Правительство Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

(НИУ ВШЭ)

Московский институт электроники и математики им. А.Н. Тихонова

ОТЧЕТ

О ПРАКТИЧЕСКОЙ РАБОТЕ № 3

по дисциплине «Криптографические методы защиты информации»

ТЕМА РАБОТЫ

***Криптосистемы с открытым ключом***

|  |
| --- |
| Студент гр. МКБ231  Е.В. Шараев  «26» апреля 2024 г. |
| Руководитель  Заведующий кафедрой информационной безопасности киберфизических систем  канд. техн. наук, доцент  \_\_\_\_\_\_\_\_\_\_\_\_\_\_О.О. Евсютин  «\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2024 г. |

Москва 2024

**СОДЕРЖАНИЕ**

[1 Задание на практическую работу 3](#_Toc155608317)

[2 Краткая теоретическая часть 4](#_Toc155608318)

3[Описание программной реализации 5](#_Toc155608319)

3.1 Описание криптосистемы RSA……..………………………………………….………5

3.2 Генератор ключевой пары …….………………………………….…………………....6

3.3 Зашифрование и расшифрование ……………………………………………...……... 6

4Пример ручного шифрования ………………………………………………………………...

5 Демонстрация работы программы………………….…………………………..…………...10

6[Криптоанализ](#_Toc155608320) 10

7 Выводы …………………………………………………………………………………………

8 [Список использованных источников](#_Toc155608321) 21

# 1 Задание на практическую работу

Целью данной работы является приобретение навыков программной реализации криптосистем с открытым ключом.

В рамках практической работы необходимо выполнить следующее:

1. написать программную реализацию одной из следующих асимметричных криптосистем (по выбору студента) с использованием больших чисел:

-**RSA (выбранный вариант)**;

- Рабина;

- Эль-Гамаля;

1. изучить методы криптоанализа выбранной криптосистемы;
2. реализовать (вручную или программно) не менее одной атаки на выбранную криптосистему, исключая наивную переборную атаку, для случая, когда параметры криптосистемы не являются большими числами;
3. подготовить отчет о выполнении работы.

Программа должна обладать следующей функциональностью:

* принимать на вход файл, содержащий открытый текст, подлежащий зашифрованию, или шифртекст, подлежащий расшифрованию;
* принимать на вход ключевую пару (открытый ключ, закрытый ключ);
* давать пользователю возможность сгенерировать ключевую пару;
* осуществлять зашифрование или расшифрование введенного текста по выбору пользователя.

Отчет должен содержать следующие составные части:

* раздел с заданием;
* раздел с краткой теоретической частью;
* раздел с двумя-тремя примерами «ручного» шифрования для произвольных последовательностей символов;
* раздел с результатами работы программы для тех же последовательностей символов, что и в предыдущем разделе;
* раздел с подробным описанием реализованной атаки на криптосистему с приведением численных результатов;
* раздел с выводами о проделанной работе.

# **2 Краткая теоретическая часть**

Алгоритм RSA (Rivest-Shamir-Adleman) - это ассиметричный криптографический алгоритм, используемый для шифрования данных. Его основой является сложность факторизации больших целых чисел. Суть данного алгоритма (как и прочих ассиметричный алгоритмов шифрования) заключается в том что для зашифрования и расшифрования используются разные ключи. Один из них — **открытый ключ** (public key), выдается прочим пользователям для шифрования данных, другой - **секретный ключ** (secret key), держится в тайне и используется для расшифрования данных зашифрованых открытым ключом. Оба ключа составляют ключевую пару, генерируются одновременно и работают только друг с другом. Таким образом решается проблема управления ключами и его компрометации. Основу криптосистемы RSA составляем алгоритм генерации ключевой пары.

#### **3. Описание программной реализации**

Все файлы данной практической работы я опубликовал в своем гитхаб репозиторииhttps://github.com/Djoongaar/rsa

Программная реализация скрипта генерации ключевой пары находится в файле https://github.com/Djoongaar/rsa/blob/master/keygen.py

Программная реализация алгоритма зашифрования и расшифрования в файле https://github.com/Djoongaar/rsa/blob/master/rsa.py

Максимально подробные комментарии к коду я постарался оставить в самих блоках кода, а здесь лишь описание основных методов класса.

Демонстрационный юпитер ноутбук (пдф файл этого ноутбука приложен к данному отчету) https://github.com/Djoongaar/rsa/blob/master/demo.ipynb

Демонстрация попытки взлома ключей RSA приведена в файле https://github.com/Djoongaar/rsa/blob/master/analyze.ipynb

Демонстрация ручного шифрования по алгоритму RSA в юпитер ноутбуке https://github.com/Djoongaar/rsa/blob/master/demo.ipynb

#### **3.1 Описание криптосистемы RSA**

**Шаг 1. Выбор простых чисел**: RSA начинается с выбора двух больших простых чисел, обозначаемых как p и q. Также важно условие, что разница этих чисел (p — q), также должно быть больгим числом.

**Шаг 2. Вычисление n**: Затем вычисляется их произведение n = p \* q. Число n будет использоваться как модуль для шифрования и расшифрования.

