МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение   
высшего профессионального образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных Систем и Технологий

Кафедра «Информационные системы»

Дисциплина «Информационная безопасность»

**ОТЧЕТ**

**по лаборатороной работе № 2**

Тема Изучение криптографических алгоритмов

Выполнил студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /\_\_\_\_\_Кузнецов В. И.\_\_\_\_\_\_\_/

подпись инициалы, фамилия

Курс\_\_\_\_\_\_\_\_\_\_4\_\_\_\_\_\_\_\_\_\_\_\_\_ Группа\_\_\_\_\_\_\_ПИбд-41\_\_\_\_\_\_\_\_\_\_

Направление/специальность\_\_\_09.03.04. Программная инженерия\_\_\_\_\_\_\_\_\_\_\_\_

Руководитель Мытарев П.В.

должность, ученая степень, ученое звание

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

фамилия, имя, отчество

Дата сдачи:

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_201\_\_г.

Дата защиты:

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_201\_\_г.

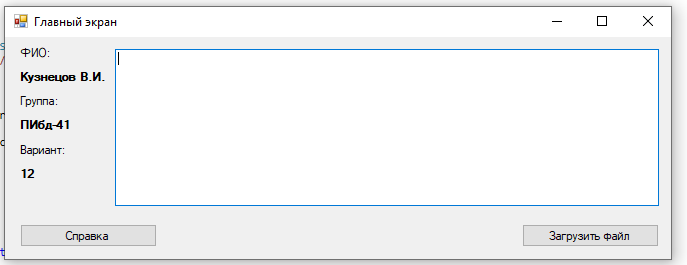
Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Ульяновск

2020 г.

Вариант 12: SHA-1

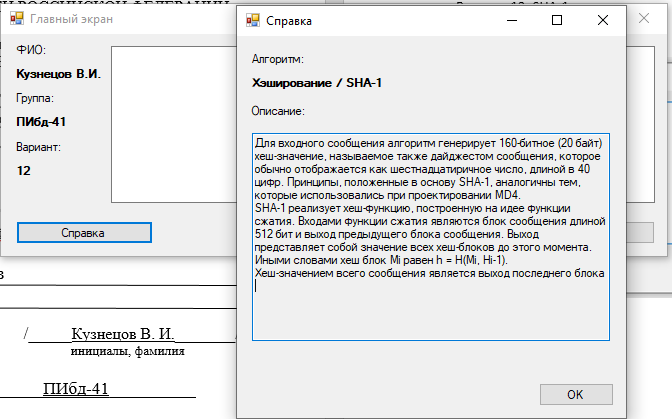
**Демонстрация работы:**

**

*Рис. 1*

Комментарий к рис.1

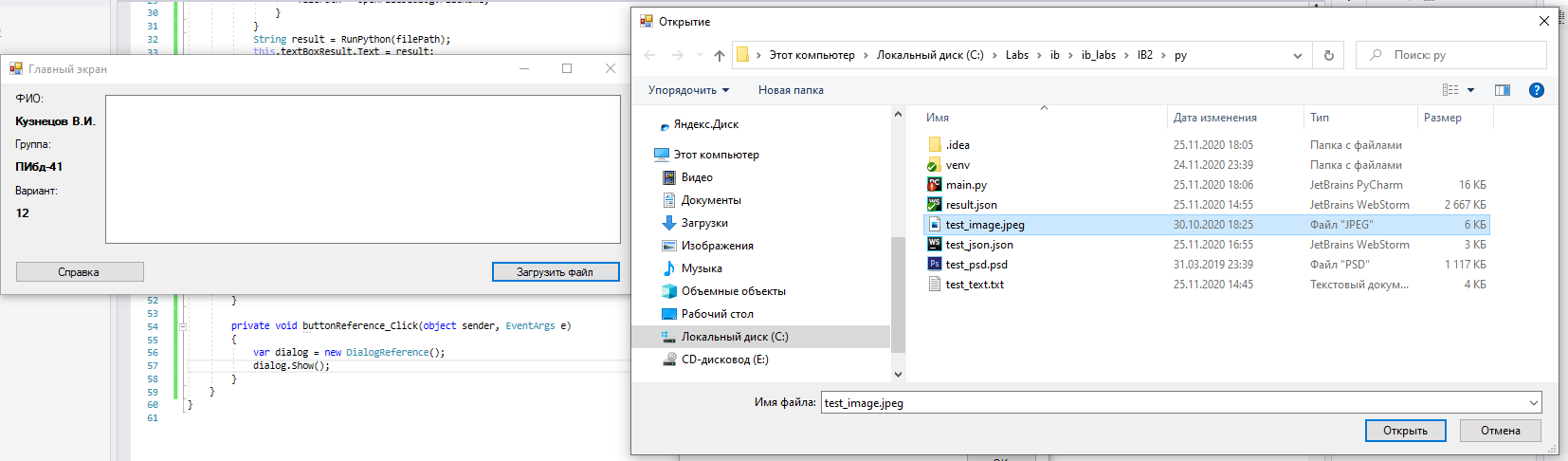
При запуске загружается экран главный экран приложения, с возможностью хэширования файла, просмотра небольшой справки и информации о студенте.

