Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Кафедра измерительных информационных технологий

Работа допущена к защите
Зав. кафедрой
_____ Г.Ф.Малыхина
«____» _____ 2015 г.

ВЫПУСКНАЯ РАБОТА БАКАЛАВРА

Исследование методов факторизации больших натуральных чисел в задачах оценки криптографической стойкости

Направление: 10.03.01 – Информационная безопасность

Выполнил студент гр. 43505/2 _____Р.Ф. Гарифуллин

Руководитель, к. т. н., доц. _____К.К. Семенов

Санкт-Петербург
2015 г.

ФГАОУ ВО «САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»
Институт информационных технологий и управления
Кафедра «Измерительные информационные технологии»

УТВЕРЖДАЮ
«_____» _____
2015 г.
Зав кафедрой Г.Ф. Малыхина
_____

ЗАДАНИЕ
на выпускную работу бакалавра

студенту Гарифуллину Руслану Фауиловичу группы 43505/2

1. Тема работы

Исследование методов факторизации больших натуральных чисел в задачах оценки криптографической стойкости.

2. Срок сдачи студентом законченной работы

08.06.15

3. Исходные данные к работе

Среди методов надлежащих исследованию: ρ –метод Полларда, P-1-метод Полларда метод эллиптических кривых, метод Ферма и др. В качестве тестов использовать псевдопростые числа полученные произведением двух простых чисел близкого порядка.

4. Содержание расчетно-пояснительной записки (перечень подлежащих разработке вопросов)

1) Обзор существующих методов факторизации больших натуральных чисел

2)Проверка соответствия задания исходным данным

3) Модификация существующих методов

4)Реализация полученных методов

5)Анализ полученных результатов

5. Перечень графического материала (с точным указанием обязательных чертежей)

Блок схемы алгоритмов диаграммы классов

6. Консультанты по проекту (с указанием относящихся к ним разделов работы)

Не требуется

7. Дата выдачи задания

10.02.15

*Руководитель*_____

*Задание принял к исполнению*_____

(дата)

_____

(подпись студента)

РЕФЕРАТ

на 40 стр., 9 рисунков, 1 приложение

КРИПТОГРАФИЯ, ПРОСТЫЕ ЧИСЛА, ФАКТОРИЗАЦИЯ, RSA, МЕТОДЫ ФАКТОРИЗАЦИИ, PARI GP.

В данной работе реализованы различные методы факторизации, а также их параллельные реализации с помощью математического пакета PARI GP. Получены зависимостей скорости разложения от разрядности числа сформированного путем перемножения близких сомножителей. Произведено сравнение реализованных методов и проведена грубая оценка стойкости RSA ключей используемых в настоящее время.

Содержание:

ВВЕДЕНИЕ

Факторизация больших нечетных составных чисел играет важную роль в криптографии и системах защиты информации. Как правило, для разбиения составных чисел, полученных в результате произведения двух больших простых сомножителей, требуются значительные вычислительные ресурсы и существенное время. Это обстоятельство используется в криптографических алгоритмах как защита от потенциальных попыток взлома: для создания и перемножения двух простых чисел большой разрядности не требуется сколько-нибудь существенных затрат, а их факторизация – длительный и трудоемкий процесс. На сегодняшний день не существует алгоритмов факторизации очень больших чисел, позволяющих выполнить разбиение на множители за практически приемлемое время.

Развитие вычислительной техники позволяет эффективно решать многие вычислительные сложные задачи экстенсивно, используя дешевые вычислительные мощности, а не быстрые алгоритмы. В самое последнее время в связи с развитием облачных и кластерных технологий существенно упростился доступ к суперкомпьютерам для рядовых пользователей. При этом обеспечивается конфиденциальность решаемой задачи.

Основной задачей данной работы является анализ, реализация и исследование возможностей алгоритмов факторизации больших чисел для использования для взлома ключей криптографического алгоритма RSA и алгоритмов на его основе.

