Правительство Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования "Национальный исследовательский университет "Высшая школа экономики"

МИЭМ им. А.Н. Тихонова

Департамент прикладной математики

ОТЧЕТ

по дисциплине «Теоретико-числовые методы в криптографии»

Выполнили студенты гр. СКБ181

Скрипкин Павел

Овсянников Александр

Тема работы: «Алгоритм Диксона (Метод случайных квадратов Диксона)»

Москва 2022 г.

Введение

Алгоритм Диксона — алгоритм целочисленной факторизации общего назначения, в основу которого положена идея Лежандра, заключающаяся в поиске пары целых чисел x, y таких, что:

$$x^2 \equiv y^2 \ (mod\ n)$$
$$x \not\equiv \pm y (mod\ n)$$

Изначально, Морис Крайчик, работая над обобщением теоремы Ферма предложил вместо привычных для теоремы пар, удовлетворяющих уравнению $x^2 - y^2 = n$, искать пары чисел, удовлетворяющих $x^2 \equiv y^2 \ (mod\ n)$, что является более общим уравнением. Позднее, Диксон воспользовался наработками Крайчика и представил миру разработанный им метод факторизации, в последствие так же была рассчитана его вычислительная сложность:

$$O\left(\exp\left(2\sqrt{2}\sqrt{\log(n)\log(log(n))}\right)\right)$$

Асимптотически, этот алгоритм должен был быть намного быстрее, чем любой другой ранее проанализированный алгоритм разложения целых чисел на множители, так как все предыдущие алгоритмы требовали $O(n^a)$ операций, где a > 1/5, однако его практичность так и не была доказана, потому как к тому времени, когда он был предложен, уже существовали более практичные методы (например CFRAC).

В настоящем отчёте мы описываем и реализуем Алгоритм Диксона (Метод случайных квадратов Диксона), а так же рассмотрим класс алгоритмов, для которых можно доказать, что "почти для всех" чисел найдётся множитель целого числа n за $O\left(\exp\left(\beta\sqrt{\log(n)\log(log(n))}\right)\right)$ для некоторой константы $\beta > 0$.

Семейство Алгоритм

Мы рассмотрим общее семейство алгоритмов, для которых Диксон искал более быстрое решение. На тот момент одним из таких был алгоритм "Разложение на множители с помощью непрерывных дробей".

Пусть n - нечётное целое число, делящееся по крайней мере на два простых числа. Из теоремы Лежандра, мы знаем, что существуют целые числа x и y, которые относительно просты для n и такие, что $x^2 \equiv y^2 \ (mod\ n)$, но $x \not\equiv \pm y (mod\ n)$. Для таких целых чисел НОД(n, x + y) является собственным множителем n.

Поиск целых чисел x и y осуществляется в два этапа:
1) Сначала мы ищем квадраты z^2, которые соответствуют по (mod n) всем целым числам, простые множители которых лежат в некотором множестве P.
2) Затем мы используем соотношения между показателями в разложение на множители этих чисел для нахождения желаемых x и y.

Этот же процесс, но более детально (Алгоритм AL):

Шаг 1: L - представляет собой список целых чисел в диапазоне [1:n], $P = \{p_1, \dots, p_h\}$ - список простых чисел h < v, а B и Z изначально являются пустыми списками (Z будет проиндексирован B)

Шаг 2: если L пусто, то завершите работу (алгоритм неудачен); в противном случае пусть z будет первым членом в L, удалите z из L и перейдите к шагу 1

Шаг 3: пусть w - наименьший положительный остаток от $z^2 (mod\ n)$. Коэффициент $w' = \prod_i p_i^a$, где w' не имеет коэффициента в P. Если w' = 1, то перейдите к шагу 4; в противном случае перейдите к шагу 1

Шаг 4: пусть a $\leftarrow (a_1, \dots, a_h)$. Присоединяем «a» к «B» и z = za к «Z». Если список B содержит не более h элементов, то перейдите к шагу 2; в противном случае перейдите к шагу 5.

