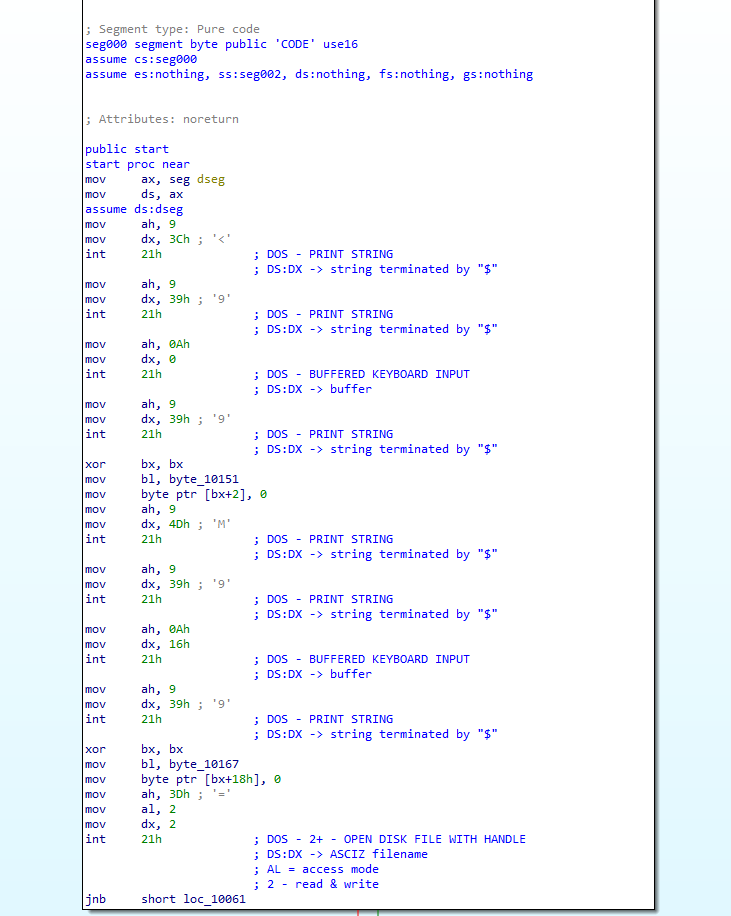
Рис.2.2 Рабочее окно программы IDA PRO

Так же выше прошлых описанных областей, но ниже меню находится полоса сегментов программы, выделенных цветом, а при наведении появиться подсказка область 3(Рис 2.2), нас будут интересовать 2 сегмента, это сегмент кода и сегмент данных. В первом храниться команды, которые необходимы для восстановления алгоритма, а во втором должен находиться ключ шифрования и все сохранённые переменные.

На Рисунке 2.3 приведена область с программным кодом

При анализе файла необходимо достичь поставленных целей, а именно:

1. Восстановить основную структуру программы
2. Восстановить назначение процедур и используемых строк и ключ