**

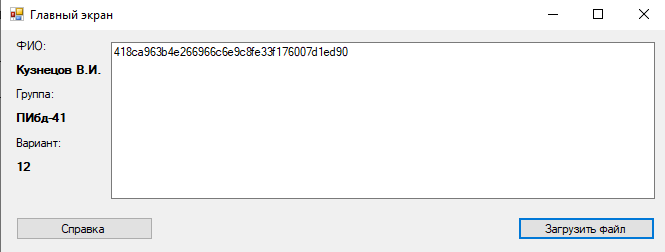
*Рис. 2*

Комментарий к рис.2

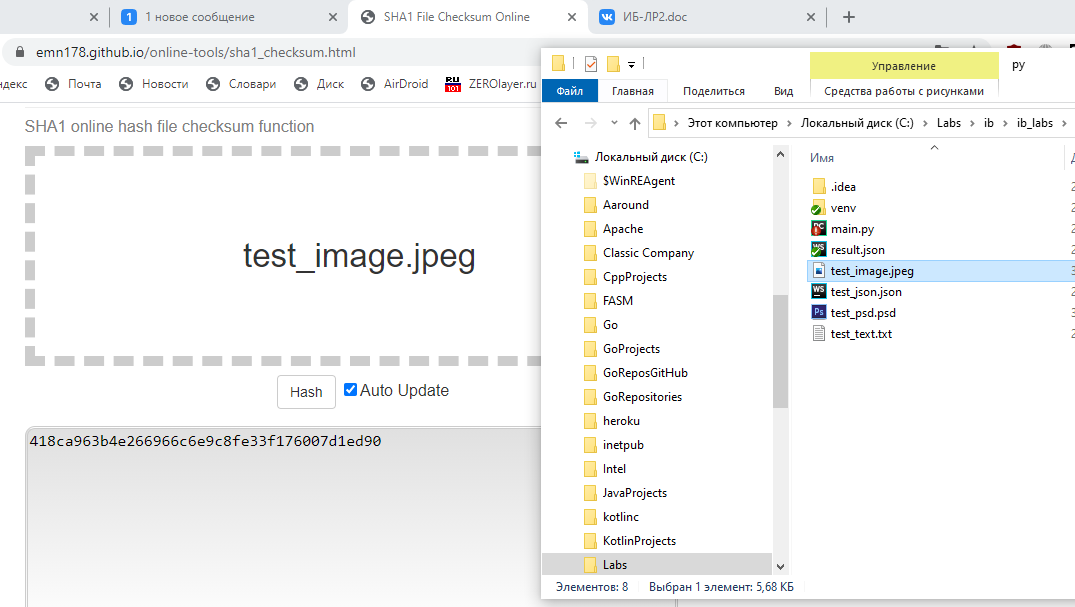
При открытии справки, выводится краткая выжимка о работе алгоритма SHA-1

**

*Рис. 3*

**

*Рис. 3.1*

**

*Рис. 3.2*

Комментарий к рис.3

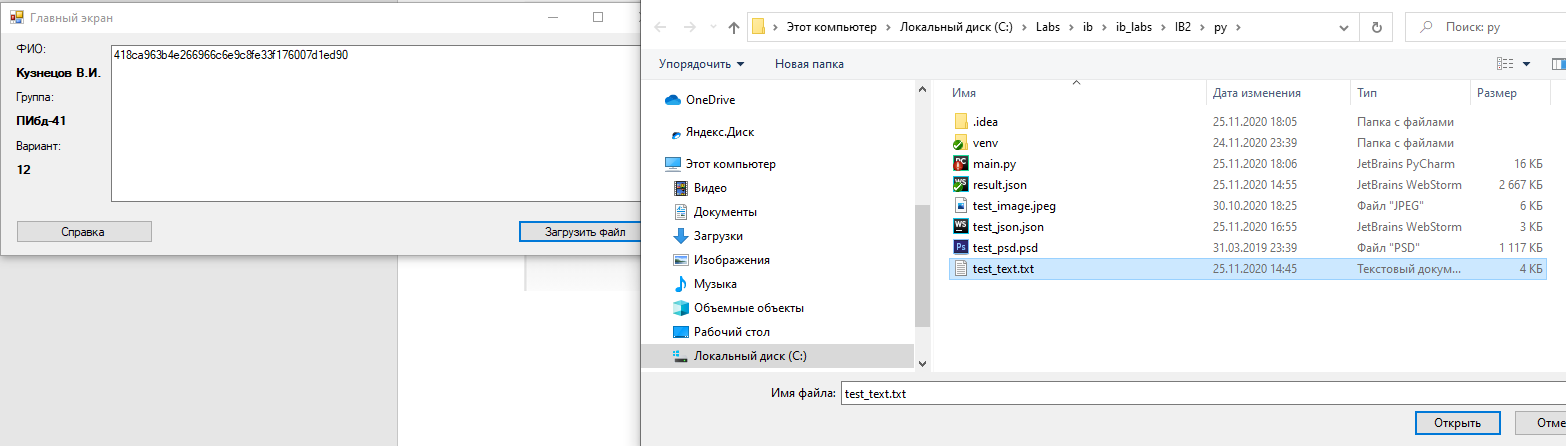
Выбираем файл, формата .jpeg (размер файла ~6 КБ.)