Шаг 5: Найдите первый вектор «c» в «B», который линейно зависит (mod 2) от более ранних векторов в B. Удалите «c» из B и z_c из Z. Вычислите коэффициенты $f_b = 0$ или 1 таким образом, чтобы:

$$c \equiv \sum_{b \in B} f_b b \ (mod\ 2)$$

Пусть:

$$d = (d_1, \dots, d_n) \leftarrow \frac{1}{2}(c + \sum f_b b) - \text{вектор целых чисел}$$

и перейти к шагу 6

Шаг 6: пусть $x \leftarrow z_c \prod_b z_b^{f_b}$ и $y \leftarrow \prod_i p_i^{d_i}$ (таким образом, что $x^2 \equiv \prod_i p_i^{2d_i} = y^2 (mod\ n)$). Если $x \equiv y$ или $-y\ (mod\ n)$, перейти к шагу 1, в другом случае вернуть НОД(n, x + y) – надлежащий множитель для n и выйти (алгоритм прошёл успешно).

Теперь предположим, что L имеет длину N и рассмотрим количество операций (сравнимых с арифметическими операциями между парами целых чисел размером n), которые участвуют в выполнении этого алгоритма. Если мы позволим Ni обозначить количество раз который AL выполняет Шаг i, тогда очевидно, что N + I > Nx > N2 > N3 и N4 = Ns = max(A/3 — h, 0). На шаге 3 для факторизации w требуется $O(h \ln(n))$ операций, поскольку каждый ai < ln(n). На этапе 5 определение “c” и вычисление коэффициентов f_b могут быть выполнены путем исключения по Гауссу и, таким образом, требуют не более 0(h3) операций. На шаге 6 НОД может быть вычислен за $O(\ln(n))$ операций. Следовательно, количество операций, выполняемых при выполнении AL:

$$N_2 O(1) + N_3 O(h \ln n) + N_4 O(1) + N_5 O(h^3) + N_6 O(h + \ln n) = O(N_2 h \ln n + N_5 h^3)$$

Проблема состоит в том, чтобы выбрать параметры v и A' таким образом, чтобы: (i) алгоритмы AL завершались успешно почти для всех списков L длины N, и (ii) количество требуемых операций было как можно меньше. Пусть $£(n, N)$ обозначает множество всех n^N списков длины N, состоящих из целых чисел в отрезке [1, n]. Тогда наш основной результат выглядит следующим образом:

Теоретическая часть

Теорема: пусть n - нечётное целое число, делящееся по крайней мере на два различных простых числа. Возьмём $v = \exp\{(2\ln n \ln\ln n)^{\frac{1}{2}}$ и $N = [v^2 + 1]$. Тогда среднее количество операций, требуемых для выполнения AL ($L \in £(\mathrm{n}, \mathrm{N})$), равно $O(\exp\{3(2\ln n \ln\ln n)\text{^}1/2\})$, а доля алгоритмов AL, для которых нельзя найти правильный коэффициент n, равна $O(N^{-\frac{1}{2}})$ (равномерно по n)

Доказательство: Начнём доказательство с общих лемм. Для любых положительных действительных чисел u, v пусть $\psi(u, v)$ обозначает число натуральных чисел $k \leq u$ у которых все простые делители < v.

Лемма 1: $\psi(v^k, v) \geq \binom{\pi(v) + k}{k}$ для каждого целого числа k > 1, где $\pi(v)$ – число простых чисел < v

Доказательство почти очевидно, поскольку биномиальный коэффициент подсчитывает количество способов выбора наборов до k целых чисел (допускающих повторения) из числа первых простых чисел $\pi(v)$. Хотя известны лучшие асимптотические оценки для $\psi(u, v)$, но они требуют условий для u и v, которые не могут быть проверены на выполнение в нашем случае.

Лемма 2: существует константа $c_0 > 0$, такая, что для всех натуральных чисел n и r и вещественных $v \geq n^{\frac{1}{2r}}$ выполняются условия

(i) $\mathrm{c}_0^{-1} \ln n \geq r \geq \ln\ln n$

(ii) все простые множители n > v вместе означают, что $|M(v)| \geq n(\ln n)^{-4r}$

Доказательство. Из (ii) следует, что целые числа в $T(v)$относительно просты до n, и поэтому мы можем разбить $T(v)$ на объединение непересекающихся подмножеств $T_i = (i = 1, \ldots, 2^d)$, соответствующие 2^d различным возможным значениям для χ. Пусть $T(v)$ обозначает число делителей t и записываем (соответственно, S_i) для суммы $\tau(t)^{-1}$, взятой по всем $t \leq \sqrt{n}$ с $t \in T(v)$ (соответственно, $t \in T_i$). Аналогично, пусть S' обозначает сумму $\tau(t)$ по всем $t \leq \sqrt{n}$ с $t \in T(v)$. Тогда, по неравенству Коши-Буняковского:

$$\psi\left(\sqrt{n}, v\right)^2 \leq SS'$$

И

$$S^2 \leq 2^d \sum_{i=1}^{2^d} S_i^2$$

Мы так же хорошо знаем, что:

$$S' \leq \sum_{t<\sqrt{n}} \tau(t) \leq \sqrt{n} \ln \sqrt{n} + \sqrt{n} < \sqrt{n} \ln n$$

для всех n достаточно больших (например, если $c_0 > 0$ и (i) выполняется).