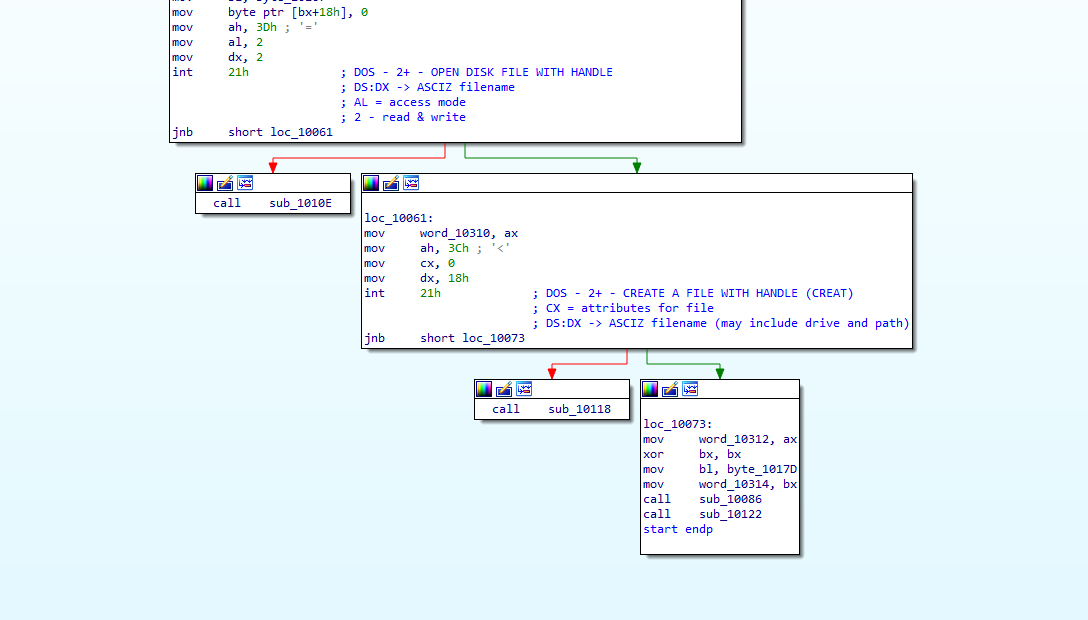


Рис.2.3 Представление кода в программе IDA Pro

При анализе функции start или основной функции необходимо обратить внимание, что:

1. Блоки разрываться и из них ведут по 2 стрелки.
2. До разрыва видно команду jnb что расшифровывается как прыгнуть если меньше.

Из этого следует, что в программе присутствует 2 ветвления, то есть условия (If else)

Далее обратив внимание на то, что и процедуры по адресу 1010Е (Рис. 2.4), 10118 (Рис. 2.5) (Команда call вызывает какую-либо процедуру) и основную программу start заканчивает одна и та же процедура, значит эта процедура завершения программы, так как необходимо закрыть файлы.

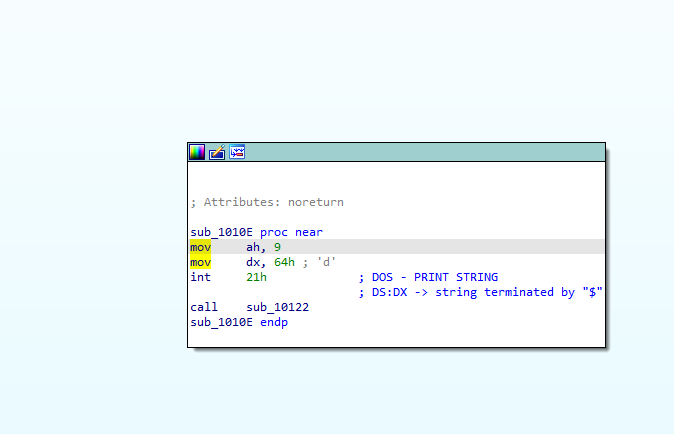


Рис.2.4 процедура 1010Е

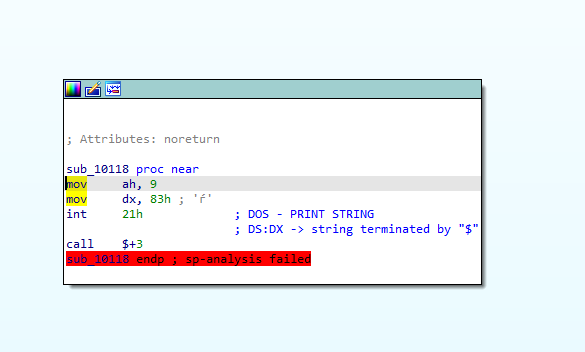


Рис.2.5 процедура 10118

Проанализировав эти две процедуры понятно, что в них вызывается прерывания DOS 21h и заноситься значение в регистр ah значение 9, а это функция вывода строки на экран, адрес строки заноситься в регистр dx (подробнее в пункте 1.1.4).

Из этого следует, что это вывод строки и зная адрес нахождения строки можем посмотреть, что именно выводят эти 2 функции

Опираясь на те строки что выводятся, а это строки: “error: cannot open source file$” и “error: cannot open out file$”, из этого следует что это процедуры вывода ошибок.

Тут вызывается 10118+3, так как знак $ указывает на текущую позицию

Что тоже ссылается на 10122

То есть 10122 идет как функция, завершающая программу (Рис 2.6) что подтверждает прошлое предположение по поводу назначения этой функции, а 2 функции, показанные до этого, то есть 10118h и 1010Eh, выводили отчёты, об ошибках, которые расположены в сегменте данных (dsegment) по адресу, указанному в регистре dx, что подтвердило выводы.