Комментарий к рис.3.1

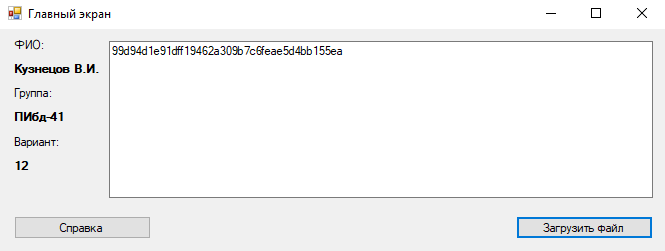
Получаем хэш-представление файла

Комментарий к рис.3.2

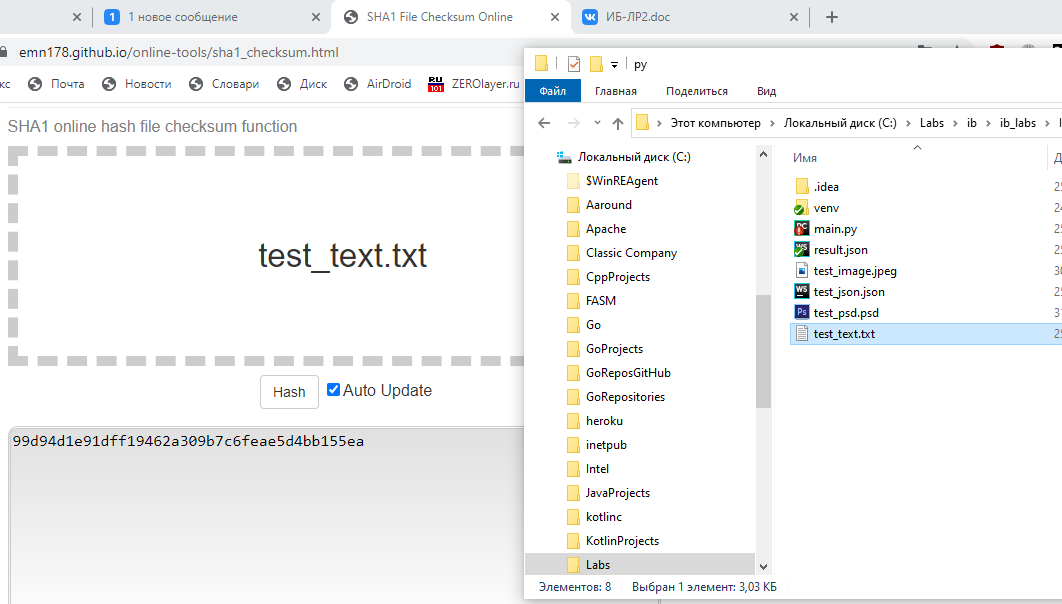
Проверяем правильность реализации, загрузив этот же файл на сайте.

**

*Рис. 4*

****

*Рис. 4.1*

**

Комментарий к рис.4

Выбираем файл, формата .txt (размер файла ~4 КБ.)

