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
«30» 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

2 Краткая теоретическая часть…….4

3 Описание программной реализации…….5

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

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

3.3 Зашифрование и расшифрование ……...……....8

4 Пример ручного шифрования …….10

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

6 Криптоанализ……10

7 Выводы ……..10

8 Список использованных источников …….12

Приложение 1 ……..13

Приложение 2 ……..21

Приложение 3 ……..28

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;

2 изучить методы криптоанализа выбранной криптосистемы;

3 реализовать (вручную или программно) не менее одной атаки на выбранную криптосистему, исключая наивную переборную атаку, для случая, когда параметры криптосистемы не являются большими числами;

4 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.

Этот алгоритм включает в себя несколько шагов. Сначала генерируются два простых числа, которые обычно обозначаются как p и q. Эти числа должны быть достаточно большими, чтобы усложнить процесс факторизации и защитить систему от атак. Затем вычисляется их произведение n = p * q, которое служит в качестве модуля для шифрования и дешифрования данных.

После этого выбирается целое число e (обычно называемое открытой экспонентой), которое является открытым ключом и должно быть взаимно простым с функцией Эйлера от числа n. Функция Эйлера от числа n равна (p-1) * (q-1).

Затем вычисляется закрытая экспонента d, которая является мультипликативным обратным числу e по модулю функции Эйлера от числа n. То есть, (d * e) mod ((p-1) * (q-1)) = 1.

После того как получены открытый и закрытый ключи, открытый ключ (n, e) передается другим участникам системы для шифрования данных, а закрытый ключ (n, d) остается в секрете и используется для расшифрования.

Шифрование данных в алгоритме RSA осуществляется путем возведения сообщения в степень e по модулю n. Расшифрование происходит путем возведения зашифрованного сообщения в степень d по модулю n.

Благодаря сложности факторизации больших чисел, алгоритм RSA обеспечивает высокий уровень безопасности и широко используется в сферах, где требуется защита данных, таких как интернет-банкинг, электронная коммерция и аутентификация пользователей.

3. Описание программной реализации

Все файлы данной практической работы я опубликовал в своем гитхаб репозитории https://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 $\varphi(n) = (p-1)*(q-1)$. 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 $\varphi(n)$. 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 $\varphi(n)$. То есть $d * e \equiv 1 \pmod{\varphi(n)}$. 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, факторизовать 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 длин 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 будут 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__():

```
def __init__(self, p_size: int, q_size: int, prefix: str = "key"):
    self.__p = self.get_primary_num(p_size)
    self.__q = self.get_primary_num(q_size)
    self.__module = self.__p * self.__q
    self.__euler = (self.__p - 1) * (self.__q - 1)
    self.__public_key = self.__get_public_key((p_size + q_size) // 2)
    self.__secret_key = self.__get_secret_key()
    self.__write_keys(prefix)
```

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.

```
@staticmethod
def get_primary_num(n: int) -> int:
    """ Возвращает простое число заданной длины (в битах) """

    while True:
        num = randprime(2 ** (n - 1), 2 ** n - 1)
        # Проверяем простоту числа по решету Эратосфена
        for i in KeyGenerator.sieve:
            if num % i == 0:
                break
        # Запускаем тест Ферма
        if KeyGenerator.test_fermat(num, 10):
            return num
```

Статический метод ***test_fermat(num, iterations)*** принимает на вход два целочисленных значения: num — кандидат для проверки и iterations — количество циклов проверки.

```
@staticmethod
def test_fermat(num: int, iterations: int) -> bool:
    """
    Проверка на простоту числа
    :param num: Число для проверки
    :param iterations: Количество итераций тестирования
    :return:
    """
    for i in range(iterations):
        a = random.randrange(2, num - 1)
        r = pow(a, num-1, num)
        if r != 1:
            return False
    return True
```

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

```
def __get_public_key(self, size) -> int:
    """ Выбирает взаимнопростое число с self.euler """
    e = self.get_primary_num(size)
    g = math.gcd(e, self.__euler)
    while g != 1:
        e = self.get_primary_num(size)
        g = math.gcd(e, self.__euler)
    return e
```

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:

```
def __get_secret_key(self) -> int:
    """
    Вычисляет обратное значение от self.public_key, то есть чтобы
    self.public_key * self.secret_key = 1 (по модулю self.euler)
    """
    return pow(self.__public_key, -1, self.__euler)
```

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

```
def __write_keys(self, prefix: str) -> None:
    public_key_path = "{}.public".format(prefix)
    secret_key_path = "{}.secret".format(prefix)

    public_key = "{}/{}".format(hex(self.__public_key), hex(self.__module)).upper()
    secret_key = "{}/{}".format(hex(self.__secret_key), hex(self.__module)).upper()

    with open(public_key_path, "w") as public_key_file:
        public_key_file.write(public_key)

    with open(secret_key_path, "w") as secret_key_file:
        secret_key_file.write(secret_key)
```

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.

```
def __get_keys(self, public_key_path, secret_key_path):
    if public_key_path:
        with open(public_key_path, 'r') as f:
            raw_data = f.read().split("/")
            self.__public_key = raw_data[0]
            self.public_key_int = int(raw_data[0], 0)
            self.public_key_size = len(bin(self.public_key_int)[2:])
            self.__module = raw_data[1]

    if secret_key_path:
        with open(secret_key_path, 'r') as f:
            raw_data = f.read().split("/")
            self.__secret_key = raw_data[0]
            self.secret_key_int = int(raw_data[0], 0)
            self.secret_key_size = len(bin(self.secret_key_int)[2:])
            self.__module = raw_data[1]

    if self.__module:
        self.module_int = int(self.__module, 0)
        self.__block_size = math.floor(math.log(self.module_int, 2))
        self.__block_size_full = math.ceil(math.log(self.module_int, 2))
```

Само зашифрование состоит из следующих этапов:

1) Открываем и считываем файл в бинарном представлении

```
with open(source_file_path, mode="rb") as file:
    open_text = bytearray(file.read())
```

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

```
@staticmethod
def __get_text_bin(open_text: bytearray) -> str:
    return bin(int.from_bytes(open_text, byteorder="big", signed=False))[2:]
```

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

```
blocks_count = math.ceil(len(open_text_bin) / self.__block_size)

for i in range(1, blocks_count + 1):
    start = -(i * self.__block_size)
    end = -(i * self.__block_size - self.__block_size)

    if not end:
        end = None

    block_bin = open_text_bin[start: end]

    enc_block_bin = self.encrypt_block(block_bin)

    result_bin = enc_block_bin + result_bin

result_int = int(result_bin, 2)
length = math.ceil(len(result_bin) / 8)

result_bytes = bytearray(int.to_bytes(result_int, length=length, byteorder="big", signed=False))
```

4) Записываем полученный байты в файл

```
with open(encrypted_file_path, "wb") as file:
    file.write(result_bytes)
```

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

В ходе реализации алгоритма RSA вы приобрели ценные знания и навыки:

1. Понимание асимметричного шифрования: Вы освоили принципы работы асимметричного шифрования, где для шифрования и расшифрования используются разные ключи, что обеспечивает безопасный обмен данными.

2. Знание математических основ RSA: Вы познакомились с математическими концепциями, лежащими в основе алгоритма RSA, такими как факторизация больших чисел и вычисление функции Эйлера.
3. Управление ключами: Вы научились генерировать, использовать и управлять открытыми и закрытыми ключами, что является важным аспектом в криптографии и информационной безопасности.
4. Понимание применения криптографии в реальной жизни: Вы осознали важность криптографических методов в защите конфиденциальной информации и безопасном обмене данными в современном информационном обществе.
5. Значение безопасности данных: Реализация алгоритма RSA помогла вам понять значимость безопасности данных и методов их защиты, особенно в контексте финансовых операций и обмена конфиденциальной информацией.

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
6. RSA простыми словами и в картинках — URL: https://habr.com/ru/articles/745820/
7. What is RSA algorithm and how does it work? — URL: https://hiruna.medium.com/what-is-rsa-algorithm-and-how-does-it-work-fd0a84862543
8. Иллюстрация работы RSA— URL: https://www.michurin.net/computer-science/rsa.html?ysclid=lvk3wgp9l6606406652
9. RSA Algorithm— URL: https://leimao.github.io/article/RSA-Algorithm/

Приложение 1

Ручное шифрование

Определю открытый текст для шифрования

In [1]:
```
# Возьму для шифрования условную строку

open_text = "evgeny"

# Предствлю её в байтах, в целочисленном виде, в битах

open_text_bytes = open_text.encode()
open_text_int = int.from_bytes(open_text_bytes, byteorder="big", sig
open_text_bin = bin(open_text_int)[2:]

import json

print(json.dumps({
    "open_text": open_text,
    "open_text_bytes": str(open_text_bytes),
    "open_text_int": open_text_int,
    "open_text_bin": open_text_bin
}, indent=4))
```

```
{
    "open_text": "evgeny",
    "open_text_bytes": "b'evgeny'",
    "open_text_int": 111559215246969,
    "open_text_bin": "11001010111011001100111011001010110111001111001
01"
}
```

Сгенерирую ключевую пару

In [2]:
```
# Сгенерирую вручную открытый ключ длиной 8 бит по модулю длиной
# 16 бит

# Шаг 1. Придумаю параметры p и q длиной 7 и 9 бит

p = 71
q = 277

len(bin(p)[2:]), len(bin(q)[2:])
```

Out[2]: (7, 9)

In [3]:
```
# Шаг 2. Вычислю n, φ(n)

n = p * q
phi = (p - 1)*(q - 1)

n, phi
```

Out[3]: (19667, 19320)

In [4]:
```
# Шаг 3. Придумаю e (экспонента зашифрования)

e = 223

assert len(bin(e)[2:]) == 8
```

In [5]:
```
# Шаг 4. Вычислю значение закрытого ключа

d = pow(e, -1, phi)
print(d)
```

```
18367
```