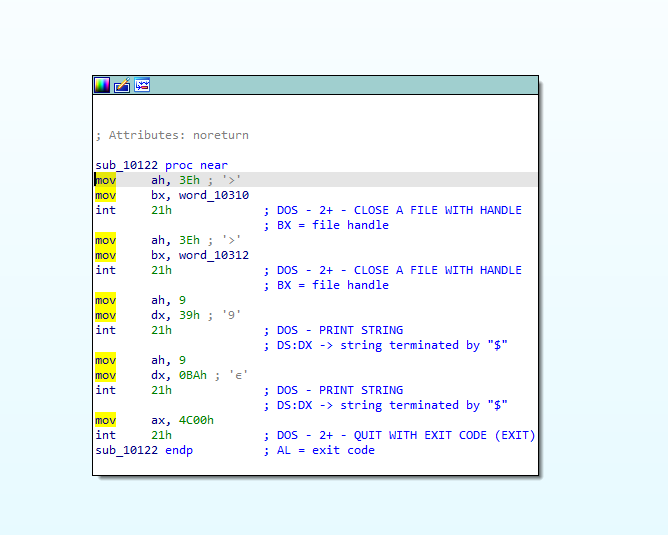


Рис.2.6 процедура завершения программы

При запуске программы будет получено 2 запроса - имя файла вывода и имя исходного файла (Рис 2.7)

Из этого был сделан вывод что те ветвления (условия if) - это проверка открытия файлов, так же на это указывает то что если файл открыть не получиться он выведет ошибку

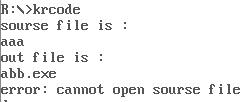


Рис.2.7 пример использования программы при неправильных входных данных

И после успешных проверок вызывается последняя не рассмотренная процедура

При рассмотрении последней функции с адресом 10090h необходимо обратить внимание на наличие цикла сложения (Рис 2.8)