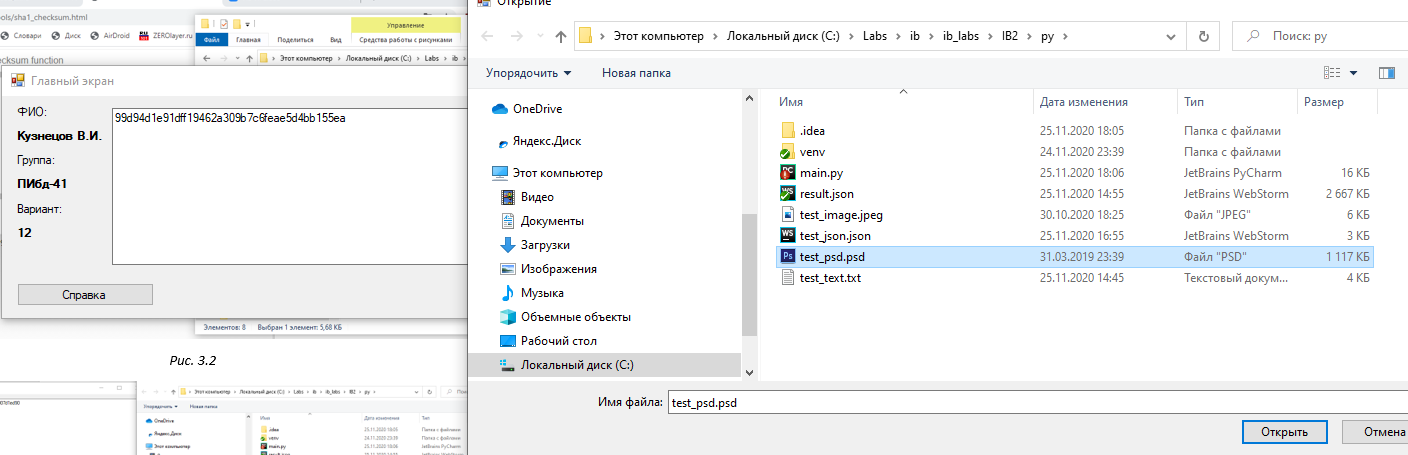
Комментарий к рис.4.1

Получаем хэш-представление файла

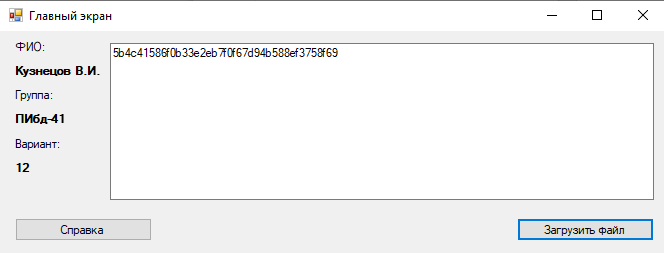
Комментарий к рис.4.2

Проверяем правильность реализации, загрузив этот же файл на сайте.

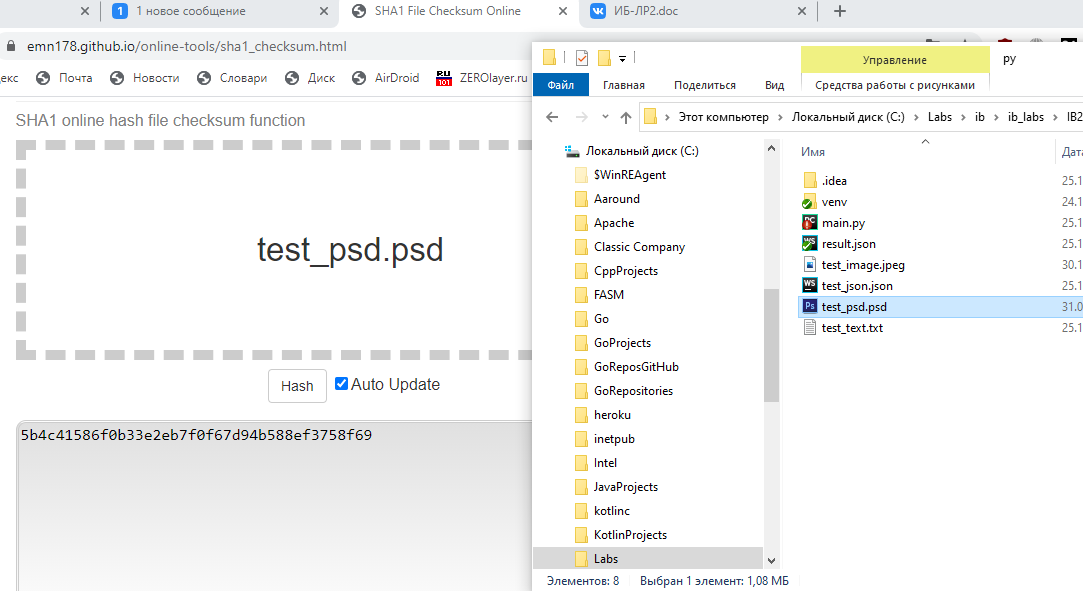
*Рис. 4.2*

**

*Рис. 5*

**

*Рис. 5.1*

**

*Рис. 5.2*

Комментарий к рис.5

Выбираем файл, формата .psd (размер файла ~1 МБ. (1 117 КБ.))