Шаг 4.1 Вычислю значение закрытого ключа вручную

q	r	x	y	a	b	x_2	x_1	y_2	y_1
-	-	-	-	19320	223	1	0	0	1
86	142	1	-86	223	142	0	1	1	-86
1	81	-1	87	142	81	1	-1	-86	87
1	61	2	-173	81	61	-1	2	87	-173
1	20	-3	260	61	20	2	-3	-173	260
3	1	11	-953	20	1	-3	11	260	-953
20	0	-223	19320	1	0	11	-223	***-953***	19320
---	---	---	---	---	---	---	---	---	---

$$y_2 = -953(mod 19320) = 18367(mod 19320)$$

Что совпадает с предыдущими вычислениями

In [6]:
```
# Шаг 5. Проверю правильность вычислений

import random
x = random.randrange(2, 99)
x_encrypted = pow(x, e, n)
x_decrypted = pow(x_encrypted, d, n)

bool(x == x_decrypted)
```

Out[6]: True

Зашифрую текст

In [7]:
```
# Вычислю размер блока

import math

block_size = math.log(n, 2)

# Округлю его вниз

block_size = int(block_size)
block_size
```

Out[7]: 14

In [8]:
```
# Разделю открытый текст на блоки
# Получится 3 полных блока и один не полный

len(open_text_bin) / block_size
```

Out[8]: 3.357142857142857

In [9]:
```
# Раздроблю открытый текст на блоки с конца


block_4 = "11001"
block_3 = "01011101100110"
block_2 = "01110110010101"
block_1 = "10111001111001"

# Превращу биты в целочисленные значения

block_4_int = int(block_4, 2)
block_3_int = int(block_3, 2)
block_2_int = int(block_2, 2)
block_1_int = int(block_1, 2)

block_4_int, block_3_int, block_2_int, block_1_int
```

Out[9]: (25, 5990, 7573, 11897)

In [10]:
```
# Зашифруем блоки возведя в экспоненту e по модулю n

block_4_int_enc = block_4_int ** e % n
block_3_int_enc = block_3_int ** e % n
block_2_int_enc = block_2_int ** e % n
block_1_int_enc = block_1_int ** e % n

# Посмотрим что получилось

block_4_int_enc, block_3_int_enc, block_2_int_enc, block_1_int_enc
```

Out[10]: (16690, 17436, 9865, 13431)

Произведу возведение в степень блока номер 1 вручную:

$$11897^{223} mod 19667 = 11897^{1+222} mod 19667 = 11897 * 11897^{222} mod 19667 =$$
$$11897 * 14877^{111} mod 19667 = 11897 * 14877^{3*37} mod 19667 =$$
$$11897 * 5385^{37} mod 19667 = 11897 * 5385^{36+1} mod 19667 =$$

$$11897 * 5385 * 5385^{36} mod 19667 = 11897 * 5385 * 5385^{3*3*2*2} mod 19667 =$$
$$11897 * 5385 * 12301^{3*2*2} mod 19667 = 11897 * 5385 * 16624^{2*2} mod 19667 =$$
$$11897 * 5385 * 16359^{2} mod 19667 = 11897 * 5385 * 8012 mod 19667 = \textbf{\textit{13431}}$$

Что совпадает с автоматическими вычислениями выше

In [11]:
```
# Теперь зашифрованные блоки преобразуем в биты

block_4_bin_enc = bin(block_4_int_enc)[2:]
block_3_bin_enc = bin(block_3_int_enc)[2:]
block_2_bin_enc = bin(block_2_int_enc)[2:]
block_1_bin_enc = bin(block_1_int_enc)[2:]

# Посмотрим что получилось

block_4_bin_enc, block_3_bin_enc, block_2_bin_enc, block_1_bin_enc
```

Out[11]:
```
('100000100110010', '100010000011100', '10011010001001', '110100011
10111')
```

In [12]:
```
# Проверим длину блоков

len(block_4_bin_enc), len(block_3_bin_enc), len(block_2_bin_enc), le
```

Out[12]:
```
(15, 15, 14, 14)
```

In [13]:
```
# Блок 1 и 2 имеют длину больше чем остальные.
# Выровняем до одной длины добавив блоками меньшего размера
# по незначащему нулю и увеличив общий размер блока до 15

block_size_full = block_size + 1

block_2_bin_enc = block_2_bin_enc.zfill(block_size_full)
block_1_bin_enc = block_1_bin_enc.zfill(block_size_full)

# Еще раз проверим длину блоков

len(block_4_bin_enc), len(block_3_bin_enc), len(block_2_bin_enc), le
```

Out[13]:
```
(15, 15, 15, 15)
```

In [14]:
```
# Преобразую биты в байты.

# Сначала сложим все биты в одну большую строку

encrypted_text_bin = block_4_bin_enc + block_3_bin_enc + block_2_bin_

# Посчитаем количество байтов

bytes_count = math.ceil(len(encrypted_text_bin) / 8)

# И наконец превратим биты в байты и запишем в файл

encrypted_text_bytes = int.to_bytes(int(encrypted_text_bin, 2), leng

with open("encrypted_text.txt", "wb") as file:
    file.write(encrypted_text_bytes)

# Прочитаем зашифрованные байты

!cat encrypted_text.txt
```

```
✓&Q☎♌D�w
```

In [15]:
```
# Расшифруем зашифрованный текст

# Преобразуем байты в биты

encrypted_text_bin = bin(int.from_bytes(encrypted_text_bytes, byteor
encrypted_text_bin
```

Out[15]:
```
'1000001001100101000100000111000100110100010010110100011101111'
```

In [16]:
```
# Измерим длину битовой строки

len(encrypted_text_bin)
```

Out[16]:
```
60
```

In [17]:
```
# Делится ли на нашу увеличенную длину блока нацело?

len(encrypted_text_bin) / block_size_full
```

Out[17]:
```
4.0
```

In [18]:
```
# Да, делится нацело!
# Значит идём по блокам и расшифровываем. Для этого
# нужно преобразовать каждый блок в целочисл значение
# затем возвести в экспоненту расшифрования d по модулю n

encrypted_block_4 = "100000100110010"
encrypted_block_3 = "100010000011100"
encrypted_block_2 = "010011010001001"
encrypted_block_1 = "011010001110111"

encrypted_block_4_int = int(encrypted_block_4, 2)
encrypted_block_3_int = int(encrypted_block_3, 2)
encrypted_block_2_int = int(encrypted_block_2, 2)
encrypted_block_1_int = int(encrypted_block_1, 2)

decrypted_block_4_int = encrypted_block_4_int ** d % n
decrypted_block_3_int = encrypted_block_3_int ** d % n
decrypted_block_2_int = encrypted_block_2_int ** d % n
decrypted_block_1_int = encrypted_block_1_int ** d % n

decrypted_block_4_int, decrypted_block_3_int, decrypted_block_2_int,
```

Out[18]:
```
(25, 5990, 7573, 11897)
```

In [19]:
```
# Если я правильно расшифровал то целочисленные значения
# расшифрованных блоков и блоков открытого текста до шифрования
# должны совпасть

bool(
    (decrypted_block_4_int, decrypted_block_3_int, decrypted_block_2_
    (block_4_int, block_3_int, block_2_int, block_1_int)
)
```

Out[19]:
```
True
```

In [20]:
```
# Для наглядности продолжу преобразования до получения исходной стро
# добавляю нолики в начале блока до исходного размера блока

decrypted_block_4_bin = bin(decrypted_block_4_int)[2:].zfill(block_s
decrypted_block_3_bin = bin(decrypted_block_3_int)[2:].zfill(block_s
decrypted_block_2_bin = bin(decrypted_block_2_int)[2:].zfill(block_s
decrypted_block_1_bin = bin(decrypted_block_1_int)[2:].zfill(block_s

decrypted_bin = decrypted_block_4_bin + decrypted_block_3_bin + decry
decrypted_int = int(decrypted_bin, 2)
int.to_bytes(decrypted_int, 6, byteorder="big", signed=False)
```

Out[20]:
```
b'evgeny'
```

In [21]:
```
# Вуаля!
```

In [22]:
```
# Теперь натравим программу на ту же строку

# Создам небольшого размера пару ключей RSA
# Видим пару ключей с префиксом manual в репозитории

from keygen import KeyGenerator

KeyGenerator(8, 8, "manual")
!ls -l
```

```
total 920
-rw-r--r-- 1 evgeny evgeny 431919 Apr 25 19:45 analyze.ipynb
-rw-r--r-- 1 evgeny evgeny  87080 Apr 25 17:06 demo.ipynb
-rw-r--r-- 1 evgeny evgeny      8 Apr 29 10:22 encrypted_text.txt
-rw-r--r-- 1 evgeny evgeny  35117 Apr 23 16:57 image.png
-rw-r--r-- 1 evgeny evgeny   4172 Apr 26 17:00 keygen.py
-rw-r--r-- 1 evgeny evgeny  25002 Apr 29 10:21 manual.ipynb
-rw-r--r-- 1 evgeny evgeny     11 Apr 29 10:22 manual.public
-rw-r--r-- 1 evgeny evgeny     13 Apr 29 10:22 manual.secret
-rw-r--r-- 1 evgeny evgeny 306562 Apr 25 16:42 mayday.png
-rw-r--r-- 1 evgeny evgeny    408 Apr 20 15:36 open.txt
drwxr-xr-x 2 evgeny evgeny   4096 Apr 26 17:09 __pycache__
-rw-r--r-- 1 evgeny evgeny   5100 Apr 25 19:02 rsa.py
-rw-r--r-- 1 evgeny evgeny     13 Apr 25 17:06 secret.crack
drwxr-xr-x 6 evgeny evgeny   4096 Apr 15 15:18 venv
```

In [23]:
```
# Создам объект класса RSA для зашифрования и расшифрования

from rsa import RSA

rsa = RSA("manual.public", "manual.secret")
rsa
```