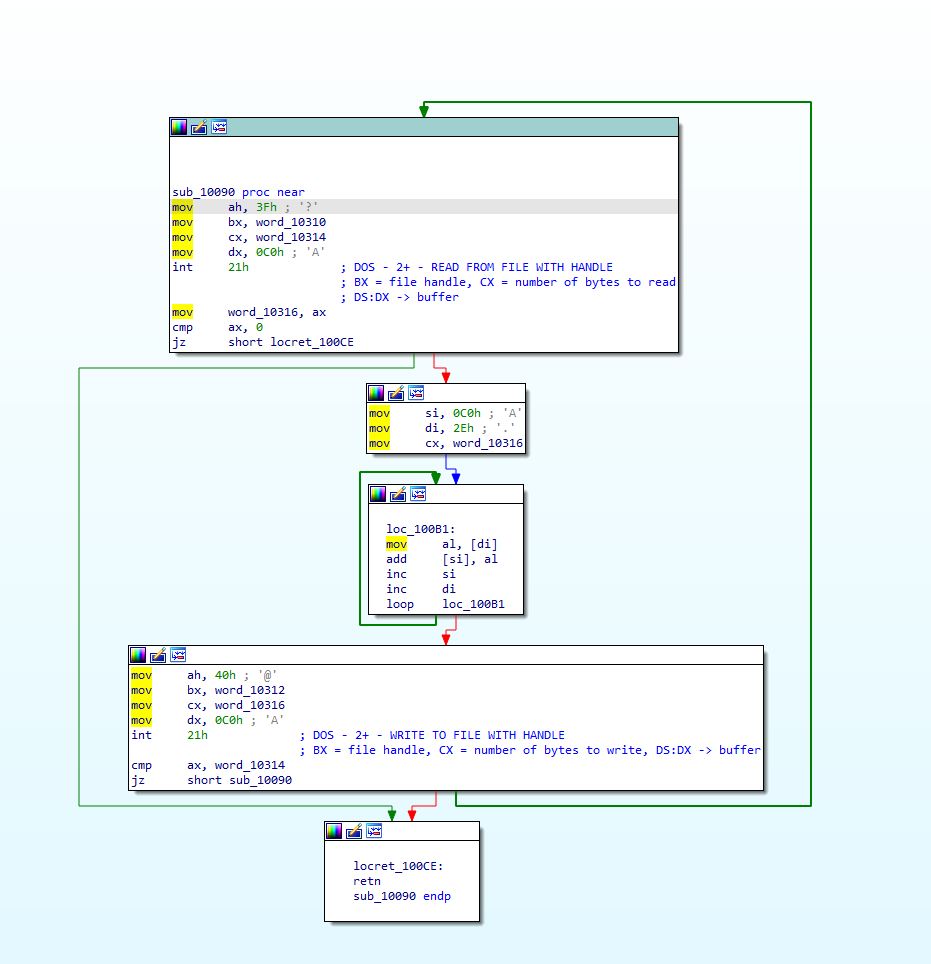


Рис.2.8 Процедура шифрования

Зная метод шифрования можно найти действие сложения в цикле, а в цикле используется регистр-счётчик cx для отслеживания количества итераций цикла.

Внутри найденного цикла, в регистре di, складываемом с другой переменной находится адрес 2Eh (Рис 2.8))

4

Рис.2.9 Ключ шифрования в сегменте данных

По этому адресу (Рис 2.9) находиться ключ шифрования «key is flag» обратим внимания на то что вся эта строка является ключом, а не только слово flag

Таким образом, была выявлена структура программы (Рис 2.10) и найден ключ (Рис 2.9) который необходим для расшифровки файлов.

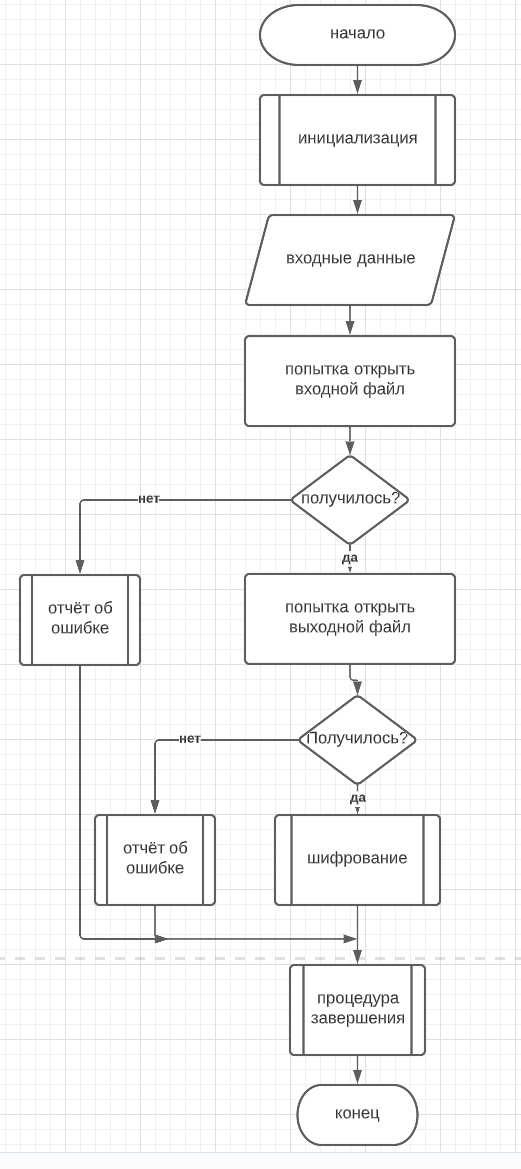


Рис.2.10 Схема алгоритма восстановленная

Необходимо определить переменные, которые будут использоваться в восстановленной программе:

-Имена файлов которые являются входными данными (хендлы файлов)

-Ключ

-Длина ключа

-используемые функции и процедуры

Используя данные полученные в результате восстановления и анализа необходимо составить восстановленную программу шифрования.

2.2. Программа на с++

*2.2.1 Задание*

Созданная программа принимает, в качестве исходных данных, название входного файла в формате \*\*\*.txt и выходного \*\*\*.txt. После этого пытается открыть исходный файл для чтения один, если это не удаётся выводиться сообщение об ошибке и программа прекращается. В ином случае программа пробует открыть второй файл для записи

Шифрование методом сложения с ключом состоит в том, чтобы складывать каждый i символ из исходного файла с i mod key\_length ключа, то есть происходит сложение каждой строки длины ключа из файла с ключом посимвольно

*2.2.2 Создание программы*

Использованные в программе переменные и их смысл описаны в таблице 2.