**Шаг 3. Функция Эйлера**: Вычисляется значение функции Эйлера φ(n) = (p-1)\*(q-1). Функция Эйлера определяет количество целых чисел от 1 до n-1, взаимно простых с n.

**Шаг 4. Выбор e**: Выбирается открытая экспонента e, которая является относительно простым числом и взаимно проста с φ(n). Эта экспонента в паре с модулем алгоритма n будут являться открытым ключом.

**Шаг 5. Нахождение d**: Закрытая экспонента d вычисляется как мультипликативная инверсия открытой экспоненты e по модулю φ(n). То есть d \* e ≡ 1 (mod φ(n)). Эта экспонента будет являться закрытым ключом алгоритма и будет храниться пользователем в тайне и использоваться для расширофвания данных зашифрованным открытым ключем. Сложность вычисления закрытого ключа d обусловлена тем, что для этого требуется вычислить функцию функцию Эйлера, а значит, факторизоваться число n (модуль алгоритма) на простые множители p и q, а это задача не решаемая, при соблюдении требований ко всем параметрам алгоритма.

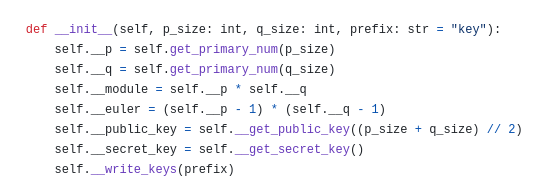
Шаг 6. Зашифрование и расшифрование. Для шифрования сообщения *m* в число *c* используется следующая формула: *c = m^e mod n.* Расшифрование осуществляется по формуле: *m = c^d mod n.*

#### **3.2 Генератор ключевой пары**

Программа реализована в классе *KeyGenerator* в файле *keygen.py* в данном репозитории.

Для генерации ключевой пары требуется вызвать класс KeyGenerator с параметрами длинн чисел p *(p\_size)* и q *(q\_size)*, например *KeyGenerator(32, 64.* Опционально можно передать название файла куда сохранять созданные два ключа, например *KeyGenerator(32, 64 «my\_keys»).* Тогда будлут созданы два файла с ключами *my\_keys.public* и *my\_keys.secret.*

Полученные параметра размера *p\_size* и *q\_size* будут переданы в в конструктор класса метод \_\_init\_\_():



Далее метод \_\_init\_\_ вызовет по порядку все необходимые методы для создания и генерации всех последующих параметров класса, включая саму пару ключей и запишет её в файлы.

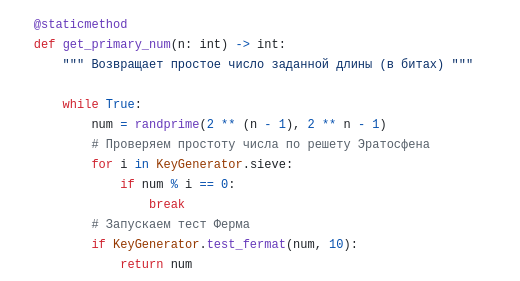
Рассчет модуля алгоритма в методе \_\_init\_\_:

***self.module = self.\_\_p \* self.\_\_q***

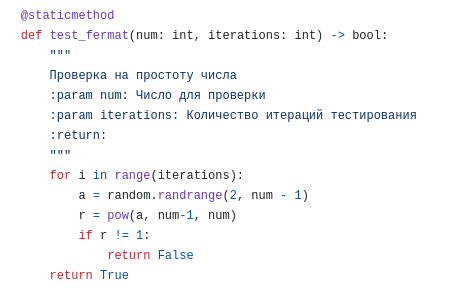
Рассчет значения функции Эйлера в методе \_\_init\_\_:

**self.\_\_euler = (self.\_\_p - 1) \* (self.\_\_q — 1)**

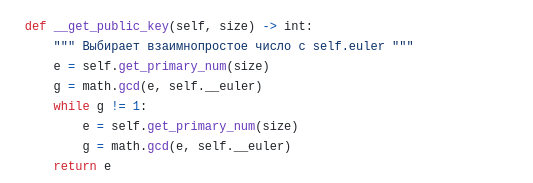
Метод ***get\_primary\_num(n)*** — генерирует просто число заданной длины в битах, проверяет простоту на решете Эратосфена, а затем проводит тест Ферма 10 раз.



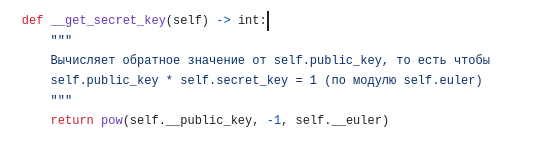
Статический метод ***test\_fermat(num, iterations)*** принимаетна вход два целочисленных значения: num — кандидат для проверки и iterations — количество циклов проверки.