Out[23]:
```
<rsa.RSA at 0x7f51f7f1e110>
```

In [24]:
```
# Создам файл с текстов "evgeny"

!echo "evgeny" > manual_open.txt
!ls -l
```

```
total 924
-rw-r--r-- 1 evgeny evgeny 431919 Apr 25 19:45 analyze.ipynb
-rw-r--r-- 1 evgeny evgeny  87080 Apr 25 17:06 demo.ipynb
-rw-r--r-- 1 evgeny evgeny      8 Apr 29 10:22 encrypted_text.txt
-rw-r--r-- 1 evgeny evgeny  35117 Apr 23 16:57 image.png
-rw-r--r-- 1 evgeny evgeny   4172 Apr 26 17:00 keygen.py
-rw-r--r-- 1 evgeny evgeny  25002 Apr 29 10:21 manual.ipynb
-rw-r--r-- 1 evgeny evgeny      7 Apr 29 10:22 manual_open.txt
-rw-r--r-- 1 evgeny evgeny     11 Apr 29 10:22 manual.public
-rw-r--r-- 1 evgeny evgeny     13 Apr 29 10:22 manual.secret
-rw-r--r-- 1 evgeny evgeny 306562 Apr 25 16:42 mayday.png
-rw-r--r-- 1 evgeny evgeny    408 Apr 20 15:36 open.txt
drwxr-xr-x 2 evgeny evgeny   4096 Apr 26 17:09 __pycache__
-rw-r--r-- 1 evgeny evgeny   5100 Apr 25 19:02 rsa.py
-rw-r--r-- 1 evgeny evgeny     13 Apr 25 17:06 secret.crack
drwxr-xr-x 6 evgeny evgeny   4096 Apr 15 15:18 venv
```

In [25]:
```
# Зашифрую

rsa.encrypt("manual_open.txt", "manual_encrypted.txt")
!ls -l
```

```
total 928
-rw-r--r-- 1 evgeny evgeny 431919 Apr 25 19:45 analyze.ipynb
-rw-r--r-- 1 evgeny evgeny  87080 Apr 25 17:06 demo.ipynb
-rw-r--r-- 1 evgeny evgeny      8 Apr 29 10:22 encrypted_text.txt
-rw-r--r-- 1 evgeny evgeny  35117 Apr 23 16:57 image.png
-rw-r--r-- 1 evgeny evgeny   4172 Apr 26 17:00 keygen.py
-rw-r--r-- 1 evgeny evgeny      8 Apr 29 10:22 manual_encrypted.txt
-rw-r--r-- 1 evgeny evgeny  25002 Apr 29 10:21 manual.ipynb
-rw-r--r-- 1 evgeny evgeny      7 Apr 29 10:22 manual_open.txt
-rw-r--r-- 1 evgeny evgeny     11 Apr 29 10:22 manual.public
-rw-r--r-- 1 evgeny evgeny     13 Apr 29 10:22 manual.secret
-rw-r--r-- 1 evgeny evgeny 306562 Apr 25 16:42 mayday.png
-rw-r--r-- 1 evgeny evgeny    408 Apr 20 15:36 open.txt
drwxr-xr-x 2 evgeny evgeny   4096 Apr 26 17:09 __pycache__
-rw-r--r-- 1 evgeny evgeny   5100 Apr 25 19:02 rsa.py
-rw-r--r-- 1 evgeny evgeny     13 Apr 25 17:06 secret.crack
drwxr-xr-x 6 evgeny evgeny   4096 Apr 15 15:18 venv
```

In [26]:
```
rsa.decrypt("manual_encrypted.txt", "manual_decrypted.txt")
!cat manual_decrypted.txt
```

```
evgeny
```

In [27]:
```
# Вуаля еще раз
```

Приложение 2. Демонстрация работы программы

Часть 1. Генератор ключей

In [1]:
```
# Сначала очистим директорию от артифактов предыдущих запусков кода

!rm -f encrypted.* decrypted.* *.public *.secret
!ls -l
```

```
total 1300
-rw-r--r-- 1 evgeny evgeny 431919 Apr 25 19:45 analyze.ipynb
-rw-r--r-- 1 evgeny evgeny  87080 Apr 25 17:06 demo.ipynb
-rw-r--r-- 1 evgeny evgeny  35117 Apr 23 16:57 image.png
-rw-r--r-- 1 evgeny evgeny   4172 Apr 26 17:00 keygen.py
-rw-r--r-- 1 evgeny evgeny  24721 Apr 29 10:25 manual.ipynb
-rw-r--r-- 1 evgeny evgeny 399380 Apr 29 10:27 manual.pdf
-rw-r--r-- 1 evgeny evgeny 306562 Apr 25 16:42 mayday.png
-rw-r--r-- 1 evgeny evgeny    408 Apr 20 15:36 open.txt
drwxr-xr-x 2 evgeny evgeny   4096 Apr 26 17:09 __pycache__
-rw-r--r-- 1 evgeny evgeny   5100 Apr 25 19:02 rsa.py
-rw-r--r-- 1 evgeny evgeny     13 Apr 25 17:06 secret.crack
drwxr-xr-x 6 evgeny evgeny   4096 Apr 15 15:18 venv
```

In [2]:
```
# Запускаем генерацию ключевой пары
# Скрипт по-умолчанию создаёт пару ключей с названиями "key.pub"
# и "key.secret", или можно напрямую передать скрипту префикс и
# получить пару ключей с другими названиями

# Генератор ключевой пары принимает два парметра p_size и q_size
# Чтобы соблюсти условие значительной величины их разницы -
# задам им разную длину

from keygen import KeyGenerator

KeyGenerator(1024, 3072)

# Проверим создались ли файлы с ключами
!ls -l
```

```
total 1308
-rw-r--r-- 1 evgeny evgeny 431919 Apr 25 19:45 analyze.ipynb
-rw-r--r-- 1 evgeny evgeny  87080 Apr 25 17:06 demo.ipynb
-rw-r--r-- 1 evgeny evgeny  35117 Apr 23 16:57 image.png
-rw-r--r-- 1 evgeny evgeny   4172 Apr 26 17:00 keygen.py
-rw-r--r-- 1 evgeny evgeny   1541 Apr 29 10:28 key.public
-rw-r--r-- 1 evgeny evgeny   2053 Apr 29 10:28 key.secret
-rw-r--r-- 1 evgeny evgeny  24721 Apr 29 10:25 manual.ipynb
-rw-r--r-- 1 evgeny evgeny 399380 Apr 29 10:27 manual.pdf
-rw-r--r-- 1 evgeny evgeny 306562 Apr 25 16:42 mayday.png
-rw-r--r-- 1 evgeny evgeny    408 Apr 20 15:36 open.txt
drwxr-xr-x 2 evgeny evgeny   4096 Apr 26 17:09 __pycache__
-rw-r--r-- 1 evgeny evgeny   5100 Apr 25 19:02 rsa.py
-rw-r--r-- 1 evgeny evgeny     13 Apr 25 17:06 secret.crack
drwxr-xr-x 6 evgeny evgeny   4096 Apr 15 15:18 venv
```

In [3]:
```
# Посмотрим содержимое key.pub
# Файл содержит строки в 16-м формате со значением
# e (открытый ключ) и n (модуль) разделенные знаком /

!cat key.public
```

```
0XA7733914AF645573B787DDD55F1CA35CF3FC0DA25639A97F784A343D4223A4E99
F784969A54D8D71FEA3BADB57B8C25ADC6F0367C30CA023E361D35DF5DE1243CE6F
B2FDC386D3B9F371D2F8B360584F7DABE642146A00C77E51E2BF802199D3FF3C892
D73A39B2F5C0BE9FB261C32D1E9F3186C98424B99748347FF0EAA1217F3D5EC49D1
EEB0CF0D9606309A1E778B2BAC6AD19E71D0A8B128C8D055C8FBF30662DD0B41BAA
DFFF8C5CB2F352D3154FAEF5B424DA62440C7A6C7786CD6ACB324479288E8CDA014
85AFCACE3DF0AD9824DBB6932CA32778A332BA09E5C88B9C621CB4D1A663276371B
3CE692AAE131729BE11BEA7BE545A8B0EB130CC966D3F/0X86C7A34204FEB43E5C3
B4940AD51819F0F729F892EF3B0F7274759D4C7BFB24B4940DC676DDEA556849E32
60288454B197CF90CEBE2FF14FD926B3DB72B5E374256294AE0E8DC2E015579DC15
2ED78AD54EC273194194812DF0368A2EE14C7FE9574452ACB23B477817BD961B47E
7101ED529A4FB771F842E042DF4F799835851B569F4FADC389940FEDBAAAF151213
EAE9EC51E2E0F6428005125C4EE3D5B06B64820752FA18DF042DB53963B85E8A8AC
462F14778FB37A5DA2DA5CAAA4157CE23CA44F3C52C50E21FC8737139C61609A0D2
A47093AB29F7386A617DB7CD0B15C988628601B243876D39459092CCBEE843A67BB
BB908935A97FEBEB50F033CAAE328D4C84A6F187B45EE6DF7FD4FF0CDDBF8E6378A
D308C4FD5F8C08648B77D75036CB9B034DEF6CDCAB27691917465F509B49E591DC5
91AC7A86B78C5C6D4AEF3A222617AD670DEA4EE22AD63FF41F948BB32B827D7ECE4
77F9866CD2E7DE7D083BE156A187CF4F0B18190952C2E3FBF8872815B09DD6635C0
F53417494354BDD47F1F95A6503BF237D430E39CE737F7D86340C96AEB9974E1C27
7EDBFE20C607BAF8C1020999468E09B004979C3A0AEAE773E814646E73E33900D6C
1D55FFF6DC40EF8D580617DAC0D72DF0A9B7D5F87EEF5D02AD046991AB742D63C35
B58CD59C3BED3385558762424EE8BEDA972026AA1263CAFE1B41754F4D70F5E25F3
```