2.1Таблица данных

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| класс | Имя | Описание | Тип | Структура | Формат |
| Входные данные | fin\_name | Имя входного файла | Char\* | Динаический одномерный массив |  |
| fout\_name | Имя входного файла | Char\* | Динаический одномерный массив |  |
| Промежуточные данные | key\_length | Длинна ключа | int | Простая переменная |  |
| key | Ключ шифрование | Char\* | Простая переменная |  |
| key\_str | Ключ шифрование | string | Простая переменная |  |
| fout | Хендл выходного файла | ifstream |  |  |
| fin | Хендл входного файла | ofstream |  |  |

Были созданы 4 процедуры, те же что были найдены в исходной программе чтобы сохранить

Таблица 2.2

|  |  |  |
| --- | --- | --- |
| Название функции | Входные данные | Назначение процедуры |
| void err\_in\_proc | нет | ошибка во входном файле(не удалось открыть) |
| void err\_out\_proc | нет | ошибка в выходном файле(не удалось открыть) |
| void code\_proc | нет | процедура шифрования |
| void end\_proc | нет | процедура конца |

*2.2.3 Схема алгоритма*

Далее будут приведены пояснения создания алгоритма данной программы, которая будет представлена ниже на рисунке 2.10.

В любой программе есть инициализация (блок 1 на Рис.2.10) которая необходима, так как без первоначальной настройки программа не сможет работать правильно. В отличии от настройки регистров на работу в Ассемблер на языках высокого уровня регистры задаются автоматически, а мы можем в простых программах пропустить этот этап.

После этого нам необходимо получить имена файлов входного и выходного (блок, отмеченный цифрой 2 на Рис2.10), далее надо проверить оба файла на открытие, и, если один из них не открылся вывести отчёт об ошибке и завершить программу это является обработкой граничных или другими словами аномальные ситуации, которые необходимо обработать чтобы программа завершалась правильно, с сообщением об ошибке, а не крашилась останавливаясь операционной системой. Их небольшой анализ приведён в таблице 1.3.

Если оба файла открыты и готовы к шифрованию начинаем шифровать до конца входного файла. (блок 5 на Рис 2.10)

После окончания процедуры закрываем оба файла, сообщаем пользователю что всё готово и завершаем программу, это входит в процедуру завершения, в которой надо вывести отчёт что программа завершена и завершить программу, вызвав функцию MS-DOS аналог return 0 и языка программирования C++.

Схема алгоритма:

