Московский авиационный институт
(национальный исследовательский университет)

Институт №8 «Информационные технологии и прикладная математика»

Кафедра 806 «Вычислительная математика и программирование»

Лабораторная работа №1 по курсу «Криптография»

Тема: Факторизация числа

Студент: А.С. Федоров

Преподаватель: А.В. Борисов

Группа: М8О-307Б-19

Дата:

Оценка:

Подпись:

Москва, 2022

Факторизация числа

Задача:

Разложить каждое из чисел n1 и n2 на нетривиальные сомножители.

Вариант 23 (23 mod 20 = 3):

n1=160706734943871920107793989624568987193024855869958859016714434
47466287976435079,

n2=684215004608788209511925294320580910987249828038503896263628671
854878834867990676121425471900938721830377798412511616753180888226
170565913014247435227790672540729012001595052680047771981415854555
644437749033484242941572168738338006086839673472700602434868065138
467260634466278850124520845045751767561017196287968876072175416665
051937939409039338744182433943913578137185996643388440141211792229
033804297598405773633300479208073155912043915548012252367280795532
5817

Описание

Факторизацией натурального числа называется его разложение в произведение простых множителей. Все алгоритмы делятся на экспоненциальные и субэкспоненциальные в зависимости от временной сложности. Не квантовых алгоритмов, решающих данную задачу за полиномиальное время, на данный момент, неизвестно. Однако, доказательства того, что такой алгоритм невозможен также нет.

Предположение о том, что для больших чисел задача факторизации является вычислительно сложной, лежит в основе широко используемых алгоритмов (например, RSA).

Ход работы

Для решения задачи были испробованы несколько алгоритмов. Наивный метод был откинут сразу же. Первым был использован метод факторизации Ферма. Алгоритм был реализован на языке Python ввиду наличия встроенной длинной арифметики.

Код программы:

```
def isSquare(n):
    return n == int(math.pow(round(math.sqrt(n)), 2))


def FermatAlgorithm(n):
    s = math.ceil(math.sqrt(n))
    attempt = int(math.pow(s, 2)) - n
    while (not isSquare(attempt)):
        # print(attempt)
        attempt = 2 * s + 1 + attempt
        s += 1

    x = int(math.sqrt(attempt + n))
    y = int(math.sqrt(attempt))
    return list[x + y, x - y]
```

Алгоритм оказался неэффективным для числа n1. На протяжении почти восьми часов он работал в сессии Google Colab и не дал результата. Судя по особенности алгоритма, можно сделать вывод, что искомые числа не находятся рядом друг с другом, что логично. Лишний раз все равно убедиться стоило.

Следующим алгоритмом был выбран вероятностный ρ-алгоритм Полларда. Реализован он был так же на Python, но в этот раз был запущен на собственном ПК.

Код программы:

```
def polard(N):
    x = random.randint(1, N - 2)
    y = 1
    i = 0
    stage = 2
    while math.gcd(N, abs(x - y)) == 1:
        if i == stage:
            y = x
            stage *= 2
        x = (x * x + 1) % N
        i += 1
    return math.gcd(N, abs(x - y))
```

Алгоритм снова оказался неэффективен. Даже для n1. Хоть в этот раз он работал суммарно около 24 часов, результата так и не было получено.

Далее было приято решение использовать более эффективные алгоритмы. Согласно Википедии метод квадратичного решета считается

наилучшим для факторизации чисел меньших 10^{110}. Однако, его суть оказалась слишком сложна для понимания. Поэтому была использована готовая онлайн-реализация ссылка: https://www.alpertron.com.ar/ECM.HTM. Ответ был получен за 11 с половиной минут, что просто поражает в сравнении с предыдущими попытками.

Решение для n1:

302406071694416266541999191429488194151 1,

531426945376504708993169157560356979468 9.

Хоть решение задачи факторизации n1 заняло немного времени, решение тем же методом аналогичной задачи для n2 оказалось неэффективным.

Было принято решение использовать обходные пути. На одной из лекций была дана подсказка, намекающая на несовершенность генерации пар простых чисел. Было выдвинуто два предположения. Первое: возможно при генерации пары, например для RSA, используется генератор случайных чисел, которому можно задать сид и получить нужную пару. Предполагалось, что сид как-то связан с номерами вариантов. Однако, если он не связан тривиальным образом, то подбор таких сидов становится неосуществимым. Вторым предположением было то, что сгенерированные числа имеют общие сомножители. Тогда достаточно перебрать все пары для необходимого варианта и найти gcd, что сработает быстрее. Из-за простоты проверки второй догадки было принято решение проверить ее первой. Сначала были проверены все пары чисел n1 с n1 нужного варианта. Совпадений не обнаружилось. Однако совпадение было найдено для чисел n2. Решение задачи можно считать найденным. Проверка осуществилась простой итерацией по массиву с печатью результата на языке Python.

Решение для n2:

217776024912858758188731783746431144809499905205019444178281665917809335919768764749887010991207046215801859194838919353900239984672068704380331525250665 23,

314182888076210905098503113311289284042961557580413672389012673726090265860807425522585724855577039050169471993543885695784856101609921434391375761091915806976896701960759304287145197092501562077955692673343267640427874689222961922379669981885730183388205600505354539481023820403203534688190427166986574883579.

Вывод

В ходе выполнения лабораторной работы я познакомился с постановкой и использованием задачи факторизации числа в криптографии. Также познакомился с некоторыми алгоритмами решения данной задачи. В частности, наивный алгоритм, метод Ферма, ρ-алгоритм Полларда. Познакомился с общими свойствами вычислительной сложности не квантовых алгоритмов факторизации. Однако ни один из изученных алгоритмов не дал результатов. Разложение n1 было получено алгоритмом квадратичного решета, понимание работы которого оказалось затруднительным. Решение же задачи факторизации числа n2 не было получено в лоб. Была использована подсказка, полученная на лекции.

Исходя из проделанной работы был сделан вывод, что решение в лоб практически всегда не результативно. Использование косвенной информации о задаче может быть гораздо более продуктивным.