Комментарий к рис.5.1

Получаем хэш-представление файла

Комментарий к рис.5.2

Проверяем правильность реализации, загрузив этот же файл на сайте.

**Описание основного функционала.**

SHA-1 алгоритм реализован на яп Python, формочки реализованы на яп C#.

Питон для реализации выбран, потому что в основном яп с которым я работаю GO, у меня возникли проблемы строкового, битового представления, а на C# я программирую мало, поэтому взял питон, с которым у меня есть опыт работы.



*Рис.1*

Комментарий к рис.1

Запускаем диалоговое окно выбора файла, получаем его имя, и запускаем питон скрипт, передав ему имя файла, после получив дайжест содержимого файла, выводим на экран.



*Рис.2*

Комментарий к рис.2

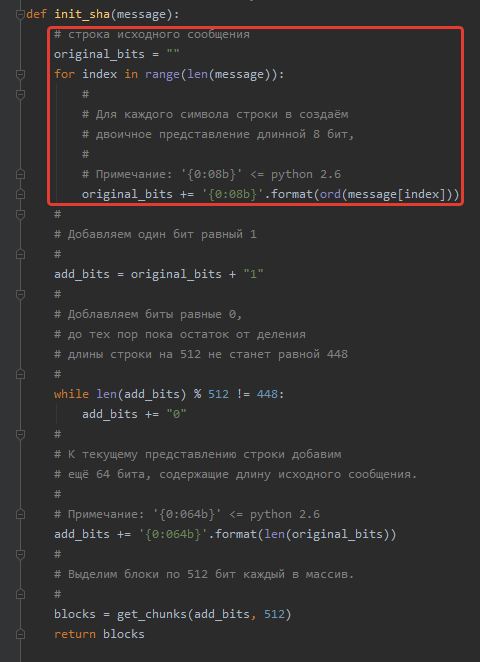
Скрипт на питоне читает байты из файла, переводит содержимо в хэкс представление из которого получает строковое представление (в обход прямого декодирования байт в определенную кодировку, с которым возникают проблемы при открытии разного формата файлов). Далее запускает функцию init\_sha() и hash\_sum(). Подробнее о работе ф-ий в описании алгоритма.

**Описание алгоритма.**

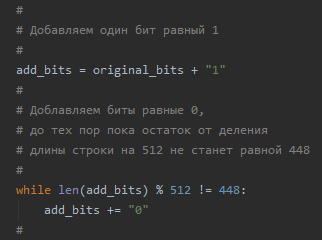
Принцип работы заключается в сжатии исходного сообщения. Хэш представлением всегда будет выступать 160 битный дайжест, состоящий из пяти 32 битных буфферов имеющих на этапе инициализации, определенные им значения, после чего будет выполнено 80 раундов преобразования этих буфферов с помощью содержимого исходного сообщения и логических операций над ними.

*Этап 1*

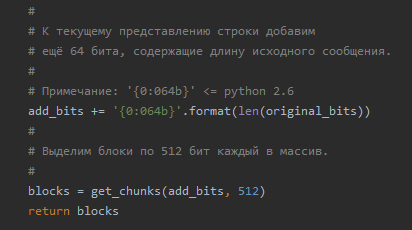
Исходное сообщение делится на блоки по 512 бит, после чего последний буффер будет расширен до длины, которая будет давать остаток 448 при делении на 512. Расширение выполняется путем добавления одного бита равного 1 и добавления битов равных 0 до тех пор, пока условие не выполнится. После чего к последнему блоку добавляется хвост длиной 64 бита, который содержит длину исходного сообщения.



*Рис 1.1*

**

*Рис 1.2*

**

*Рис 1.3*

Комментарий к рис.1.1

Исходное сообщение представляется в виде битовой строки

Комментарий к рис.1.2

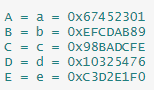
Добавление одного бита равного 1 и добавление битов равных 0, до тех пор, пока не выполнится условие

Комментарий к рис.1.3

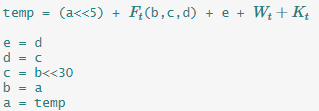
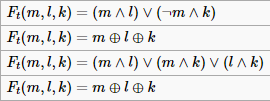
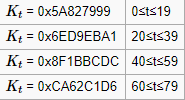
Добавление блока длиной 64 бита, содержащего длину исходного сообщения. Дробление полученной строки на блоки по 512 бит каждый.