В силу трудоемкости задачи факторизации разработанные к сегодняшнему дню методы ее выполнения могут быть подвергнуты модификациям, позволяющим ускорить их работу. Настоящая работа повсящена оценке возможностей и выигрышей от расспараллеливания процессов вычислений в методах факторизации.

1 МЕТОДЫ ФАКТОРИЗАЦИИ НАТУРАЛЬНЫХ ЧИСЕЛ

Факторизация – это процесс разложения натурального числа на простые сомножители. Факторизация является вычислительно сложной задачей, так в настоящий момент не существует эффективного не квантового алгоритма факторизации, но и доказательства, что не существует решения этой задачи за полиномиальное время, также нет [5]. Задача факторизации натуральных чисел интересовала математиков давно, но особый интерес появился лишь с появлением криптографического алгоритма шифрования RSA, эксплуатирующего вычислительную сложность этой задачи.

В 1977 г. Рональд Райвест, Ади Шамир и Леонард Адлеман опубликовали в журнале «Scientific American» новый алгоритм шифрования [1]. Для того, что быть уверенными в криптостойкости своего алгоритма шифрования, они провели конкурс: в журнале был опубликован список из 42 тестовых чисел различной разрядности, за факторизацию каждого числа была назначена премия. Так, например, за факторизацию 129-значного десятичного числа, было положено вознаграждение в размере 100$ [2]. Алгоритм шифрования RSA основывается на вычислительной сложности задачи факторизации больших чисел. Открытый ключ в этом алгоритме генерируется путем перемножения двух больших простых сомножителей. Одним из методов взлома данного алгоритма является факторизация открытого ключа. Для решения данной задачи в работу включились известные математики, криптографы и специалисты по теории чисел. В результате их работы были созданы многие алгоритмы факторизации, такие как ρ – метод и $(p-1)$ – метод Полларда, метод эллиптических кривых, метод квадратичного решета и метод решета числового поля и другие. В настоящий момент лишь 14 чисел из списка было факторизовано.

Последний рекорд по факторизации был поставлен в 2009 году, группой европейских ученых, под руководством Торштена Кляйнъюнга, им

удалось факторизовать 768-значное десятичное число с помощью метода решета числового поля [3].

Во многих задачах шифрования и обеспечения безопасной передачи данных используются ключи существенно меньшей длины (например, из-за ограничений, накладываемых на пропускную способность используемых каналов связи). Настоящая работа посвящена изучению возможностей использования параллельного исполнения методов факторизации для атак на ключи RSA.

В работе рассмотрены классические алгоритмы факторизации, так как именно они являются фундаментом, на основе которого строятся составные программные модули и модификации классических алгоритмов для решения задач факторизации. Среди них:

- метод Ферма,
- ρ -метод Полларда,
- p-1 метод Полларда,
- метод Шермана-Лемана,
- метод Шенкса,
- метод эллиптических кривых.

Цель данной работы – реализовать параллельные алгоритмы факторизации, получить зависимости скорости разложения от разрядности числа сформированного путем перемножения близких сомножителей, оценить эффективность реализованных алгоритмов, а также систематизировать полученные результаты.

1.1 Метод Ферма

Метод был предложен Пьером Ферма в 1643 году [4].

Пусть n целое число, которое является произведением двух простых сомножителей p и q. Основная идея факторизации числа n заключается в поиске таких пар натуральных чисел x и y, которые бы удовлетворяли условию $x^2 - y^2 = n$, то есть $(x-y)(x+y) = n$ где $(x-y)$ и $(x+y)$ являются нетривиальными множителями числа n ,если они не равны 1 и n.[4] Метод Ферма может быть описан следующим псевдокодом.

На входе: нечетное число n

На выходе: нетривиальные делители числа n

1. $x \leftarrow [\sqrt{n}]$
2. $y \leftarrow x^2 - n$
3. ***while*** $isSquare(y) \neq$ ***true do***
4. $x \leftarrow x + 1$
5. $y \leftarrow x^2 - n$
6. **end *while***
7. $s \leftarrow x - \sqrt{y}$
8. $t \leftarrow x + \sqrt{y}$
9. ***if*** $s \neq 1$ ***and*** $s \neq n$ ***then***
10. ***return*** s, t
11. ***end if***

Функция $isSquare(y)$ возвращает true, если y является полным квадратом.