С другой стороны, $\sum S_i^2 = \sum_{t \leq n} c(t)$, где $c(t) = \sum \tau(s)^{-1} \tau(s')^{-1}$, где последняя сумма берётся по всем парам (s, s') таким образом, что $ss' = t$, $s \leq \sqrt{n}$, $s' \leq \sqrt{n}$ и оба s и s' лежат в одном и том же T_i. Поскольку $ss' \in T_i$, значит $\chi(s) = \chi(s')$. Из вышесказанного следует, что $c(t) \neq 0$, подразумевает, что $t \in Q$. Более того, если $t \in Q$, то неравенство $\tau(t)\, \tau(s') \geq \tau(ss')$ показывает, что $c(t) < 1$. Таким образом, $\sum S_i^2$ является нижней границей числа $t \in Q$ с $t \in T(v)$. Следовательно,

$$|T(v) \cap Q| \geq \sum S_i^2$$

Объединив с $\psi\left(\sqrt{n}, v\right)^2 \leq SS'$, получаем:

$$|M(v)| = 2^d |T(v) \cap Q| \geq n^{-1} (\ln n)^{-2} \psi\left(\sqrt{n}, v\right)^4$$

Теперь условие (i) подразумевает, что $v \geq n^{\frac{1}{2r}} \geq e^{\frac{c_0}{2}}$ и $r \geq \ln c_0$. Следовательно, если c_0 выбрано достаточно большим, то (i) подразумевает, что v и r оба будут большими и поэтому:

$$\pi(v) \geq v(2 \ln v)^{-1} \quad \text{и} \quad r! \leq \left(\frac{r}{2}\right)^r$$

Из первой Леммы у нас есть:

$$\psi\left(\sqrt{n}, v\right) \geq \frac{\pi(v)^2}{r!} \geq v^r (r \ln v)^{-r} \geq \sqrt{n} \left(\frac{1}{2} \ln n\right)^{-r}$$

Из $|M(v)| = 2^d|T(v) \cap Q| \geq n^{-1}(\ln n)^{-2}\psi\left(\sqrt{n}, v\right)^4$ и (i), получаем:

$$|M(v)| \geq 2^{4r} n(\ln n)^{-4r-2} \geq n(\ln n)^{-4r}$$

Как и было предположено.

Теперь для каждого $L \in £(\mathrm{n}, \mathrm{N})$ мы определим $\sigma(L) \in £(\mathrm{n}, \mathrm{N})$ как список, чей i-ый член w задается $w \equiv z^2 (mod\ n)$, где z i-ый член списка L. Для каждого L_0, обозначим $[L_0]$ как множество всех $L \in £(\mathrm{n}, \mathrm{N})$ такое, что $\sigma(L) = \sigma(L_0)$.

Лемма 3: пусть n имеет факторизацию $n = q_1^{l_1} q_2^{l_2} \ldots q_d^{l_d}$ $(d \geq 2)$.Тогда для каждого целого числа k доля алгоритмов AL; $L \in £(\mathrm{n}, \mathrm{N})$, которые выполняют шаг 6 более k раз, составляет не более 2^{-k}.

Доказательство: допустим, что L является "плохим", если AL выполняет шаг 6 более k раз. Тогда достаточно показать, что если $[L_0]$ является плохим, то не более 2^{-k} списков в $[L_0]$ являются плохими. Запишем z_j для значения z_c, которое возникает, когда AL выполняет шаг 6 в j-ый раз, и предположим, что z_j первоначально встречался как i_j -ый член в L_0. Затем для каждого $L \in [L_0]$, член z в i-й позиции удовлетворяет $z^2 \equiv z_j^2 (mod\ n)$. Поскольку каждый элемент в Q имеет ровно 2^d квадратных корня (mod n), $[L_0]$ разбивается на 2^{dk} подмножеств одинакового размера, где L и L' лежат в одном и том же подмножестве тогда и только тогда, когда они имеют те же термины в позициях $i_1, \ldots, i_k$ соответственно. Однако, $L \in [L_0]$ является плохим тогда и только тогда, когда его члены в позициях $i_1, \ldots, i_k$ равны $\pm z_1, \ldots, \pm z_k$ поскольку значение для y на шаге 6 зависит только от $\sigma(L)$). Таким образом, доля плохих списков в $[L_0]$ равна $\frac{2^k}{2^{dk}} \leq 2^{-k}$, как и утверждалось.