*Этап 2*

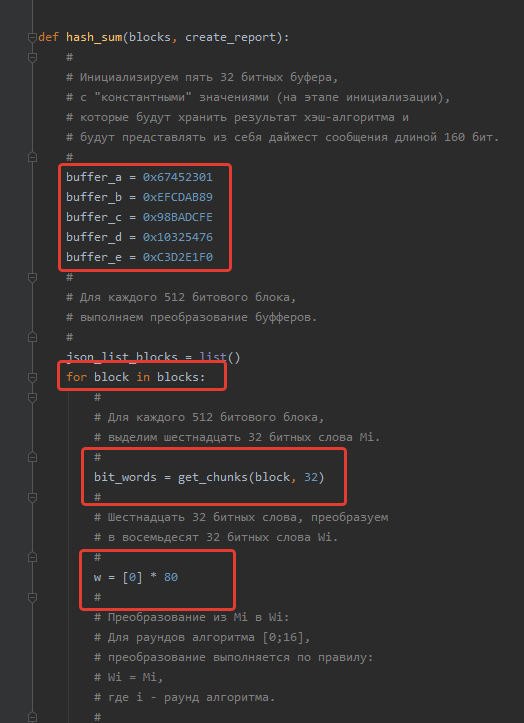
Определяем 5 буферов длиной по 32 бита.



Циклом проходим по каждому 512 битному блоку и выполняем следующие действия на каждой итерации:

* Преобразуем исходные шестнадцать 32 битных слова (M) из блока в восемьдесят 32 битных слова (W) по следующим правилам:
  + Для раундов 0 … 16: Wi = Mi
  + Для раундов 16 … 79: Wi = (Wi-3 xor Wi-8 xor Wi-14 xor Wi-16) <<< 1
* Копируем во временные буферы текущее состояние основных буферов
* Далее выполняем 80 раундов преобразования временных буферов по следующим правилам:
  + 
  + Новое состояние временного буфера А определяется суммой циклического сдвига текущего состояния влево на 5, функцией преобразования, состояния буфера Е, 32 битного слова W и константой K
  + Функция преобразования выглядит как система по условию:
    -  
    - Где M – временный буфер B, L - временный буфер C, K - временный буфер D, а t – текущий раунд
  + Константа K, зависит от текущего раунда
    - 
* После выполнения 80 раундов преобразований временных буферов, выполняем преобразования основных буферов путём сложения текущего состояния основных буферов с временными.

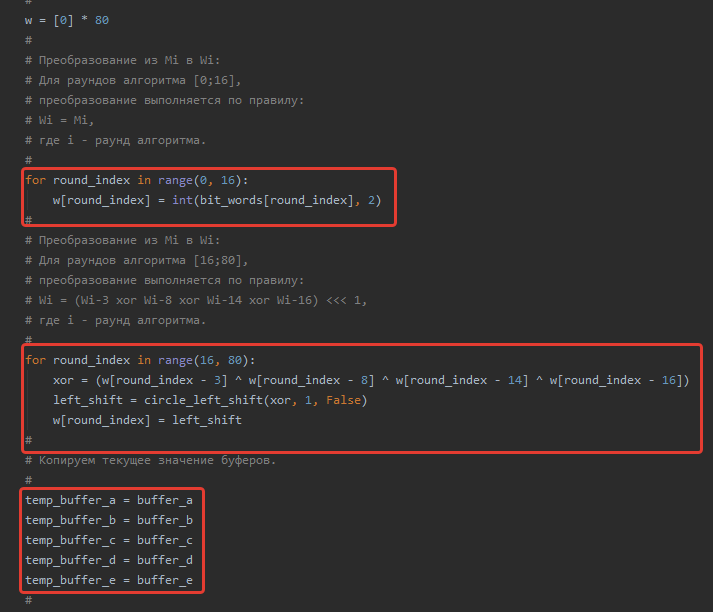
После обхода каждого 512 битного блока получим преобразованные значения основных буферов, которые и будут являться дайджестом или хэшем исходного сообщения.