Быстрее всего данный метод будет работать, если множители p и q очень близки друг к другу, именно поэтому не рекомендуется генерировать RSA ключи на основе близких друг к другу простых чисел. К недостаткам данного метода относится тот факт, что в случае, когда множители p и q близки по значению к 1 и n алгоритм будет работать хуже, чем метод пробных делений, что отмечается, например, в работе [5, с. 53-54].

Для того, чтобы ускорить данный алгоритм, можно перед использованием метода Ферма, выполнить пробное деление числа n на простые числа от 2 до некоторой константы B, исключая малые делители

числа n (это и есть метод пробных делений). Также для ускорения работы можно обратиться к параллельным вычислениям.

Основная идея параллельной реализации данного метода заключается в выборе случайного значения x, для каждого отдельного параллельного процесса, каждый из которых будет выполнять алгоритм, изложенный выше. Как только один из процессов найдет значения p и q, нужно завершить работу всех запущенный процессов.

1.2 ρ –метод Полларда

Идея метод заключается в следующем [4, с. 209].

Пусть n – факторизуемое число, $n = p \cdot q$, где p и q простые сомножители. Пусть есть некоторый многочлен f степени не ниже двойки, по модулю n. Например, $f(x) = x^2 + 1(mod\ n)$. Выбираем случайный элемент x_0 от 0 до $n - 1$.

Далее строим последовательность $x_0, x_1, x_2 \ldots$ такую, что $x_{i+1} = f(x_i)$, пока не найдем такие числа i и j, что $i < j$ и $x_i = x_j$. Такие числа существуют и последовательность с некоторого номера будет зацикливаться. Если p – простой делитель числа n и $x_i = x_j(mod\ n)$, то разность $x_i - x_j$ делится на p и $p = \text{НОД}(x_i - x_j, n) > 1$, то есть если $p \neq n$ и $p > 1$, то нетривиальный делитель найден. [4]

ρ-метод Полларда может быть описан следующим псевдокодом:

На входе: нечетное число n

На выходе: нетривиальные делитель числа n

1. $x \leftarrow rand(n - 2)$
2. $y \leftarrow 1$
3. $stage \leftarrow 2$
4. ***while*** $\gcd([(x - y)], n) = 1$ ***do***
5. ***if*** $i = stage$ ***then***
6. $y \leftarrow x$
7. $stage \leftarrow stage * 2$

8. ***end if***
9. $x \leftarrow (x^2 + 1)(mod\ n)$
10. $i \leftarrow i + 1$
11. ***end while***
12. **return** $\gcd([(x - y)], n)$

Функция rand(n) возвращает случайное число от 1 до n. Функция gcd() возвращает наибольший общий делитель.

Алгоритм можно распараллелить очевидным образом, нужно выбирать отдельное случайное число x для каждого отдельного процесса. Как только один из процессов найдет значения p и q, нужно завершить работу всех запущенный процессов.

1.3 P-1-метод Полларда

Метод был разработан Джоном Поллардом в 1974 г.[5].

Идея метод заключается в следующем [5, с. 54-57].