In [4]:
```
# Посторим файл key.secret
# Файл содержит строки в 16-м формате со значением
# d (закрытый ключ) и n (модуль) разделенные знаком /

!cat key.secret
```

```
0X45470070F08F341787D931C8291287A4394B387086E8F29ED105E76FB058F59B8
6109BDC3B25EB0396E8C5F055B7CB29E26074C99227113CFEBCF6C009DEF7EED47E
82EAB95C840EFAFA7017A745CF31E7C3B637F4D720694EA2A4A52C345EAD99F6CC6
7BA0D1DAFF5FB86B02804D207AF26AA14960D3374C3F0938E630A193FBF4544CFDA
9E94015FE35D4FDC6F964229EB7A0F79AD762A91F1899641C5BA8036609B45A8A48
1D5F3DBEDAAB8972EB1A04345D0170803FC76C60AB2BC03C92A9E41EB26A80915C1
67545570F0B2E2FEB4C3B324545E318F93D8B5E909B9E86FE14BE16A918AFD7E3DD
B0277D8BFDBE6EE8D51BB5C20CA837C9AA7EE13536841A57C858C699752E3EE65B3
9625D88D6075F98F947B0B7895B74B777BAA9D5C5407496E3EDC84D201E8CBEE2F3
2B998DB78B1512FC42D9C3BE40CB29F0270E24617CEE20E3E04210F47FBF2F2785B
62D6037B39087AED14DE7ED322729545BA819C366F12F63B5D29415A86632D8416C
445F3CD52F8D4095ADDA03B5D96886F0EE9FCD4B7F058027A3438AA7A23894BCA74
3A625980F3128B1F7245C960EB70335FD4B2DDD1D7D9CE24B98DB21E254563288258
8E8D0EB1F2B62E8D4B55BFF7AB68BA5C7426D6A54CC9E4B589D3B1C54E36C26C384
D3E5162C77279625F0608EBFC47FB8755DFBF2CDFC36A16D11310CB049F87AE50DE
01CCA7A128E36F93E6A5F/0X86C7A34204FEB43E5C3B4940AD51819F0F729F892EF
3B0F7274759D4C7BFB24B4940DC676DDEA556849E3260288454B197CF90CEBE2FF1
4FD926B3DB72B5E374256294AE0E8DC2E015579DC152ED78AD54EC273194194812D
F0368A2EE14C7FE9574452ACB23B477817BD961B47E7101ED529A4FB771F842E042
DF4F799835851B569F4FADC389940FEDBAAAF151213EAE9EC51E2E0F6428005125C
4EE3D5B06B64820752FA18DF042DB53963B85E8A8AC462F14778FB37A5DA2DA5CAA
A4157CE23CA44F3C52C50E21FC8737139C61609A0D2A47093AB29F7386A617DB7CD
0B15C988628601B243876D39459092CCBEE843A67BBBB908935A97FEBEB50F033CA
AE328D4C84A6F187B45EE6DF7FD4FF0CDDBF8E6378AD308C4FD5F8C08648B77D750
36CB9B034DEF6CDCAB27691917465F509B49E591DC591AC7A86B78C5C6D4AEF3A22
2617AD670DEA4EE22AD63FF41F948BB32B827D7ECE477F9866CD2E7DE7D083BE156
A187CF4F0B18190952C2E3FBF8872815B09DD6635C0F53417494354BDD47F1F95A6
503BF237D430E39CE737F7D86340C96AEB9974E1C277EDBFE20C607BAF8C1020999
468E09B004979C3A0AEAE773E814646E73E33900D6C1D55FFF6DC40EF8D580617DA
C0D72DF0A9B7D5F87EEF5D02AD046991AB742D63C35B58CD59C3BED338555876242
4EE8BEDA972026AA1263CAFE1B41754F4D70F5E25F3
```

In [5]:
```
# Создам объект класса RSA для шифрования файлов и
# передам ему ключ для шифрования (открытый)

from rsa import RSA

encryptor = RSA(public_key_path="key.public")
encryptor
```

Out[5]:
```
<rsa.RSA at 0x7f9369fc2e90>
```

In [6]:
```
# Для наглядности вывожу в строчном представлении
# модели и её основные параметры. Видим что объект
# класса обладает только ключами для зашифрования

encryptor.__str__()
```

Out[6]:
{'public_key': '0XA7733914AF645573B787DDD55F1CA35CF3FC0DA25639A97F7
84A343D4223A4E99F784969A54D8D71FEA3BADB57B8C25ADC6F0367C30CA023E361
D35DF5DE1243CE6FB2FDC386D3B9F371D2F8B360584F7DABE642146A00C77E51E2B
F802199D3FF3C892D73A39B2F5C0BE9FB261C32D1E9F3186C98424B99748347FF0E
AA1217F3D5EC49D1EEB0CF0D9606309A1E778B2BAC6AD19E71D0A8B128C8D055C8F
BF30662DD0B41BAADFFF8C5CB2F352D3154FAEF5B424DA62440C7A6C7786CD6ACB3
24479288E8CDA01485AFCACE3DF0AD9824DBB6932CA32778A332BA09E5C88B9C621
CB4D1A663276371B3CE692AAE131729BE11BEA7BE545A8B0EB130CC966D3F',
 'public_key_int': 211386154944227170267489187876316867482387530591
157969469491411767877433568885413093407760707033488961653048748004 1
612304218234386387630558075618674653493055329054417047982977502683 6
572768509153574530859656854528610617212330097231292491356864845519 6
472454917921211144760580832383328654427094581676286101514398441416 1
144772335934315166849029998681578918881873745508444910492779875686 5
886713206978917513513577156729502250566865739291622126545508579744 3
282628658383966072046100060161349179103090533871176728969699748453 9
644313014593063111704452628920588441000121707795663714532092178745 3
793890838782501407513705394367807,
 'public_key_size': 2048,
 'secret_key': None,
 'secret_key_int': None,
 'secret_key_size': None,
 'module': '0X86C7A34204FEB43E5C3B4940AD51819F0F729F892EF3B0F727475
9D4C7BFB24B4940DC676DDEA556849E3260288454B197CF90CEBE2FF14FD926B3DB
72B5E374256294AE0E8DC2E015579DC152ED78AD54EC273194194812DF0368A2EE1
4C7FE9574452ACB23B477817BD961B47E7101ED529A4FB771F842E042DF4F799835
851B569F4FADC389940FEDBAAAF151213EAE9EC51E2E0F6428005125C4EE3D5B06B
64820752FA18DF042DB53963B85E8A8AC462F14778FB37A5DA2DA5CAAA4157CE23C
A44F3C52C50E21FC8737139C61609A0D2A47093AB29F7386A617DB7CD0B15C98862
8601B243876D39459092CCBEE843A67BBBB908935A97FEBEB50F033CAAE328D4C84
A6F187B45EE6DF7FD4FF0CDDBF8E6378AD308C4FD5F8C08648B77D75036CB9B034D
EF6CDCAB27691917465F509B49E591DC591AC7A86B78C5C6D4AEF3A222617AD670D
EA4EE22AD63FF41F948BB32B827D7ECE477F9866CD2E7DE7D083BE156A187CF4F0B
18190952C2E3FBF8872815B09DD6635C0F53417494354BDD47F1F95A6503BF237D4
30E39CE737F7D86340C96AEB9974E1C277EDBFE20C607BAF8C1020999468E09B004
979C3A0AEAE773E814646E73E33900D6C1D55FFF6DC40EF8D580617DAC0D72DF0A9
B7D5F87EEF5D02AD046991AB742D63C35B58CD59C3BED3385558762424EE8BEDA97
2026AA1263CAFE1B41754F4D70F5E25F3',
 'module_int': 549853753806690391887839250713982549030937317542518 1
774944152097950566364707022887603884178531900712473849398113381197 2
571661020469059335509817507217216391017923869449432796440654579732 4
105340797918945320410425405415008686196996176865393431734552716480 4
353093187753113255261930651625381565886538005929110251510658211390 9
182577793444378415361082623513241851336726274498272528374526656279 7
262984505669928439794765498490576729168937325374703806639360697298 3
448499074247061243625400727030172229482751108154337571483444205413 6
010783673181705378924202977915384541681390322489692015719843592060
290085925064034458025826970133082746480502416783656983333328894293
323034245676444072242777575984576790168994537967507136868290352181 8
247215891195901678003357841520335808386718218992928778865612182662 4
804130845839763894644231703044567404140599767422608409489082866526 9
635533871154951941202309884422864484136144219309591682370149884460 2
593123844871227548335590479055091509824590693873712472213114516688 6
759155926757109770837178369055775681460173859732591990642479169259 7
270554608630294721547560451779251470897915373347026270850735416677 0
148848913493662241036710003233738783245167115525953339766763140009 4
344423776236469168396028721893029428733427,
 'block_size': 4095,
 'block_size_full': 4096}

In [7]:
```
# Создам отдельный объект для расштфрования и
# передам ему закрытый ключ

decryptor = RSA(secret_key_path="key.secret")
decryptor
```

Out[7]:
```
<rsa.RSA at 0x7f9369651bd0>
```

In [8]:
```
# Также пристально посмотрим на объект через специальный метод
# Этот объект напротив - содержит лишь ключи для расшифрования
# Таким образом можно ограничить функциональность класса просто
# передав или не передавая ей нужные данные.

decryptor.__str__()
```

