ОТЧЕТ

ХЭШ СТРИБОГ

# Общее описание алгоритма

Описание алгоритма хэшировония находится в стандарте ГОСТ 34.11-2012.

Длина выходного блока может быть 256 или 512 бит. Сами по себе эти функции идентичны, а изменения только в начальном заполнении переменных. На вход алгоритм принимает блок, длина которого 512 бит.

Прежде, чем рассматривать сам алгоритм хэширования, необходимо рассмотреть преобразования, которые там используются. В алгоритме есть как и стандартный набор – Sbox и перестановки, так и умножение вектора на матрицу.

# Описание криптографических свойств алгоритма и принципов его построения

В ходе вычисления хэша используются следующие преобразования –

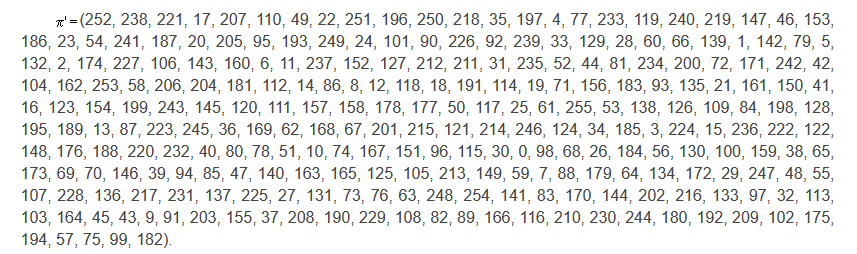
1. X-преобразование. На вход к X подаются две последовательности 512 бит. Результат – побитовое сложение по модулю 2 этих последовательностей;



1. S-преобразование. Это стандартная функция подстановки, каждый 8 битный блок заменяется на соответствующим байтом из таблицы подстановок;

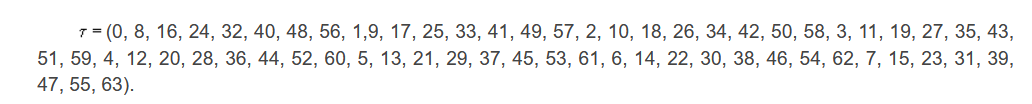


Значение подстановки –



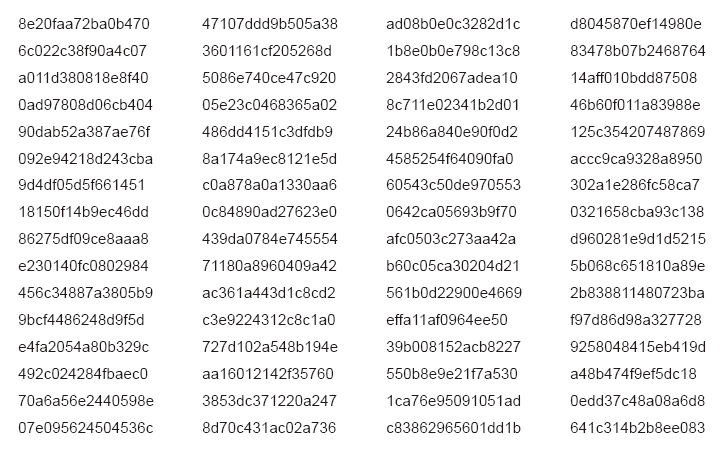
1. P-преобразование. Является функцией перестановки. Для каждой пары байт из выходной последовательности происходит замена одного байта другим;



Значение перестановки –

1. L-преобразование. Представляет собой умножение 64-битного вектора на квадратную матрицу A с размером 64x64.



Значение матрицы –Путья

После этого небольшого экскурса можно приступать к основной функции – функции сжатия g(). Это основной элемент любой хэш функции.

Пусть h, N, m – 512 битные последовательности. Для вычисления g(N, m, h) выполняем следующие шаги:

1. Вычисляем значение K = h ⊕ N;
2. Присваиваем K = S(K);
3. Присваиваем K = P(K);
4. Присваиваем K = L(K);
5. Вычисляем T = E(K, m);
6. Присваиваем t = h ⊕ t;
7. Вычисляем G = t ⊕ m;
8. Вернуть G как результат вычисления функции g(N, m, h)

В этих шагах есть функция, которая ранее не рассматривалась – E(K, m). Она выполняет следующие действия –

1. Вычисляем значение state = K ⊕ m;
2. В цикле от 0 до 11 выполняется:  
   state = S(state);  
   state = P(state);  
   state = L(state);  
   K = KeySchedule(K, i);  
   state = state ⊕ K;
3. Вернуть state.

Здесь снова неизвестная функция – KeySchedule(K, i). Она отвечает за формирование временного ключа K на каждом раунде функции E(K, m). Алгоритм у нее следующий –

1. K = K ⊕ C[i];
2. K = S(K);
3. K = P(K);
4. K = L(K);
5. Вернуть K.

C – это набор 512 битных констант.



Начальное заполнение вектора h, при длине выходного блока 512 бит, - 0x0064, для выходного блока 256 бит – 0x0164

# Длина блока, количество раундов

Описано выше

# Известные оценки стойкости к поиску коллизий по открытым источникам

Криптографическая стойкость данного алгоритма была описана в 2013 году в статье «Rebound attack on Stribog»

Rebound атака – это атака, которая применяется для поиска коллизий для алгоритмов хэширования и блочных шифров.

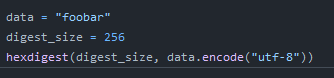
Если описывать коротко, то у ГОСТ Р и Whirlpool очень похожи и к ним обоим можно применять данную атаку. Но в нашей реализации есть матричная перестановка и она создает серьезную уязвимость.