Пусть n – факторизуемое число, p – простой делитель n. Согласно малой теореме Ферма для любого a, $1 \leq a < p$ будет выполняться условие $a^{p-1} \equiv 1(mod\ p)$. Возьмем натуральное произвольное число M, кратное $p - 1$ то есть если $M = (p - 1) * k$ то $a^M = (a^{p-1})^k \equiv 1^k \equiv 1(mod\ p)$. Это эквивалентно $a^M - 1 = pr$ для некоторого целого r. Отсюда p является делителем n и совпадает с НОД(a^M, n) если $a^M - 1 < n$. Пусть $p - 1 = p_1^{r_1} * p_2^{r_2} * \ldots * p_t^{r_t}$ Основная идея p-1 метода Полларда заключается в выборе числа M в виде произведения как можно большего числа сомножителей или их степеней так, чтобы M делилось на каждый сомножитель $p_i^{r_i}$. Тогда НОД$(a^M - 1, n)$ даст искомый делитель.

p-1-метод Полларда может быть описан следующим псевдокодом

На входе: нечетное число n

На выходе: нетривиальные делитель числа n

1. $a \leftarrow 2$
2. $j \leftarrow 2$
3. ***while*** $j \leq B$ ***do***
4. $a \leftarrow a^j (mod\ n)$

5. $d \leftarrow \gcd(a-1, n)$
6. ***if*** $1 < d < n$ **then**
7. **return** d
8. **end if**
9. $j \leftarrow j + 1$
10. ***end while***
11. **return** FAIL

Идея параллельной реализации заключается в одновременном выполнении алгоритма несколькими потоками до нахождения множителей числа n.

1.4 Метод Лемана

Метод был впервые предложен в 1974 год математиком Шерманом Леманом. [5]. Метод на первом этапе своей работы пытается найти простые делители числа n от 2 до $\left[\sqrt[3]{n}\right]$. На втором этапе происходит перебор значений k и d для проверки условий следующей теоремы: [5].

Пусть n – составное число, которое является произведением двух простых чисел p и q, удовлетворяющих неравенству $n^{1/3} < p < q < n^{2/3}$. Тогда найдутся натуральные числа x, y и $k \geq 1$, такие, что выполняются следующие условия:

1. $x^2 - y^2 = 4kn$ при $k < n^{1/3}$,

2. $0 \leq x - \left[\sqrt{4kn}\right] < \frac{n^{1/6}}{4\sqrt{k}} + 1$

Если делитель числа n не найден, то n – простое число, именно поэтому этот алгоритм может быть использован для детерминированной проверки чисел на простоту. Данный алгоритм более подробно рассмотрен в статье [7, с. 637-646].

Метод Лемана может быть описан следующим псевдокодом:

На входе: нечетное число n

На выходе: нетривиальный делитель n

1. ***for*** $a \leftarrow 2$ ***to*** $\left[\sqrt[3]{n}\right]$ ***do***
2. ***if*** $a|n$ ***then***

3. **return** a
4. **end if**
5. **end for**
6. **for** $k \leftarrow 1$ **to** $\left[\sqrt[3]{n}\right]$ **do**
7. **for** $d \leftarrow 0$ **to** $\left[\frac{\sqrt[6]{n}}{4\sqrt{k}}\right] + 1$ **do**
8. **if** $isSquare\left(\left(\left[\sqrt{4kn}\right] + d\right)^2 - 4kn\right) = true$ **then**
9. $A \leftarrow \left[\sqrt{4kn}\right] + d$
10. $B \leftarrow \sqrt{A^2 - 4kn}$
11. **if** $1 < \gcd(A + B, n) < n$ **then**
12. **return** $\gcd(A + B, n)$
13. **else if** $1 < \gcd(A - B, n) < n$ **then**
14. **return** $\gcd(A - B, n)$
15. **end if**
16. **end if**
17. **end for**
18. **end for**

Функция $isSquare(y)$ возвращает true, если y является полным квадратом. Функция gcd() возвращает наибольший общий делитель.

Одним из способов ускорения алгоритма является его распараллеливание. Идея распараллеливания метода аналогична методу Ферма. . На первом шаге каждый процесс работает со своими значениями переменной a, т.е проверяет делимость числа n на a из промежутка от 2 до $\sqrt[3]{n}$. На втором шаге – свой диапазон чисел k , вычисляя нетривиальный делитель числа n при k от 1 до $\sqrt[3]{n}$.

1.5 Метод Шенкса

Метод был разработан Даниелем Шенксом в 1975 г.[5] Метод основан на квадратичных бинарных формах.