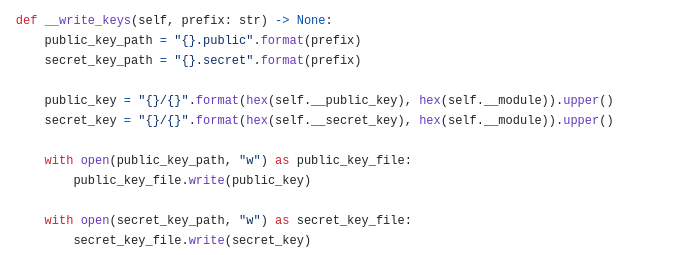
******

Метод ***\_\_get\_public\_key(size)*** принимает параметр размера числа которое станет экспонентой зашифрования, а затем просто проверяет его простоту относительно значения функции Эйлера и возвращает его



Закрытый ключ рассчитывается в методе ***\_\_get\_secret\_key()*** весьма просто: возвожу открытый ключ в -1 степень и получаю обратное значение. Вероятно правильнее было бы по честному реализовать расширенный алгоритм Евклида, но поскольку задача данной работы в получении навыков разработки — то предпочту пользоваться всеми благами языка программирования:

Наконец метод ***\_\_write\_keys()*** записывает полученную нами пару ключей в файлы с нужными названиями, по умолчанию — *key.public / key.secret*



**3.3 Зашифрование и расшифрование**

Программная реализация алгоритма зашифрования и расшифрования находится в одном общем классе RSA в файле rsa.py.

Метод-конструктор \_\_init\_\_() получает на вход путь к файлу с открытым ключом в параметре public\_key\_path или к закрытому ключу в параметре secret\_key\_path или оба эти пути, тогда объект этого класса может и зашифровывать и расшифровывать данные.

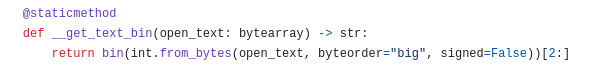
Ключи и модуль алгоритма добываются из файла в 16 — ричном формате, затем преобразуются в десятичные значения и записываются в параметры объекта: *self.public\_key\_int, self.secret\_key\_int, self.module\_int.* Также подсчитывается размер блока как двоичный логарифм модуля округленный вниз до целочисленного значения.

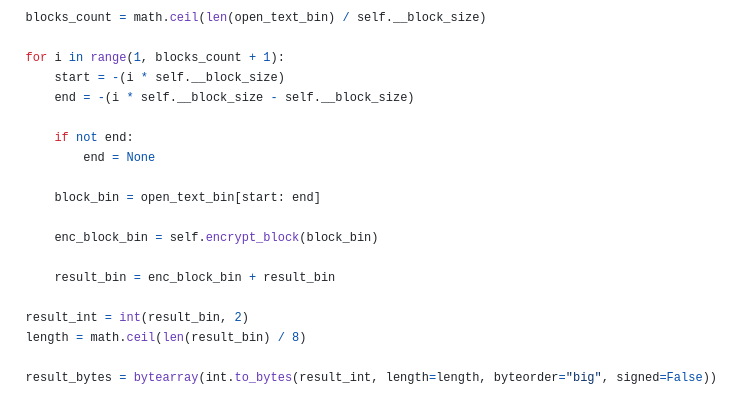
Само зашифрование состоит из следующих этапов:

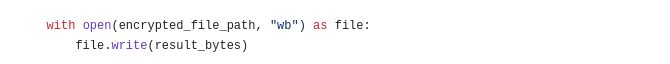
1) Открываем и считываем файл в бинарном представлении

****

2) Превращаем байты в биты и записываем их в виде строки

3) Делим биты на блоки (размер блока ранее был посчитан) и каждый блок шифруем отдельно и увеличиваем в размере нулями, складываем все зашифрованные биты в одну большую строку и превращаем обратно в байты

4) Записываем полученный байты в файл

5) Расшифровывание происходит в точно также, только с применением секретного ключа вместо открытого и зашифрованный текст дробится на блоки увеличенного размера self.block\_size\_full = self.block\_size\_full + 1

**4. Пример ручного шифрования**

Пример ручного шифрования в Приложении 1.

**5. Демонстрация работы программы**

Подробная демонстрация работы программы приведена в Приложении 2.

**6. Демонстрация криптоанализа ключей алгоритма**

Подробная демонстрация криптоанализа приведена в Приложении 3.

**7. Выводы**

Глубоко изучил алгоритм асиметричного шифрования «RSA» и получил навыки его реализации.

# ПРИЛОЖЕНИЕ 4. Список использованных источников

1. Исходный код скрипта для генерации ключевой пары — URL: https://github.com/Djoongaar/rsa/blob/master/keygen.py

2. Исходный код скрипта для зашифрования и расшифрования — URL: <https://github.com/Djoongaar/rsa/blob/master/rsa.py>

3. Демонстрационный юпитер ноутбук — URL: https://github.com/Djoongaar/rsa/blob/master/demo.ipynb

4. Пример криптоанализа с ключевой пары RSA — URL: <https://github.com/Djoongaar/rsa/blob/master/analyze.ipynb>

5. Демонстрация ручного шифрования по алгоритму RSA — URL: https://github.com/Djoongaar/rsa/blob/master/manual.ipynb