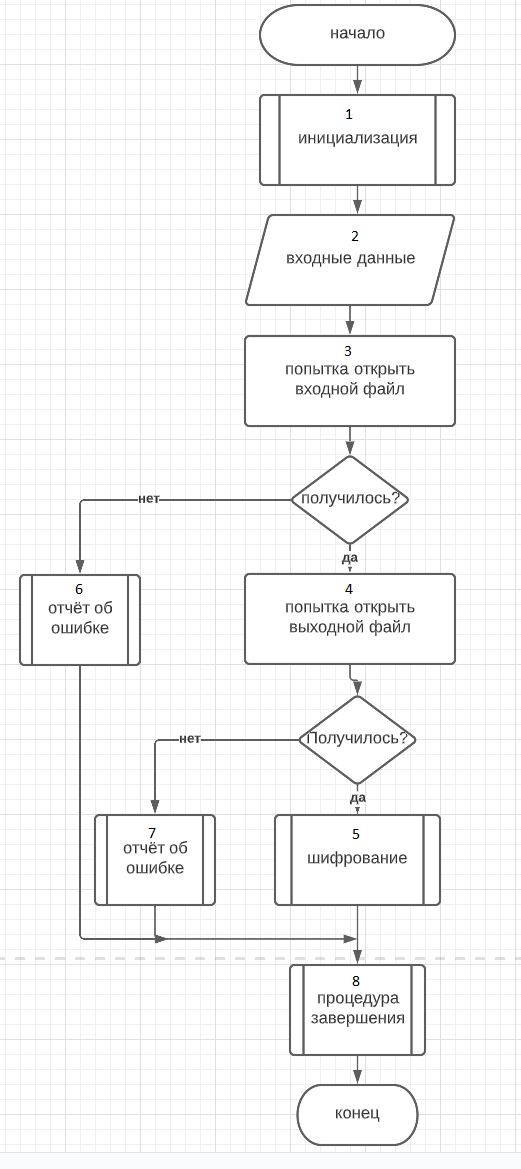


Рис.2.10 схема алгоритма программы

*2.2.4 Пользовательский интерфейс*

Программа работает как консольное приложение, поэтому весь интерфейс представляет собой консоль с выводящимися на экран сообщениями. Сначала приглашающими подать входные данные потом либо отчёт об ошибке, либо же сообщение об удачном завершении программы.

Первым сообщением будет выводиться сообщение

sourse file is :

это приглашение напечать название входного файла

далее после ввода возможно два варианта развития, либо выведется сообщение

error: cannot open sourse file

это будет значит, что не удалось открыть входной файл. За этим будет следовать прекращение программы.

Или

out file is :

Если пользователь видит данное сообщение — это значит, что удалось открыть входной файл и программа просит ввести имя выходного файла, далее опять возможно два варианта после ввода. Если пользователь видит следующее сообщение

error: cannot open out file

это значит, что не удалось открыть выходной файл и результат будет как при прошлом сообщении об ошибке. Если это сообщение не получено пользователем будет выведено сообщение

done

Это означает что приложение завершило работу и выводиться оно не взирая было ли выведено сообщение об ошибке или нет.

Далее будет приведён пример, когда пользователь ввел имя не существующего входного файла (Рис 2.10)

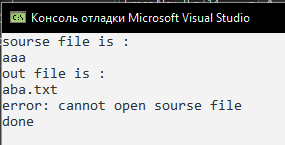


Рис.2.11 пример использования программы при неправильных входных данных

2.3 Результаты проведённых тестов и сравнений

Таблица 2.3 сравнительные тесты исходной и восстановленной программ

|  |  |  |  |
| --- | --- | --- | --- |
| Номер теста | Исходный текст | Assembler | C++ |
| 1 | 11111111111  1  1  1  1 | bbbbbbbbbbb>;b>;b>;b>;b | bbbbbbbbbbb>;b>;b>;b>;b>; |
| 2 | abc | МЗЬ | МЗЬ-s |
| 3 | 11111111111  1  1  1  1 | њ–ЄQљ¤Q—ќ’xoЄ-s¤-pќnqњ | њ–ЄQљ¤Q—ќ’xoЄ-s¤-pќnqњrѓ |
| 4 | i hate everything about you | Ф…бЃЭШ@ЛвЖЩдЩб‰ЧЪ@ЗОРЬЯ…тЏЮ | Ф…бЃЭШ@ЛвЖЩдЩб‰ЧЪ@ЗОРЬЯ…тЏЮЂ\* |

Для первого теста ключ шифрования был изменён на “11111111111” чтобы посмотреть при ключе без изменения символов.

И мы можем увидеть, что assembler при считывании считывает два дополнительный “невидимых” символа это символ номер 10 по таблице Win1251(Рис 2.12) и символ номер 13.



Рис.2.12 таблица кодов символов win-1251

По данной таблице не видно, что именно означают данные символы. Но это символы “\n” то есть переноса на новую строку символ номер 10 и символ номер 13 “\r” возврата каретки.

Так же проверим что по этой таблице код символа “1” то есть 49 складывается с кодом символа “1”- 49 и получается 49+49=98 что является кодом символа “b” что и показывает наш тест

Используемые в программе на с++ функции не считывают символы переноса строки и каретки, значит надо добавить их к каждой строке вручную. Из этого возникает единственное различие в том, что в конце файла появляются лишние символы, которые не повлияют на расшифровку.

Обратив внимание на тест номер два и посчитаем

Код символа

a+k=М то есть 97+107=204

b+e=З 98+101=199

c+y =Ь 99+121=220

Тест номер 3 из таблицы 2.3 был проведён на тех же входных данных, но на ключе «key is flag»

2.4 Программы декодирования

Так же для проверки были созданы две дополнительные процедуры декодирования, работающие почти так же, как и процедуры кодирования за некоторыми исключениями.