Теперь рассмотрим основную теорему. Мы начнём с доказательства того, что второе утверждение теоремы выполняется при несколько более слабой гипотезе $n \geq N \geq 4hv$. Обозначим через X_L число w в $\sigma(L)$, лежащее в *T(v)*, и заметим, что X_L, как случайная величина в пространстве $\pounds(\mathrm{n}, \mathrm{N})$, имеет биномиальное распределение со средним значением λ_N и дисперсией $\lambda(1-\lambda)N$, где λ (вероятность событие $w \in T(v)$.) удовлетворяет $\lambda \geq (\ln n)^{-4r}$ для достаточно большого n по лемме 2. По нашему выбору v это показывает, что $\lambda \geq v^{-1}$. По неравенству Чебышёва доля $L \in \pounds(\mathrm{n}, \mathrm{N})$, для которого $X_L \leq \lambda N - c$, составляет не более $c^{-2}\lambda(1-\lambda)N$ для любого c > 0. В частности, принимая $c = \frac{1}{2}\lambda N$, мы находим, что доля алгоритмов AL, для которых $X_L < c$, меньше, чем $2c^{-1}$. Но, выбрав N, мы имеем $c \geq \frac{1}{2}v^{-1}hv = 2h$, если n достаточно велико. Следовательно, если $X_L > c$ и AL не удается найти множитель n, то AL должен выполнить шаг 4 не менее 2h раз и шаги 5 и 6 не менее h раз. Но тогда лемма 3 показывает, что доля AL $L \in \pounds(\mathrm{n}, \mathrm{N})$, которые являются неудачными и имеют $X_L > c$, составляет не более 2^{-h}. Таким образом, доля всех AL $L \in \pounds(\mathrm{n}, \mathrm{N})$, которые не в состоянии найти множитель n, составляет не более

$$2c^{-1} + 2^{-h} = O(vN^{-1}) + O(n^{-1}) = O(N^{-\frac{1}{2}})$$

Это доказывает второе утверждение теоремы. Чтобы завершить доказательство, напомним, что в разделе 2 мы показали, что количество операций, необходимых для выполнения AL, равно $O(N_2 h \ln n + N_5 h^3)$. Из того, что мы только что доказали, все, кроме $O(v^{-1})$ из AL, найдут правильный коэффициент n с не более чем 4hv + 2 выполнением шага 2. Таким образом, среднее значение $N_1 h \ln n$ равно

$$O\big(v^{-1}(N+1)\big)h \ln n + (4hv+2)n \ln n) = O(vh^2 \ln n) = O(v^3)$$

поскольку $h = O(\frac{v}{\ln v})$.

С другой стороны, по лемме 3, среднее значение $N_5 h^3$ равно $O(h^3)$.

Таким образом, среднее количество операций, требуемых AL, равно $O(v^3)$, как и утверждалось.

Практическая часть

Псевдокод

Шаг 1: пусть $x_i = ceil(\sqrt{n})$, пусть P = {p1, ...pt} - простой базис простых чисел, меньших параметра B. Инициализируем X, список x_i, V, список векторов $v_B(Q(x_i))$, и Q_{exp}, список экспонент в простом разложении $Q(x_i)$ относительно P, в пустые списки. Списки X, V и Q_{exp} будут проиндексированы таким образом, что $X[i] = x_i, V[i] = v_B(Q(x_i))$ и $Q_{exp}[i] = [a_1, a_2, \dots, a_t]$, где $Q(x_i) = p_1^{a_1}, \dots, p_t^{a_t}$

Шаг 2: если $x_i \geq n$ – завершить алгоритм неуспешно. В ином случае, вычислить $Q(x_i) = x_i^2 \ mod \ n$ и перейти к шагу 3

Шаг 3: определить, является ли $Q(x_i)$ B-гладким, последовательно разделив $Q(x_i)$ на все p_j, начиная с p_1, и переходя к следующему элементу в базисе, $p_j + 1$, только если p_j не делит текущее значение $Q(x_i)$. Если в конце этой процедуры $Q(x_i) = 1$, то $Q(x_i)$ является B-гладким. Если $Q(x_i) = p_1^{a_1}, \dots, p_t^{a_t}$ является B-гладким, добавьте x_i к X, $v_B(Q(x_i))$ к V и $= [a_1, a_2, \dots, a_t]$ к Q_{exp}. Если X, V и Q_{exp} имеют размер не менее t + 1, где t – размерность простого базиса, переходите к шагу 4. В противном случае увеличьте x и вернитесь к шагу 2.