Out[8]:

```
{'public_key': None,
 'public_key_int': None,
 'public_key_size': None,
 'secret_key': '0X45470070F08F341787D931C8291287A4394B387086E8F29ED
105E76FB058F59B86109BDC3B25EB0396E8C5F055B7CB29E26074C99227113CFEBC
F6C009DEF7EED47E82EAB95C840EFAFA7017A745CF31E7C3B637F4D720694EA2A4A
52C345EAD99F6CC67BA0D1DAFF5FB86B02804D207AF26AA14960D3374C3F0938E63
0A193FBF4544CFDA9E94015FE35D4FDC6F964229EB7A0F79AD762A91F1899641C5B
A8036609B45A8A481D5F3DBEDAAB8972EB1A04345D0170803FC76C60AB2BC03C92A
9E41EB26A80915C167545570F0B2E2FEB4C3B324545E318F93D8B5E909B9E86FE14
BE16A918AFD7E3DDB0277D8BFDBE6EE8D51BB5C20CA837C9AA7EE13536841A57C85
8C699752E3EE65B39625D88D6075F98F947B0B7895B74B777BAA9D5C5407496E3ED
C84D201E8CBEE2F32B998DB78B1512FC42D9C3BE40CB29F0270E24617CEE20E3E04
210F47FBF2F2785B62D6037B39087AED14DE7ED322729545BA819C366F12F63B5D2
9415A86632D8416C445F3CD52F8D4095ADDA03B5D96886F0EE9FCD4B7F058027A34
38AA7A23894BCA743A625980F3128B1F7245C960EB70335FD4B2DDD1D7D9CE24B98
DB21E254563288258E8D0EB1F2B62E8D4B55BFF7AB68BA5C7426D6A54CC9E4B589D
3B1C54E36C26C384D3E5162C77279625F0608EBFC47FB8755DFBF2CDFC36A16D113
10CB049F87AE50DE01CCA7A128E36F93E6A5F',
 'secret_key_int': 282626931940940314234408806868721788655649415624
331743753929656547037022746890800369287973849873306308202663306273 6
065911590564761442313200000039547362764958281288908708873888969385 1
802633262478052914375124996175430660108982359109296515347151099775 1
951011537729391365607708545693887700472154810518460866072658387550 5
195441774351011183527814380364029459417814353687513508022507506381 5
359407282257475967515749845925215283976610580189279068463385647805 0
266159119344022727431400651648613066612833844345854526567710482499 5
704570858528550142325459802485202528424349611303688874542478536399 5
556471311358551377524412947871092988396741401142839357168605583442 7
628456762308865811538099116930370770139037759648926179650558708648 9
983757896717779900593911032298575280175882792957537457400727133337 7
954010076637595658653244402826626903177280654693691749262213915590 5
813557895209070257695306167248462904304795029065185538056857090803 7
981825164898801690457526088270037840162188071703654168671928577226 0
102449348641560050661495662658993508141145815456347389534092946155 1
073610562924451354660950627702944027004092607333684313573541907984 4
110227604515409525431697868492604409331492846319882088420913789303 6
667503906270900789658429462510114806295838166 3,
 'secret_key_size': 4095,
 'module': '0X86C7A34204FEB43E5C3B4940AD51819F0F729F892EF3B0F727475
9D4C7BFB24B4940DC676DDEA556849E3260288454B197CF90CEBE2FF14FD926B3DB
72B5E374256294AE0E8DC2E015579DC152ED78AD54EC273194194812DF0368A2EE1
4C7FE9574452ACB23B477817BD961B47E7101ED529A4FB771F842E042DF4F799835
851B569F4FADC389940FEDBAAAF151213EAE9EC51E2E0F6428005125C4EE3D5B06B
64820752FA18DF042DB53963B85E8A8AC462F14778FB37A5DA2DA5CAAA4157CE23C
A44F3C52C50E21FC8737139C61609A0D2A47093AB29F7386A617DB7CD0B15C98862
8601B243876D39459092CCBEE843A67BBBB908935A97FEBEB50F033CAAE328D4C84
A6F187B45EE6DF7FD4FF0CDDBF8E6378AD308C4FD5F8C08648B77D75036CB9B034D
EF6CDCAB27691917465F509B49E591DC591AC7A86B78C5C6D4AEF3A222617AD670D
EA4EE22AD63FF41F948BB32B827D7ECE477F9866CD2E7DE7D083BE156A187CF4F0B
18190952C2E3FBF8872815B09DD6635C0F53417494354BDD47F1F95A6503BF237D4
30E39CE737F7D86340C96AEB9974E1C277EDBFE20C607BAF8C1020999468E09B004
979C3A0AEAE773E814646E73E33900D6C1D55FFF6DC40EF8D580617DAC0D72DF0A9
B7D5F87EEF5D02AD046991AB742D63C35B58CD59C3BED3385558762424EE8BEDA97
2026AA1263CAFE1B41754F4D70F5E25F3',
 'module_int': 549853753806690391887839250713982549030937317542518 1
774944152097950566364707022887603884178531900712473849398113381197 2
571661020469059335509817507217216391017923869449432796440654579732 4
105340797918945320410425405415008686196996176865393431734552716480 4
353093187753113255261930651625381565886538005929110251510658211390 9
182577793444378415361082623513241851336726274498272528374526656279 7
```

```
26298450566992843979476549849057672916893732537470380663936069729834
44849907424706124362540072703017222948275110815433757148344420541360
10783673181705378924202977915384541681390332248969201571984359206029
00859250640344580258269701330827464805024167836569833333288942933
23034245676440722427775759845767901689945379675071368682903521818247
21589119590167800335784152033580838671821899292877886561218266248041
30845839763894644231703044567404140599767422608409489082866526963553
38711549519412023098844228644841361442193095916823701498844602593123
84487122754833559047905509150982459069387371247221311451668867591559
26757109770837178369055775681460173859732591990642479169259727055460
86302947215475604517792514708979153733470262708507354166770148848913
49366224103671000323373878324516711552595333976676314000943444237762
36469168396028721893029428733427,
 'block_size': 4095,
 'block_size_full': 4096}
```

In [9]:
```
# Хотя можно было все сделать одним объектов
# просто передав ему оба ключа

rsa = RSA(public_key_path="key.public", secret_key_path="key.secret"
rsa.__str__()
```

Out[9]:
{'public_key': '0XA7733914AF645573B787DDD55F1CA35CF3FC0DA25639A97F7
84A343D4223A4E99F784969A54D8D71FEA3BADB57B8C25ADC6F0367C30CA023E361
D35DF5DE1243CE6FB2FDC386D3B9F371D2F8B360584F7DABE642146A00C77E51E2B
F802199D3FF3C892D73A39B2F5C0BE9FB261C32D1E9F3186C98424B99748347FF0E
AA1217F3D5EC49D1EEB0CF0D9606309A1E778B2BAC6AD19E71D0A8B128C8D055C8F
BF30662DD0B41BAADFFF8C5CB2F352D3154FAEF5B424DA62440C7A6C7786CD6ACB3
24479288E8CDA01485AFCACE3DF0AD9824DBB6932CA32778A332BA09E5C88B9C621
CB4D1A663276371B3CE692AAE131729BE11BEA7BE545A8B0EB130CC966D3F',
 'public_key_int': 211386154944227170267489187876316867482387530591
157969469491411767877433568885413093407760707033488961653048748004 1
612304218234386387630558075618674653493055329054417047982977502683 6
572768509153574530859656854528610617212330097231292491356864845519 6
472454917921211144760580832383328654427094581676286101514398441416 1
144772335934315166849029998681578918881873745508444910492779875686 5
886713206978917513513577156729502250566865739291622126545508579744 3
282628658383966072046100060161349179103090533871176728969699748453 9
644313014593063111704452628920588441000121707795663714532092178745 3
793890838782501407513705394367807,
 'public_key_size': 2048,
 'secret_key': '0X45470070F08F341787D931C8291287A4394B387086E8F29ED
105E76FB058F59B86109BDC3B25EB0396E8C5F055B7CB29E26074C99227113CFEBC
F6C009DEF7EED47E82EAB95C840EFAFA7017A745CF31E7C3B637F4D720694EA2A4A
52C345EAD99F6CC67BA0D1DAFF5FB86B02804D207AF26AA14960D3374C3F0938E63
0A193FBF4544CFDA9E94015FE35D4FDC6F964229EB7A0F79AD762A91F1899641C5B
A8036609B45A8A481D5F3DBEDAAB8972EB1A04345D0170803FC76C60AB2BC03C92A
9E41EB26A80915C167545570F0B2E2FEB4C3B324545E318F93D8B5E909B9E86FE14
BE16A918AFD7E3DDB0277D8BFDBE6EE8D51BB5C20CA837C9AA7EE13536841A57C85
8C699752E3EE65B39625D88D6075F98F947B0B7895B74B777BAA9D5C5407496E3ED
C84D201E8CBEE2F32B998DB78B1512FC42D9C3BE40CB29F0270E24617CEE20E3E04
210F47FBF2F2785B62D6037B39087AED14DE7ED322729545BA819C366F12F63B5D2
9415A86632D8416C445F3CD52F8D4095ADDA03B5D96886F0EE9FCD4B7F058027A34
38AA7A23894BCA743A625980F3128B1F7245C960EB70335FD4B2DDD1D7D9CE24B98
DB21E254563288258E8D0EB1F2B62E8D4B55BFF7AB68BA5C7426D6A54CC9E4B589D
3B1C54E36C26C384D3E5162C77279625F0608EBFC47FB8755DFBF2CDFC36A16D113
10CB049F87AE50DE01CCA7A128E36F93E6A5F',
 'secret_key_int': 282626931940940314234408806868721788655649415624
331743753929656547037022746890800369287973849873306308202663306273 6
065911590564761442313200000039547362764958281288908708873888969385 1
802633262478052914375124996175430660108982359109296515347151099775 1
951011537729391365607708545693887700472154810518460866072658387550 5
195441774351011183527814380364029459417814353687513508022507506381 5
359407282257475967515749845925215283976610580189279068463385647805 0
266159119344022727431400651648613066612833844345854526567710482499 5
704570858528550142325459802485202528424349611303688874542478536399 5
556471311358551377524412947871092988396741401142839357168605583442 7
628456762308865811538099116930370770139037759648926179650558708648 9
983757896717779900593911032298575280175882792957537457400727133337 7
954010076637595658653244402826626903177280654693691749262213915590 5
813557895209070257695306167248462904304795029065185538056857090803 7
981825164898801690457526088270037840162188071703654168671928577226 0
102449348641560050661495662658993508141145815456347389534092946155 1
073610562924451354660950627702944027004092607333684313573541907984 4
110227604515409525431697868492604409331492846319882088420913789303 6
667503906270900789658429462510114806295838166 3,
 'secret_key_size': 4095,
 'module': '0X86C7A34204FEB43E5C3B4940AD51819F0F729F892EF3B0F727475
9D4C7BFB24B4940DC676DDEA556849E3260288454B197CF90CEBE2FF14FD926B3DB
72B5E374256294AE0E8DC2E015579DC152ED78AD54EC273194194812DF0368A2EE1
4C7FE9574452ACB23B477817BD961B47E7101ED529A4FB771F842E042DF4F799835
851B569F4FADC389940FEDBAAAF151213EAE9EC51E2E0F6428005125C4EE3D5B06B
64820752FA18DF042DB53963B85E8A8AC462F14778FB37A5DA2DA5CAAA4157CE23C

```
A44F3C52C50E21FC8737139C61609A0D2A47093AB29F7386A617DB7CD0B15C98862
8601B243876D39459092CCBEE843A67BBBB908935A97FEBEB50F033CAAE328D4C84
A6F187B45EE6DF7FD4FF0CDDBF8E6378AD308C4FD5F8C08648B77D75036CB9B034D
EF6CDCAB27691917465F509B49E591DC591AC7A86B78C5C6D4AEF3A222617AD670D
EA4EE22AD63FF41F948BB32B827D7ECE477F9866CD2E7DE7D083BE156A187CF4F0B
18190952C2E3FBF8872815B09DD6635C0F53417494354BDD47F1F95A6503BF237D4
30E39CE737F7D86340C96AEB9974E1C277EDBFE20C607BAF8C1020999468E09B004
979C3A0AEAE773E814646E73E33900D6C1D55FFF6DC40EF8D580617DAC0D72DF0A9
B7D5F87EEF5D02AD046991AB742D63C35B58CD59C3BED3385558762424EE8BEDA97
2026AA1263CAFE1B41754F4D70F5E25F3',
 'module_int': 549853753806690391887839250713982549030937317542518177
749441520979505663647070228876038841785319007124738493981133811972
571661020469059335509817507217216391017923869449432796440654579732410
534079791894532041042540541500868619699617686539343173455271648043530
931877531132552619306516253815658865380059291102515106582113909182577
793444378415361082623513241851336726274498272528374526656279726298
450566992843979476549849057672916893732537470380663936069729834484
990742470612436254007270301722294827511081543375714834442054136010
783673181705378924202977915384541681390332248969201571984359206029008
592506403445802582697013308274648050241678365698333333288942933230
342456764440722427775759845767901689945379675071368682903521818247
215891195901678003357841520335808386718218992928778865612182662480
413084583976389464423170304456740414059976742260840948908286652696
355338711549519412023098844228644841361442193095916823701498844602
593123844871227548335590479055091509824590693873712472213114516688675
915592675710977083717836905577568146017385973259199064247916925972
705546086302947215475604517792514708979153733470262708507354166770
148848913493662241036710003233738783245167115525953339766763140009434
442377623646916839602872189302942873342 7,
 'block_size': 4095,
 'block_size_full': 4096}
```