# Общее описание программной реализации

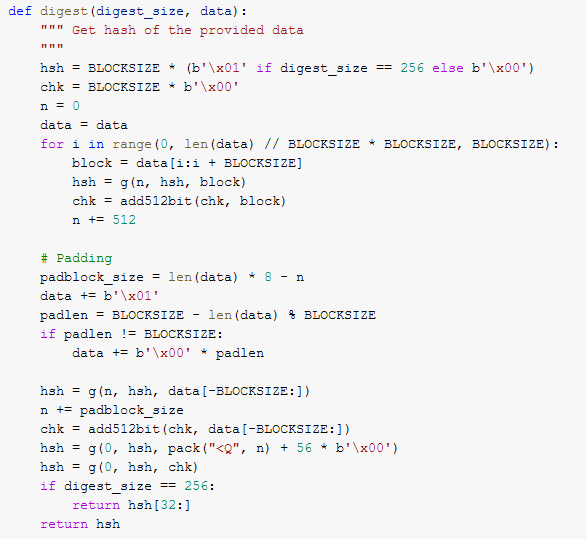
Реализация представлена в отдельном файле, в данном репозитории.

Программа выполнена на языке Python, среда выполнения – Google colab.

Сперва указываем длину выходного хэша – 256 или 512. Затем вводим данные, которые собираемся хэшровать, и эти два параметра передаем на вход функции.

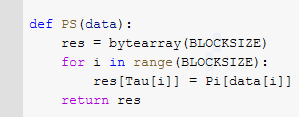


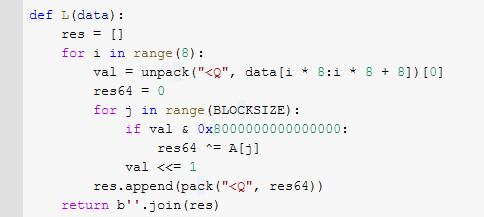
Вот так выглядит сама функция вычисления хэша. Сперва тут определяется начальное заполнение, в зависимости от выходной длины, затем данные разбиваются на блоки и происходит хэширование.



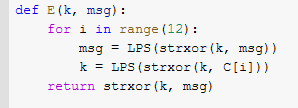
L, P, S преобразования представлены в виде одной функции, так как по алгоритму приходится эти преобразования выполнять одновременно.



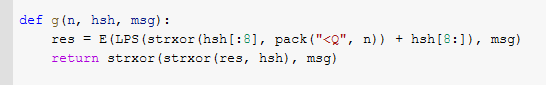




Функции E и KeySchedule тоже объединены в одну –

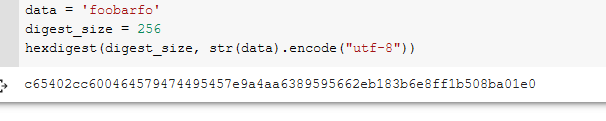


Функция g –

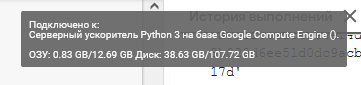


# Скорость выполнения тестового задания

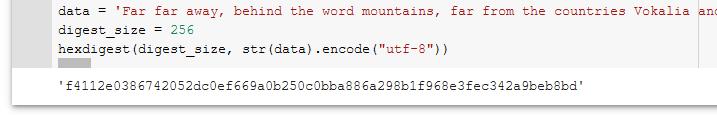
1 блок –



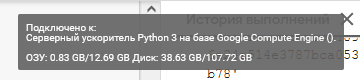




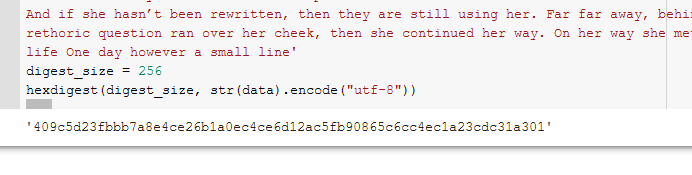
103 блоков



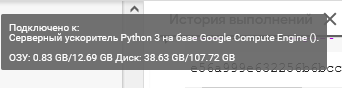




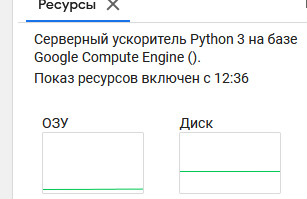
106 блоков





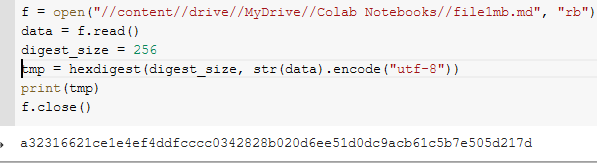


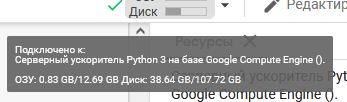
Ресурсы всегда были такими –



В дальнейшем используем просто пустые файлы, которые имеют соответствующий размер. Это будет достаточно долгий процесс, так как colab не будет автоматически расширять мощности при очень долгой их нагрузке.

Файл размером 1 мб –







Файл размером 100 мб –

А здесь происходит то, что я описывал в прошлой работе. Достаточно долго файлы с диска обрабатываются данной функцией.

То есть нужно около 5 часов времени для получения хэша из 100 мб файла.

# Описание тестового стенда

Описание выше

# Фиксация нагрузки на ОЗУ и ЦП во время выполнения тестового задания

Представлена выше

# Проверка встречной работы с таким же алгоритмом реализованным другим автором. Приоритетно выбирать алгоритмы, реализованные другими студентами этого курса

Сверить не имеется возможности