Шаг 4: возьмем матрицу $A = \left[V[1]\ V[2] \dots V[t']\right]$, где $t' \geq t + 1$ и вычислим её ядро (т.е. набор векторов q, таких что $A_q = 0$). Перейдём к шагу 5.

Шаг 5: для каждого вектора $q \neq 0$ в ядре A инициализируйте x равным 1, а y_{exp} - списком нулей размером t'. Если элемент вектора q, скажем, q_j = 1, выполните x = (x ∗ X[j]) mod n и y_{exp} = y_{exp} + V [j]. Другими словами, если j-ая запись вектора q равна 1, то мы группируем пару (xj , Q(xj)) для вычислению пары x и y. У нас есть значение x, и для вычисления *y* мы выполняем $y = y * p_j^{y_{exp}\left(\frac{[j]}{2}\right)} \ mod \ n$. Если какой-либо из векторов в ядре A даёт нетривиальное решение для x и y, завершим программу успешно с выводом d1 = gcd(x + y, n) и d2 = $\frac{n}{d1}$, где n = d1d2. В противном случае увеличим x и вернёмся к шагу 2.

Код программы

Языком для реализации послужил SageMath:

```
def Dixon(n, B):
    prime_base = list(primes(1, B))
    prime_base_size = len(prime_base)
    X = []
    Q_x_exponents = []
    V = []
    x_i = int(sqrt(n)) + 1

    while x_i <= n:
        while True:
            vect = [0] * prime_base_size
            Q_x_i = (x_i * x_i) % n
            Q_x_i_exponents = [0] * prime_base_size
            for j in range(0, prime_base_size):
                p_j = prime_base[j]
                while Q_x_i % p_j == 0:
                    Q_x_i = Q_x_i // p_j
                    vect[j] = (vect[j] + 1) % 2
                    Q_x_i_exponents[j] = Q_x_i_exponents[j] + 1
                if Q_x_i == 1:
                    X.append(x_i)
                    V.append(vect)
                    Q_x_exponents.append(Q_x_i_exponents)
                    break
            if len(X) >= prime_base_size:
                break
            x_i = x_i + 1
        A = matrix(Zmod(2), V)
        K = A.kernel()
        K_size = len(K)
        for j in range(1, K_size):
            x = 1
            y = 1
            K_vect = K[j]
            y_exponent_vect = [0] * prime_base_size
            for k in range(0, len(X)):
                if K_vect[k] == 1:
                    x_factor = X[k]
                    for l in range(0,prime_base_size):
                        y_exponent_vect[l] = (y_exponent_vect[l] +
Q_x_exponents[k][l])
                    x = (x * x_factor) % n
            for l in range(0, prime_base_size):
                y = (y * pow(prime_base[l], y_exponent_vect[l] // 2)) % n
        if x != y and x != n - y:
            print(x, y, y_exponent_vect)
            d_1 = gcd(x + y, n)
            d_2 = n // d_1
            return (d_1, d_2)
        x_i += 1

    return None
```

Время работы алгоритма:

Тестируемое значение	Время выполнения
3317	0.031024694442749023
7535	0.007600307464599609
10738	0.08529973030090332
23449	1.5498180389404297
75336	0.005049228668212891
183352	0.01245570182800293
248775	13.688943862915039
248776	1.1989307403564453
521217	3.5037736892700195
852414	2.00112247467041
1378964	1.596555471420288
1378965	0.09749531745910645
2373711	0.6627767086029053
32458925	0.41768383979797363
784453589	2.6775689125061035
1263801330	0.6113300323486328
14919642914	1.0039305686950684
19971670372	2.365872621536255

Литература

https://git.miem.hse.ru/axelkenzo/nt-archive/-/blob/master/09%20-%20Dixon%20(1981,%20Asymptotically%20Fast%20Factorization%20Of%20Integers).pdf

http://maths.dk/teaching/courses/math357-spring2016/projects/factorization.pdf

https://eprint.iacr.org/2017/1087.pdf