Полином от двух переменных называют квадратичной бинарной формой:[5] $f(\mathrm{x},\mathrm{y}) = ax^2 + bxy + cy^2 = (x\ y)\begin{pmatrix} a & b \\ 0 & c \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix}$

Квадратичную бинарную форму принято обозначать в более кратком виде: $f = (a, b, c)$. Дискриминантом (a, b, c). является число $D = b^2 - 4ac$. В методе Шенкса используются только формы с положительным дискриминантом.

Две квадратичные формы $f = (a, b, c)$ и $g = (p, q, r)$ называются эквивалентными, если найдется целочисленная матрица $\begin{pmatrix} \alpha & \beta \\ \gamma & \delta \end{pmatrix}$ с определителем, равным 1, преобразовывающая матрицу f в матрицу g: $\begin{pmatrix} p & q \\ 0 & r \end{pmatrix} = \begin{pmatrix} \alpha & \gamma \\ \beta & \delta \end{pmatrix} \begin{pmatrix} a & b \\ 0 & c \end{pmatrix} \begin{pmatrix} \alpha & \beta \\ \gamma & \delta \end{pmatrix}$

Форма вида (k, kn, c) называется неоднозначной. Если форма – неоднозначна, то ее определитель делится на k: $D = (\mathrm{kn})^2 - 4kc = k(\mathrm{kn}^2 - 4c)$. Идея метода Шенкса состоит в сопоставлении числу n, которое надо разложить, квадратичной бинарной формы f с дискриминантом $D = 4n$, с которой потом выполняется серия эквивалентных преобразований и переход от формы f к неоднозначной форме *(a0, b0, c0)*. Тогда, *Н.О.Д.(a0, b0)* будет являться делителем n [5].

Хотя теоретическая часть алгоритма связана с эквивалентными преобразованиями квадратичных форм, практическая часть алгоритма выполняется по формулам вычисления коэффициентов P, Q и r без обращения к квадратичным формам.

Более подробно алгоритм можно представить в виде следующего псевдокода:

Вход: Составное число n, маленький множитель b.

Выход: Нетривиальный делитель n или FAIL, если процесс факторизации закончился неудачно.

1. ***if*** $isSquare(n)$ ***then***
2. ***return*** $\sqrt{n}$
3. ***end if***
4. $P_0 = 0$;
5. $Q_0 = 1$;
6. $r_0 = P_1 = [\sqrt{bn}]$;
7. $Q_1 = bn - {r_0}^2$;
8. $r_1 = [2 * r_0/Q_1]$

9. $\boldsymbol{repeat}$
10. $P_k = r_{k-1} * Q_{k-1} - P_{k-1};$
11. $Q_k = Q_{k-2} + (P_{k-1} - P_k) * r_{k-1};$
12. $r_k = \left[\frac{P_k + [\sqrt{bn}]}{Q_k}\right]$
13. $\boldsymbol{until}\ isSquare(Q_k) == false$
14. $P'_0 = -P_k;$
15. $Q'_0 = \sqrt{Q_k};$
16. $r'_0 = \left[\frac{P'_0 + [\sqrt{bn}]}{Q'_0}\right]$
17. $P'_1 = r'_0 * Q'_0 - P'_0;$
18. $Q'_1 = (ln - {P'_1}^2)/Q'_0;$
19. $r'_1 = \left[\frac{P'_1 + [\sqrt{bn}]}{Q'_1}\right]$
20. $\boldsymbol{repeat}$
21. $P'_j = r'_{j-1} * Q'_{j-1} - P'_{j-1};$
22. $Q'_j = Q'_{j-2} + (P'_{j-1} - P'_j) * r'_{j-1};$
23. $r'_j = \left[\frac{P'_j + [\sqrt{bn}]}{Q'_j}\right]$
24. $\boldsymbol{until}\ P'_j \neq P'_{j+1}$
25. $\boldsymbol{if}\ 1 < \gcd(Q'_j, n) < n\ \boldsymbol{then}$
26. $\boldsymbol{return}\ \gcd(Q'_j, n)$
27. $\boldsymbol{end\ if}$
28. $\boldsymbol{return}$ FAIL

Функция $isSquare(y)$ возвращает true, если y является полным квадратом. Функция gcd() возвращает наибольший общий делитель.