In [10]:
```
# Проверим правильность работы ключей у объектов encrypted
# и decrypted. Сгенерируем случайное число и зашифруем а
# затем расшифруем их

import random

_int = random.randrange(2, 100)

encr = pow(_int, encryptor.public_key_int, encryptor.module_int)
decr = pow(encr, decryptor.secret_key_int, decryptor.module_int)

bool(decr == _int)
```

Out[10]: True

In [11]:
```
# Проделаем то же самое с объектом rsa чтобы подтвердить
# корректность суждений

_int = random.randrange(2, 100)

encr = pow(_int, rsa.public_key_int, rsa.module_int)
decr = pow(encr, rsa.secret_key_int, rsa.module_int)

bool(decr == _int)
```

Out[11]: True

Часть 2. Зашифрование и расширование

In [12]:
```
# Для демонстрации зашифрования и расшифрования мною были
# подготовлены два файла с тескстом open.txt и изображением
# image.png

# Передам методу encrypt относительный путь к файлу для
# зашифрования и название файлы куда дб записан зашифрованый
# файл encrypted.txt

rsa.encrypt("open.txt", "encrypted.txt")
```

In [13]:
```
# Попробуем прочитать зашифрованый файл

!cat encrypted.txt
```

```
kV������e◄◅◁��p������ᛉ@i,g7J,%#K��-��~@@��ᛊ0
����LIG⍉dpp>'оԋqB&GY№=g☎E▷I♪Z№U⍉I
gS№I⍉ᛊ⍉ek_n⌕Y>Np⍉ز⍉,⍉ᛉI⍉),@`L№g⌕3◷⍉♀
⍉Y?⍉⍉m1⍉⍉p⍉⍉~№UZ ˬ}⍉ş⍉ ⍉⍉⍉¨⍉y⌀4
⍉9♪n⍉%⍉⍉~⍉⍉$¨⍉p⍉v⍉[⍉N⍉O⍉#4⍉f⍉E#⍉⍉⍉¤⍉耻uk◄⍉⍉q⍉
⍉⍉⍉ᛊʸK⍉⍉~⍉⍉⍉⍉⍉⍉⍉F}nO⍉⍉$⍉p
T⌕⍉⍉gx⍉⍉/K⍉⍉z,⍉ᛊe_⍉8⍉ȥc⍉kز⍉⍉Pe⍉@⍉⍉⍉⍉0⍉z ⍉⍉4:⍉-Y⍉
P。ӝ⍉⍉W⍉-J∝⍉80|⍉(⌐v⍉k⍉G^Q^⍉⍉PƑ⍉hd⍉⍉⍉fV⍉îN⍉⍉o⍉C⍉⍉ě
⍉⍉
Ƀ⍉⍉V(⍉~o⍉⍉A⍉7⍉⍉<⍉⍉s⍉⍉☎⍉⍉⍉⍉⍉B⍉⍉⍉⍉∝⍉⍉⍉⍉⍉⍉
⍉/⍉⍉r⍉X♫x⍉w⍉⍉{&⍉⍉\⍉⍉+⍉.z26⍉⍉♀/E⍉⍉8
```

In [14]:
```
# Передам методу decrypt относительный путь к файлу для
# расшифрования и название файлы куда дб записан расшифрованый
# файл decrypted.txt

rsa.decrypt("encrypted.txt", "decrypted.txt")
```

In [15]:
```
!cat decrypted.txt
```

```
Прощай немытая Россия,
Страна рабов, страна господ,
И вы, мундиры голубые,
И ты, им преданный народ.

Быть может, за стеной Кавказа
Сокроюсь от твоих пашей,
От их всевидящего глаза,
От их всеслышащих ушей.

Михаил Лермонтов, 1841г.
```

In [16]:
```
# Проверим корректность расшифровки

with open("open.txt", "rb") as file:
    open_text = file.read()

with open("decrypted.txt", "rb") as file:
    decrypted_text = file.read()

bool(open_text == decrypted_text)
```

Out[16]:
```
True
```

In [17]:
```
# Установлю библиотеку для просмотра изображений

!pip3 install pillow
```

```
Defaulting to user installation because normal site-packages is not
writeable
Requirement already satisfied: pillow in /usr/lib/python3/dist-pack
ages (9.4.0)
```

In [18]:
```
# Зашифрую изображение и сохраню в файл encrypted.png

rsa.encrypt("image.png", "encrypted.png")
!ls -l
```

```
total 1352
-rw-r--r-- 1 evgeny evgeny 431919 Apr 25 19:45 analyze.ipynb
-rw-r--r-- 1 evgeny evgeny    408 Apr 29 10:29 decrypted.txt
-rw-r--r-- 1 evgeny evgeny  88771 Apr 29 10:29 demo.ipynb
-rw-r--r-- 1 evgeny evgeny  35328 Apr 29 10:30 encrypted.png
-rw-r--r-- 1 evgeny evgeny    512 Apr 29 10:29 encrypted.txt
-rw-r--r-- 1 evgeny evgeny  35117 Apr 23 16:57 image.png
-rw-r--r-- 1 evgeny evgeny   4172 Apr 26 17:00 keygen.py
-rw-r--r-- 1 evgeny evgeny   1541 Apr 29 10:28 key.public
-rw-r--r-- 1 evgeny evgeny   2053 Apr 29 10:28 key.secret
-rw-r--r-- 1 evgeny evgeny  24721 Apr 29 10:25 manual.ipynb
-rw-r--r-- 1 evgeny evgeny 399380 Apr 29 10:27 manual.pdf
-rw-r--r-- 1 evgeny evgeny 306562 Apr 25 16:42 mayday.png
-rw-r--r-- 1 evgeny evgeny    408 Apr 20 15:36 open.txt
drwxr-xr-x 2 evgeny evgeny   4096 Apr 26 17:09 __pycache__
-rw-r--r-- 1 evgeny evgeny   5100 Apr 25 19:02 rsa.py
-rw-r--r-- 1 evgeny evgeny     13 Apr 25 17:06 secret.crack
drwxr-xr-x 6 evgeny evgeny   4096 Apr 15 15:18 venv
```

In [19]:
```
# Попробую открыть файл

from PIL import Image

try:
    Image.open("encrypted.png")
except:
    print("Невозможно прочитать файл!")
```

```
Невозможно прочитать файл!
```