*Рис 2.1*

Комментарий к рис.2.1

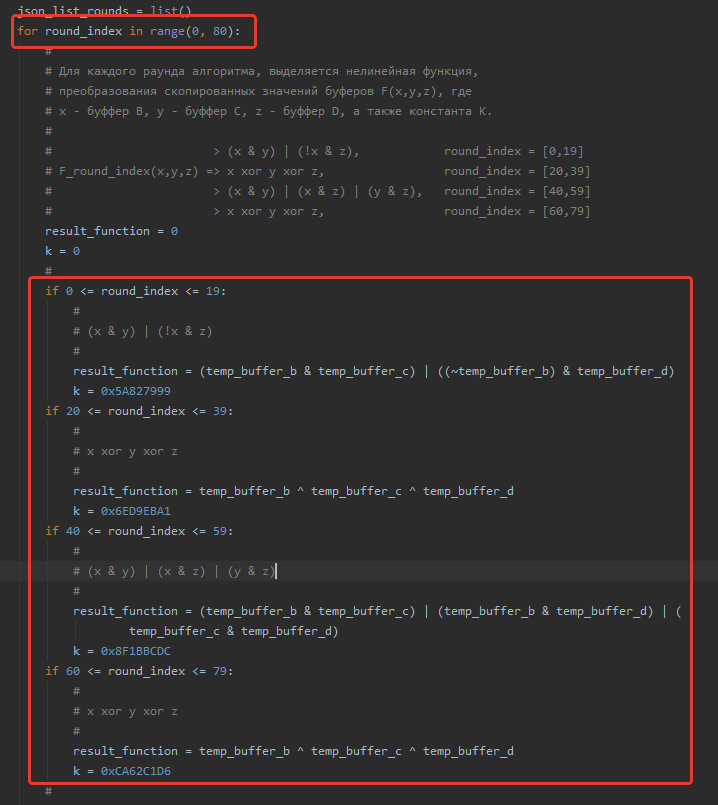
Выделяем пять 32 битных основных буфера, начинаем итерироваться по каждому 512 битному блоку, выделяя шестнадцать 32 битных слов из блока и восемьдесят 32 битных слова.



*Рис 2.2*

Комментарий к рис.2.2

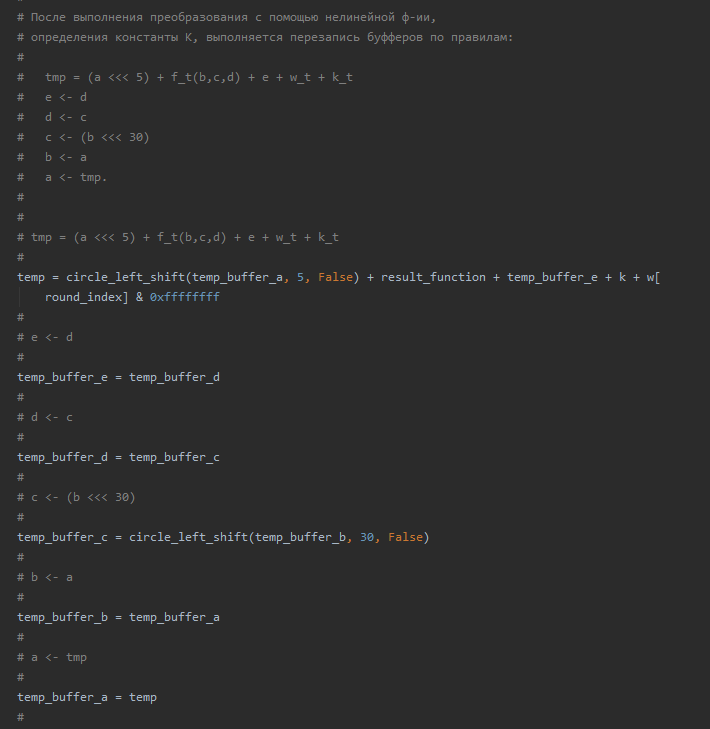
Заполняем восемьдесят 32 битных слова по установленным правилам. Копируем текущее состояние основных буферов во временные.



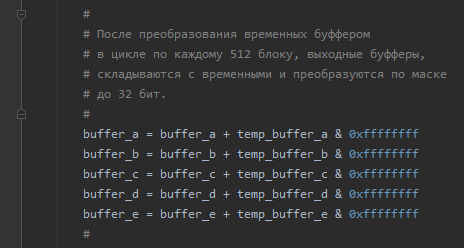
*Рис 2.3*

Комментарий к рис.2.3

Выполнение основного цикла из 80 раундов, с преобразованиями временных буферов по установленным правилами



*Рис 2.4*

**

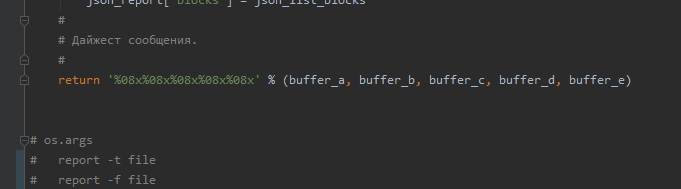
*Рис 2.5*

Комментарий к рис.2.4

Перезапись значений временных буферов на каждом раунде.

Комментарий к рис.2.5

Перезапись значений основных буферов в конце итерации цикла, обходов 512 битных блоков.