Опираясь на метод шифрования сложением с ключом, метод дешифровки будет как вычитание из зашифрованного материала ключа. Следовательно, было создано две функции при замене функций шифрования на которые программа преобразуются из шифрующих в дешифрующие.

Касательно программы на ассемблере действие сложения меняется на вычитание. Для программы на С++ же требуется кроме смены действия добавить условие пропуска символа переноса каретки, чтобы избежать двойного переноса строки. Две эти программы прописаны в приложении 3.

Таблица 2.4 тесты программ дешифрования

|  |  |  |  |
| --- | --- | --- | --- |
| Номер теста | Зашифрованный текст | Дешифровка ассемблер | Дешифровка С++ |
| 1 | bbbbbbbbbbb>;b>;b>;b>;b>; | 11111111111  1  1  1  1 | 11111111111  1  1  1  1 |
| 2 | bbbbbbbbbbb>;b>;b>;b>;b | 11111111111  1  1  1  1 | 11111111111  1  1  1  1 |
| 3 | Ф…бЃЭШ@ЛвЖЩдЩб‰ЧЪ@ЗОРЬЯ…тЏЮ | i hate everything about you | i hate everything about you |

**Заключение**

В результате выполнения научно-исследовательской работы на тему «Восстановление исходного кода и структуры программы по исполняемому файлу» была подобрана и изучена литература по принципу работы интерактивного дизассемблера, восстановление программ из языка ассемблера и исходного файла, а также по теме формирования команд в двоичном формате.

Результатом выполнением НИР стало описание процесса создания программ и анализа для восстановления исходной, так же две программы шифрования одна на языке ассемблер другая на языке С++.

Разработана, а затем ассемблирована исходная программа шифрования. Так же проведён анализ исполняющего файла и последующее восстановление данных, ключа шифрования и структуры программы. По этим данным создана программа шифрования на языке С++ имеющая ту же структуру что и исходная программа.

В результате был проведён сравнительный набор тестов показывающий идентичный результат. Они являются взаимозаменяемыми программами. Так же для тестирования были разработаны две программы декодирования, точнее функции при смене которых программа будет работать реверсивно.

Описанная технология, иллюстрированная примером восстановления программы, может быть использована как материал для углубленного изучения процесса дизассемблирования и восстановления программы, или как начало для более детального анализа и разработке собственного дизассемблера.

**ЛИТЕРАТУРА**

* 1. Оголюк А. А. Защита приложений от модификации: учебное пособие. – СПб: СПбГУ ИТМО, 2013. – 56 с.
  2. Реверс-инжиниринг для самых маленьких: взлом кейгена: сайт – URL: https://habr.com/ru/post/220245/ (дата обращения: 04.03.2022).– Режим доступа: для всех пользователей. – Текст: электронный.
  3. Избранное: ссылки по reverse engineering: сайт – URL: https://habr.com/ru/company/dsec/blog/334832/ (дата обращения: 04.03.2022).– Режим доступа: для всех пользователей. – Текст: электронный.
  4. Просмотр дизассемблированного кода в VisualStudio: сайт – URL: https://m-i-kuznetsov.livejournal.com/152857.html (дата обращения: 04.03.2022).– Режим доступа: для всех пользователей. – Текст: электронный.
  5. Долгова К.Н., Чернов А.В., Деревенец Ю. O. Методы и алгоритмы восстановления программ на языке ассемблера в программы на языке высокого уровня г. Москва, ИСП РАН УДК № 681.3.06
  6. Сайт создателей IDA PRO https://hex-rays.com/ida-pro/ (дата обращения: 21.05.2022) Режим доступа: для всех пользователей. – Текст: электронный.
  7. Юров В. И. Assembler. Учебник для вузов. 2-е изд. — СПб.: Питер, 2003. —637 с.: ил. ISBN 5-94723-581-1
  8. Far Manager https://www.farmanager.com/index.php?l=ru (дата обращения: 20.06.2022) Режим доступа: для всех пользователей. – Текст: электронный.