ОТЧЕТ

ЭЛЕКТРОННАЯ ПОДПИСЬ

FULL DOMAIN HASH

# Общее описание алгоритма

Обычно, схема электронной подписи следующая – данные загоняются в хэш-функцию, тем самым получается строка фиксированной длины. Генерируют 2 пары ключей – открытый и закрытый, и с помощью закрытого ключа, который доступен только владельцу подписи, зашифровывают ранее полученный хэш. Затем добавляют полученный результат к данным.

Так выглядит алгоритм получения электронной подписи. на выходе мы имеем документ и подпись к нему. Осталось только проверить верна ли подпись или есть ошибка.

Для этого нужно отделить данные и взять от них через ту же хэш-функцию хэш, затем открытым ключом, который доступен всем, расшифровать подпись. Если хэш и расшифровка подписи совпали, то подпись верна и документ не был модифицирован.

Это был общий алгоритм получения электронной подписи. В моей работе фигурирует алгоритм электронной подписи Full Domain Hash – FDH. Принцип его работы почти идентичен тому, что было описано выше.

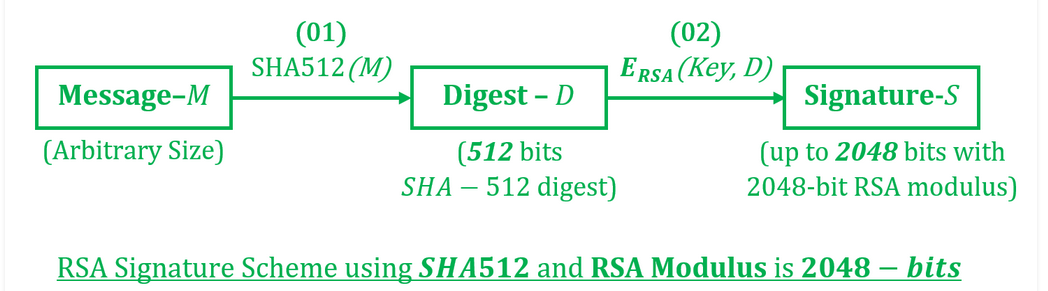
Вопросы остаются в части того, какую взять хэш функцию и как сгенерировать пару ключей.

# Описание криптографических свойств алгоритма и принципа его построения

В качестве хэш-функции берем SHA512, а для ключей используем RSA.

Так как в RSA, для достижения большего уровня защиты, необходимо брать большие простые числа, то для тестового стенда стоит ограничить поиск простых чисел до 216 или 65536.

После алгоритма RSA остается пары (e, n) и (d, n) – открытый и закрытый ключ, соответственно. Для шифрования надо возвести в степень – e и взять по модулю n, а для расшифрования надо возвести в степень d и взять по модулю n.



# Рекомендуемые параметры ключей

Ключи будем генерировать через RSA, ограничим максимальную длину – 216

# Количество раундов (шагов, циклов)

В самом алгоритме только возведение хэша в степень для шифрования и дешифрования. Все шаги и раунды находятся в хэш функции.

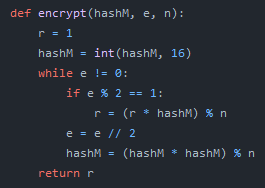
# Известные оценки стойкости по открытым источникам

Все зависит от того – какую хэш функцию использовать и насколько большие простые числа генерировать в RSA

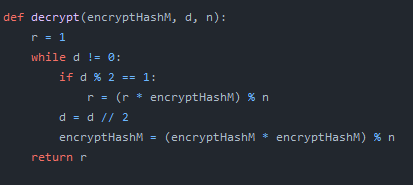
# Тестирование и описание программной реализации

# Общее описание программной реализации алгоритма

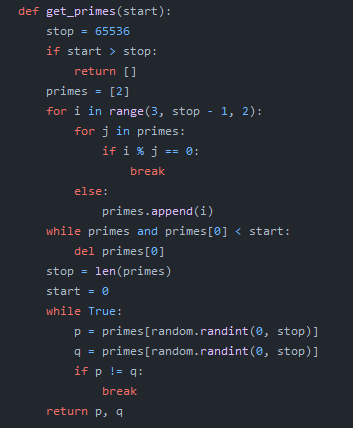
Функция шифрования представлена через алгоритм быстрого возведения в степень –