In [20]:
```
# Расшифрую изображение и сохраню в файл decrypted.png

rsa.decrypt("encrypted.png", "decrypted.png")
!ls -l
```

```
total 1388
-rw-r--r-- 1 evgeny evgeny 431919 Apr 25 19:45 analyze.ipynb
-rw-r--r-- 1 evgeny evgeny  35117 Apr 29 10:30 decrypted.png
-rw-r--r-- 1 evgeny evgeny    408 Apr 29 10:29 decrypted.txt
-rw-r--r-- 1 evgeny evgeny  88771 Apr 29 10:29 demo.ipynb
-rw-r--r-- 1 evgeny evgeny  35328 Apr 29 10:30 encrypted.png
-rw-r--r-- 1 evgeny evgeny    512 Apr 29 10:29 encrypted.txt
-rw-r--r-- 1 evgeny evgeny  35117 Apr 23 16:57 image.png
-rw-r--r-- 1 evgeny evgeny   4172 Apr 26 17:00 keygen.py
-rw-r--r-- 1 evgeny evgeny   1541 Apr 29 10:28 key.public
-rw-r--r-- 1 evgeny evgeny   2053 Apr 29 10:28 key.secret
-rw-r--r-- 1 evgeny evgeny  24721 Apr 29 10:25 manual.ipynb
-rw-r--r-- 1 evgeny evgeny 399380 Apr 29 10:27 manual.pdf
-rw-r--r-- 1 evgeny evgeny 306562 Apr 25 16:42 mayday.png
-rw-r--r-- 1 evgeny evgeny    408 Apr 20 15:36 open.txt
drwxr-xr-x 2 evgeny evgeny   4096 Apr 26 17:09 __pycache__
-rw-r--r-- 1 evgeny evgeny   5100 Apr 25 19:02 rsa.py
-rw-r--r-- 1 evgeny evgeny     13 Apr 25 17:06 secret.crack
drwxr-xr-x 6 evgeny evgeny   4096 Apr 15 15:18 venv
```

In [21]:
```
# Проверим корректность расшифровки

with open("image.png", "rb") as file:
    source_image = file.read()

with open("decrypted.png", "rb") as file:
    decrypted_image = file.read()

bool(source_image == decrypted_image)
```

Out[21]: True

In [22]:
```
# Попробую открыть расшифрованый файл

Image.open("decrypted.png")
```

Out[22]:

Приложение 3

Криптоанализ алгоритма RSA

In [1]:
```
# Сначала очистим директорию от артифактов предыдущих запусков кода

!rm -f encrypted.* decrypted.* *.public *.secret
!ls -l
```

```
total 2044
-rw-r--r-- 1 evgeny evgeny 432370 Apr 29 10:47 analyze.ipynb
-rw-r--r-- 1 evgeny evgeny 635921 Apr 29 10:40 analyze.pdf
-rw-r--r-- 1 evgeny evgeny 112888 Apr 29 10:44 bbyoda.jpeg
-rw-r--r-- 1 evgeny evgeny  89067 Apr 29 10:30 demo.ipynb
-rw-r--r-- 1 evgeny evgeny 317847 Apr 29 10:32 demo.pdf
-rw-r--r-- 1 evgeny evgeny  35117 Apr 23 16:57 image.png
-rw-r--r-- 1 evgeny evgeny   4172 Apr 26 17:00 keygen.py
-rw-r--r-- 1 evgeny evgeny  24721 Apr 29 10:25 manual.ipynb
-rw-r--r-- 1 evgeny evgeny 399380 Apr 29 10:27 manual.pdf
-rw-r--r-- 1 evgeny evgeny    408 Apr 20 15:36 open.txt
drwxr-xr-x 2 evgeny evgeny   4096 Apr 26 17:09 __pycache__
-rw-r--r-- 1 evgeny evgeny   5100 Apr 25 19:02 rsa.py
drwxr-xr-x 6 evgeny evgeny   4096 Apr 15 15:18 venv
```

In [2]:
```
# Сгенерирую игрушечный пример - пару ключей с префиксом small

from keygen import KeyGenerator

KeyGenerator(8, 8, "small")

# Проверим создались ли файлы с ключами
!ls -l
```

```
total 2052
-rw-r--r-- 1 evgeny evgeny 432370 Apr 29 10:47 analyze.ipynb
-rw-r--r-- 1 evgeny evgeny 635921 Apr 29 10:40 analyze.pdf
-rw-r--r-- 1 evgeny evgeny 112888 Apr 29 10:44 bbyoda.jpeg
-rw-r--r-- 1 evgeny evgeny  89067 Apr 29 10:30 demo.ipynb
-rw-r--r-- 1 evgeny evgeny 317847 Apr 29 10:32 demo.pdf
-rw-r--r-- 1 evgeny evgeny  35117 Apr 23 16:57 image.png
-rw-r--r-- 1 evgeny evgeny   4172 Apr 26 17:00 keygen.py
-rw-r--r-- 1 evgeny evgeny  24721 Apr 29 10:25 manual.ipynb
-rw-r--r-- 1 evgeny evgeny 399380 Apr 29 10:27 manual.pdf
-rw-r--r-- 1 evgeny evgeny    408 Apr 20 15:36 open.txt
drwxr-xr-x 2 evgeny evgeny   4096 Apr 26 17:09 __pycache__
-rw-r--r-- 1 evgeny evgeny   5100 Apr 25 19:02 rsa.py
-rw-r--r-- 1 evgeny evgeny     11 Apr 29 10:49 small.public
-rw-r--r-- 1 evgeny evgeny     13 Apr 29 10:49 small.secret
drwxr-xr-x 6 evgeny evgeny   4096 Apr 15 15:18 venv
```

In [3]:
```
# Вижу два ключа small.public и small.secret
# Измерю их длину и длину модуля

# Открытый ключ имеет длину 8 бит

with open("small.public", "r") as file:
    public_key, modul = file.read().split("/")

print({
    "public_key": public_key,
    "public_key_int": int(public_key, 0),
    "public_key_bin": bin(int(public_key, 0)),
    "public_key_len": len(bin(int(public_key, 0))[2:])
})
```

```
{'public_key': '0XBF', 'public_key_int': 191, 'public_key_bin': '0b
10111111', 'public_key_len': 8}
```

In [4]:
```
# Закрытый ключ имеет длину 14 бит

with open("small.secret", "r") as file:
    secret_key = file.read().split("/")[0]

print({
    "secret_key": secret_key,
    "secret_key_int": int(secret_key, 0),
    "secret_key_bin": bin(int(secret_key, 0)),
    "secret_key_len": len(bin(int(secret_key, 0))[2:])
})
```

```
{'secret_key': '0X20DB', 'secret_key_int': 8411, 'secret_key_bin':
'0b10000011011011', 'secret_key_len': 14}
```

In [5]:
```
# Модуль - 16 бит

print({
    "modul": modul,
    "modul_int": int(modul, 0),
    "modul_bin": bin(int(modul, 0)),
    "modul_len": len(bin(int(modul, 0))[2:])
})
```

```
{'modul': '0X8CF9', 'modul_int': 36089, 'modul_bin': '0b10001100111
11001', 'modul_len': 16}
```

In [6]:
```
# Ключи большой блины с правильными параметрами взломать
# за вменяемое время невозможно.
# Буду демонстрировать попытку взлома RSA малой длины и
# не верными параметрами p и q (с малой разницей)
#
# Параметры p и q рекомендуется подбирать так чтобы их разница
# - тоже была большим числом. Близкие по значению параметры p
# и q облегчили бы криптоанализ - поскольку уменьшили бы
# количество вариантов для перебора.

# Например:
#
# Зная длину модуля алгоритма n (16 бит) я могу предполагать что
# это значение появилось в результате перемножения неких p * q
# значения которые (грубо) могли варьироваться от 2^1 до 2^16
# отсюда следует что для факторизации числа n нужно было бы
# перебрать все значения попадающие в этот диапазон, то есть всего:

print(2**16 - 2**1)
```

```
65534
```

In [7]:
```
# А если параметры выбраны не верно и предположим оба числа p, q
# имеют длину 8, то диапазон перебора сужается до следующего
# от 2^7 до 2^8-1, то есть

print(2**8-2**7)
```

```
128
```

In [8]:
```
# То есть задача облегчилась примерно в 512 раз

# С увеличением разрядности ключей, разница в количестве
# вариантов перебора в случае с правильными и не правильными
# параметрами p,q пропорционально возрастает

print(65534 / 128)
```

```
511.984375
```

In [9]:
```
# При генерации пары ключей small в классе KeyGenerator
# я указал параметры 8 и 8. Они означают как раз длину
# параметров p, q

# Предположим, что злоумышленник видит длину модуля 16
# и рассчитывая что выбраны близкие параметры p, q перебирает
# только значения с длиной 8

modul_int = int(modul, 0)
p = None

for i in range(2**7, 2**8):
    if modul_int % i == 0:
        p = i
```

In [10]:
```
# Получили p и q
q = int(modul_int / p)

print("p = ", p)
print("q = ", q)
```

```
p =  239
q =  151
```

In [11]:
```
# Отсюда находим функцию Эйлера

phi = (p - 1)*(q - 1)
phi
```

Out[11]:
```
35700
```

In [12]:
```
# А теперь вычисляем и секретный ключ

public_key_int = int(public_key, 0)
secret_key = pow(public_key_int, -1, phi)
secret_key
```

Out[12]:
```
8411
```

In [13]:
```
# Проверяем правильно ли найден секретный ключ

import random

random_int = random.randrange(2, 1024)
random_int_encrypted = pow(random_int, public_key_int, modul_int)
random_int_decrypted = pow(random_int_encrypted, secret_key, modul_i

bool(random_int == random_int_decrypted)
```