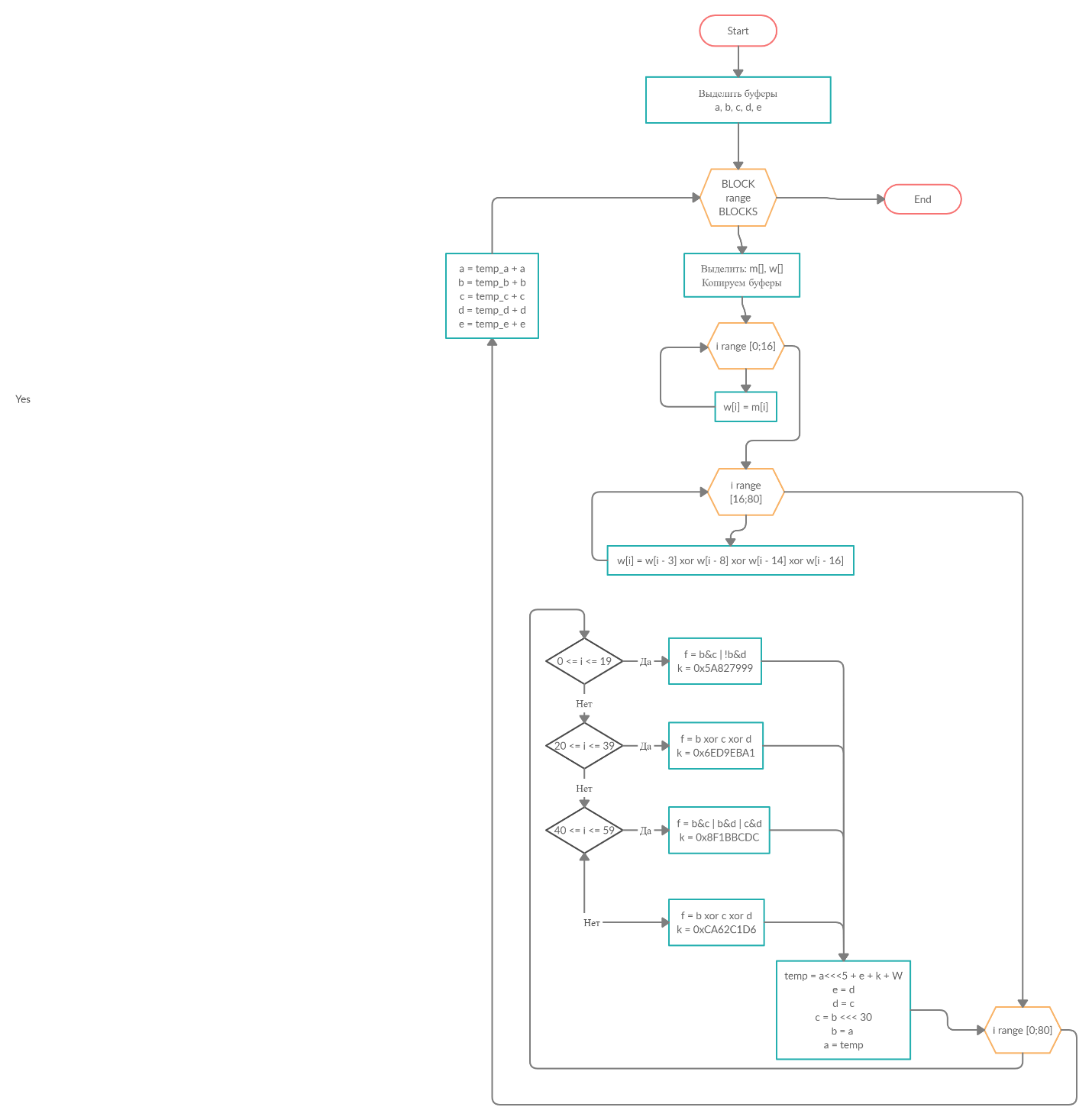


*Рис 2.6*

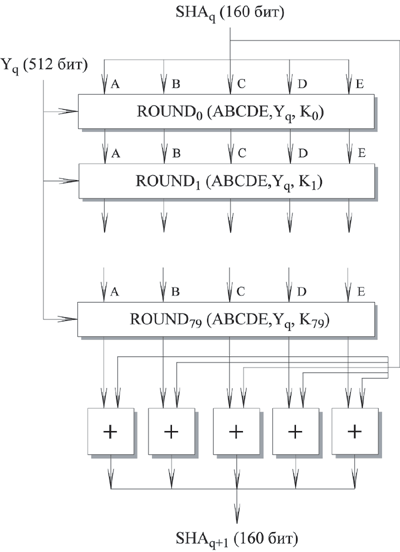
Комментарий к рис.2.6

Дайджест или хэш представление исходного сообщения.

**Блок схема алгоритма:**



*Блок-схема*



*Общая схема работы.*

**Раунды на примере хэширования строки “Hello world”:**

Результаты записаны в файл result.json, в папке py в проекте <https://github.com/RobertGumpert/ib_labs/tree/main/IB2/py>

**Листинг:**

<https://github.com/RobertGumpert/ib_labs/tree/main/IB2>