Как и функция дешифрования –

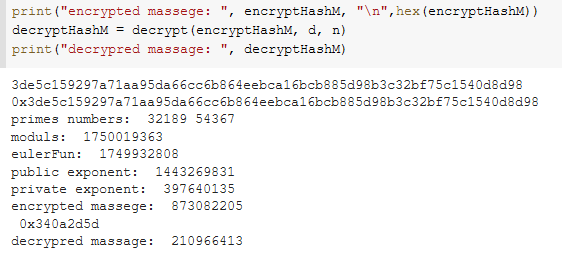


Это получение простых чисел для RSA –

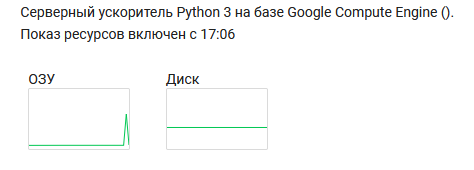


# Скорость выполнения тестового задания

1 блок –

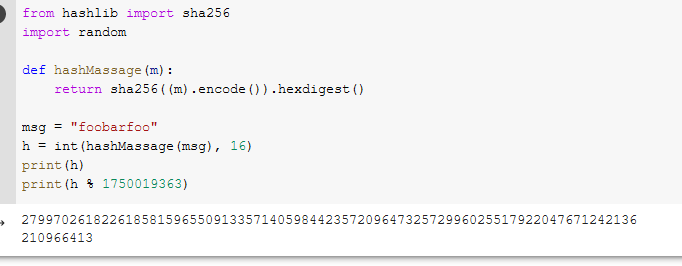






Данный скачок на ОЗУ вызван тем, что параллельно считается хэш на 100мб файле

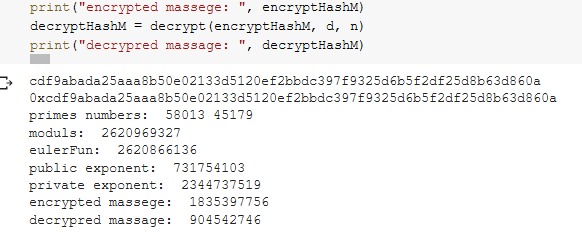
Проверка



103 блоков –

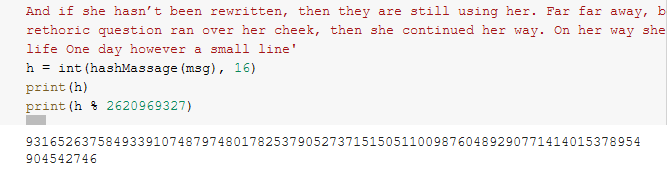


106 блоков –

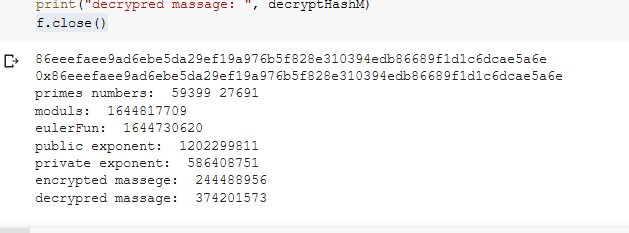




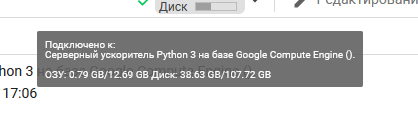
Проверка –



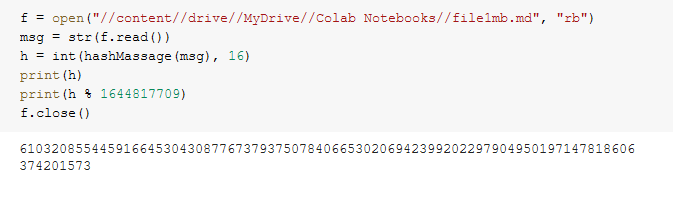
Файл 1мб –



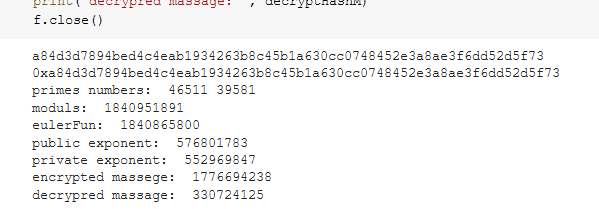




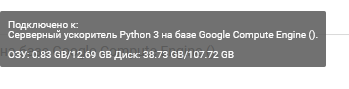
Проверка –



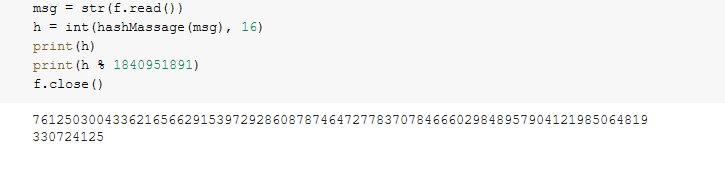
Файл 100 мб –



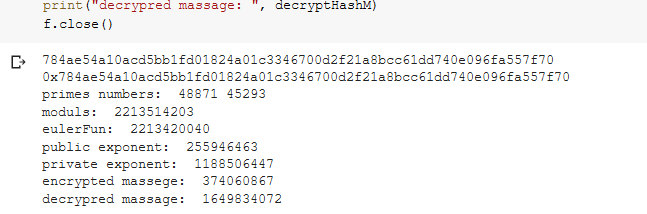


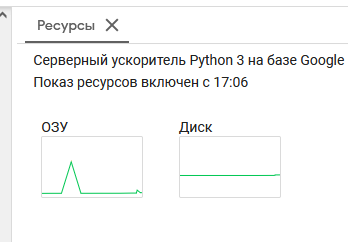


Проверка –



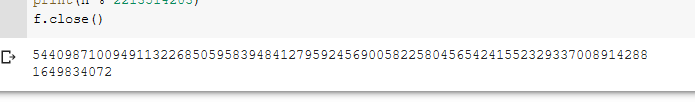
Файл 1гб –







Проверка –



# Описание тестового стенда

Тестовый стенд такой же, как и в предыдущих отчетах

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

Представлена выше

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

Не имеется возможность проверить