Out[13]:
```
True
```

In [14]:
```
# Сохраним взломанный ключ в отдельный файл secret.crack
# в формате который принимает программа, то есть в 16м формате
# и с разделеием знаком /

with open("secret.crack", "w") as file:
    text = "{}/{}".format(hex(secret_key), hex(modul_int))
    file.write(text)

!ls -l
```

```
total 2056
-rw-r--r-- 1 evgeny evgeny 432370 Apr 29 10:47 analyze.ipynb
-rw-r--r-- 1 evgeny evgeny 635921 Apr 29 10:40 analyze.pdf
-rw-r--r-- 1 evgeny evgeny 112888 Apr 29 10:44 bbyoda.jpeg
-rw-r--r-- 1 evgeny evgeny  89067 Apr 29 10:30 demo.ipynb
-rw-r--r-- 1 evgeny evgeny 317847 Apr 29 10:32 demo.pdf
-rw-r--r-- 1 evgeny evgeny  35117 Apr 23 16:57 image.png
-rw-r--r-- 1 evgeny evgeny   4172 Apr 26 17:00 keygen.py
-rw-r--r-- 1 evgeny evgeny  24721 Apr 29 10:25 manual.ipynb
-rw-r--r-- 1 evgeny evgeny 399380 Apr 29 10:27 manual.pdf
-rw-r--r-- 1 evgeny evgeny    408 Apr 20 15:36 open.txt
drwxr-xr-x 2 evgeny evgeny   4096 Apr 26 17:09 __pycache__
-rw-r--r-- 1 evgeny evgeny   5100 Apr 25 19:02 rsa.py
-rw-r--r-- 1 evgeny evgeny     13 Apr 29 10:49 secret.crack
-rw-r--r-- 1 evgeny evgeny     11 Apr 29 10:49 small.public
-rw-r--r-- 1 evgeny evgeny     13 Apr 29 10:49 small.secret
drwxr-xr-x 6 evgeny evgeny   4096 Apr 15 15:18 venv
```

In [15]:
```
# Ради приличий давайте что-то зашифруем и расшифруем этими ключами
# Ну вот например картинку bbyoda.jpeg

!ls -l
```

```
total 2056
-rw-r--r-- 1 evgeny evgeny 432370 Apr 29 10:47 analyze.ipynb
-rw-r--r-- 1 evgeny evgeny 635921 Apr 29 10:40 analyze.pdf
-rw-r--r-- 1 evgeny evgeny 112888 Apr 29 10:44 bbyoda.jpeg
-rw-r--r-- 1 evgeny evgeny  89067 Apr 29 10:30 demo.ipynb
-rw-r--r-- 1 evgeny evgeny 317847 Apr 29 10:32 demo.pdf
-rw-r--r-- 1 evgeny evgeny  35117 Apr 23 16:57 image.png
-rw-r--r-- 1 evgeny evgeny   4172 Apr 26 17:00 keygen.py
-rw-r--r-- 1 evgeny evgeny  24721 Apr 29 10:25 manual.ipynb
-rw-r--r-- 1 evgeny evgeny 399380 Apr 29 10:27 manual.pdf
-rw-r--r-- 1 evgeny evgeny    408 Apr 20 15:36 open.txt
drwxr-xr-x 2 evgeny evgeny   4096 Apr 26 17:09 __pycache__
-rw-r--r-- 1 evgeny evgeny   5100 Apr 25 19:02 rsa.py
-rw-r--r-- 1 evgeny evgeny     13 Apr 29 10:49 secret.crack
-rw-r--r-- 1 evgeny evgeny     11 Apr 29 10:49 small.public
-rw-r--r-- 1 evgeny evgeny     13 Apr 29 10:49 small.secret
drwxr-xr-x 6 evgeny evgeny   4096 Apr 15 15:18 venv
```

In [16]:
```
# Импортируем класс RSA и передаем ему наш игрушечный ключ
# Затем шифруем изображение и сохраняем в отдельный файл bbyoda_encr

from rsa import RSA

encryptor = RSA(public_key_path="small.public")
encryptor.encrypt("bbyoda.jpeg", "bbyoda_encrypted.jpeg")
!ls -l
```

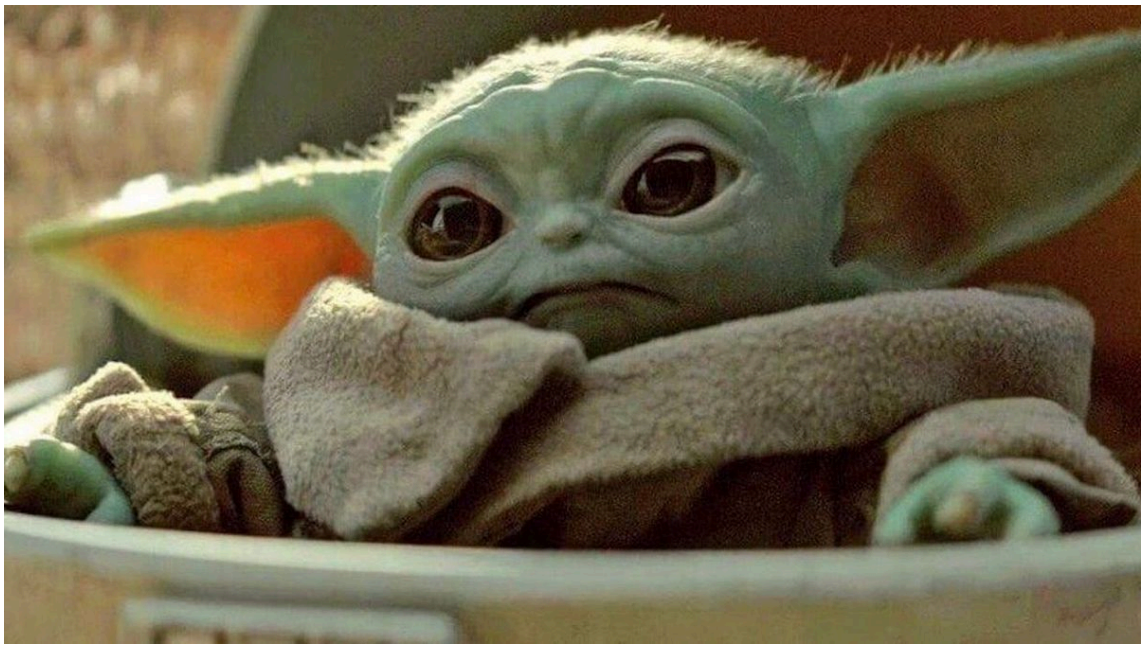
```
total 2176
-rw-r--r-- 1 evgeny evgeny 432370 Apr 29 10:47 analyze.ipynb
-rw-r--r-- 1 evgeny evgeny 635921 Apr 29 10:40 analyze.pdf
-rw-r--r-- 1 evgeny evgeny 120414 Apr 29 10:49 bbyoda_encrypted.jpe
g
-rw-r--r-- 1 evgeny evgeny 112888 Apr 29 10:44 bbyoda.jpeg
-rw-r--r-- 1 evgeny evgeny  89067 Apr 29 10:30 demo.ipynb
-rw-r--r-- 1 evgeny evgeny 317847 Apr 29 10:32 demo.pdf
-rw-r--r-- 1 evgeny evgeny  35117 Apr 23 16:57 image.png
-rw-r--r-- 1 evgeny evgeny   4172 Apr 26 17:00 keygen.py
-rw-r--r-- 1 evgeny evgeny  24721 Apr 29 10:25 manual.ipynb
-rw-r--r-- 1 evgeny evgeny 399380 Apr 29 10:27 manual.pdf
-rw-r--r-- 1 evgeny evgeny    408 Apr 20 15:36 open.txt
drwxr-xr-x 2 evgeny evgeny   4096 Apr 26 17:09 __pycache__
-rw-r--r-- 1 evgeny evgeny   5100 Apr 25 19:02 rsa.py
-rw-r--r-- 1 evgeny evgeny     13 Apr 29 10:49 secret.crack
-rw-r--r-- 1 evgeny evgeny     11 Apr 29 10:49 small.public
-rw-r--r-- 1 evgeny evgeny     13 Apr 29 10:49 small.secret
drwxr-xr-x 6 evgeny evgeny   4096 Apr 15 15:18 venv
```

In [17]:
```
# Создам отдельный объект для дешифрации и передам ему взломанный кл
# в файле secret.crack

decryptor = RSA(secret_key_path="secret.crack")
decryptor.decrypt("bbyoda_encrypted.jpeg", "bbyoda_decrypted.jpeg")
!ls -l
```

```
total 2288
-rw-r--r-- 1 evgeny evgeny 432370 Apr 29 10:47 analyze.ipynb
-rw-r--r-- 1 evgeny evgeny 635921 Apr 29 10:40 analyze.pdf
-rw-r--r-- 1 evgeny evgeny 112888 Apr 29 10:49 bbyoda_decrypted.jpe
g
-rw-r--r-- 1 evgeny evgeny 120414 Apr 29 10:49 bbyoda_encrypted.jpe
g
-rw-r--r-- 1 evgeny evgeny 112888 Apr 29 10:44 bbyoda.jpeg
-rw-r--r-- 1 evgeny evgeny  89067 Apr 29 10:30 demo.ipynb
-rw-r--r-- 1 evgeny evgeny 317847 Apr 29 10:32 demo.pdf
-rw-r--r-- 1 evgeny evgeny  35117 Apr 23 16:57 image.png
-rw-r--r-- 1 evgeny evgeny   4172 Apr 26 17:00 keygen.py
-rw-r--r-- 1 evgeny evgeny  24721 Apr 29 10:25 manual.ipynb
-rw-r--r-- 1 evgeny evgeny 399380 Apr 29 10:27 manual.pdf
-rw-r--r-- 1 evgeny evgeny    408 Apr 20 15:36 open.txt
drwxr-xr-x 2 evgeny evgeny   4096 Apr 26 17:09 __pycache__
-rw-r--r-- 1 evgeny evgeny   5100 Apr 25 19:02 rsa.py
-rw-r--r-- 1 evgeny evgeny     13 Apr 29 10:49 secret.crack
-rw-r--r-- 1 evgeny evgeny     11 Apr 29 10:49 small.public
-rw-r--r-- 1 evgeny evgeny     13 Apr 29 10:49 small.secret
drwxr-xr-x 6 evgeny evgeny   4096 Apr 15 15:18 venv
```

In [18]:
```
from PIL import Image

Image.open("bbyoda_decrypted.jpeg")
```

Out[18]:



Спасибо за внимание и с наступающими вас майскими праздниками!

And May 4th be with you