Если делитель не найден, то следует взять другое значение переменной b и повторить алгоритм. Идея параллельной реализации метода заключается в использовании разных значений переменной b в разных потоках и полного выполнения алгоритма, описанного выше, для каждого отдельного потока. Как только один из потоков находит нетривиальный делитель, все остальные потоки должны завершить работу.

1.6 Метод эллиптических кривых (метод Ленстры)

Метод находится на третьем месте по скорости факторизации, работая медленнее лишь метода квадратичного решета и метода решета числового

поля.[5] На практике часто используется для нахождения небольших простых делителей числа в составных программных модулях по факторизации целых чисел.[5]

Пусть есть конечное поле характеристик $F_q, q = p^k$ при $p \geq 2$. Тогда эллиптической кривой над полем F_q называют множество точек $(x, y) \in F_q \oplus F_q$, которые удовлетворяют уравнению $y^2 + ay + b = x^3 + \mathrm{c}x^2 + \mathrm{d}x + e\ (mod\ q)$.

Если характеристика поля $p \geq 3$, то эллиптическую кривую можно представить в виде $y^2 = x^3 + \mathrm{a}x + b(mod\ q)$ где $a, b \in F_q$ и $4a^3 + 27b^2 \neq 0$. Также следует отметить, что к множеству точек на эллиптической кривой добавляется специальная точка, называемая точкой бесконечности, которая обозначается ∞.

На множестве точек эллиптической кривой можно определить операцию сложения + с точкой ∞ в качестве нуля.

Пусть точка $P(x, y)$ принадлежит эллиптической кривой, тогда обратной к точке P является точка – $P(x, -y)$. $P + (-P) = \infty$.

Сумма точек $P(x_1, y_1)$ и $Q(x_2, y_2)$, где $P \neq -Q$ вычисляется по формулам: $x_3 = \lambda^2 - x_1 - x_2$, $y_3 = \lambda(x_1 - x_3) - y_1$. При $P \neq Q, \lambda = \frac{y_2 - y_1}{x_2 - x_1}$ а при $P = Q$, $\lambda = \frac{3\,x_1^2 + a}{2y_1}$.

Основная идея метода факторизации Ленстры (метода эллиптических кривых) заключается в использовании псевдокривых вида $y^2 = x^3 + \mathrm{a}x + b$ так как в них не всегда выполнимы операции нахождения обратного элемента необходимые для нахождения суммы точек кривой. Ленстра заметил, при невозможности нахождения суммы двух точек $P(x_1, y_1)$ и $Q(x_2, y_2)$, разность координат $x_2 - x_1$ должна равняться 0 по модулю одного из делителей числа n. Тогда вычисление Н.О.Д.($x_2 - x_1, n$) дает искомый множитель числа n.

Весь метод сводится к выбору произвольной базовой точки P и домножении ее на всевозможные простые числа и их делители до

определенной границы B, пока не получим $kP_0 = \infty(mod\ p)$ p –делитель числа n.

Стоит также отметить, что нет единой формулы для нахождения кратной точки на эллиптической кривой, поэтому вся задача, поиска кратных точек сводится к операции сложения по модулю и удвоению исходной точки.[5]

Метод эллиптический кривых может быть представлен в виде следующего псевдокода:

На входе: нечетное число n, границы $B1, B2 \in \mathbb{N}$ Выбираем случайную эллиптическую кривую $E_{a,b}\ mod\ n$ и точку $P_0 = (x_0, y_0)$ на ней.

1. $k \leftarrow 1$
2. $m \leftarrow 0$
3. $\boldsymbol{for}\ r -$ простое $r \leftarrow 2\ \boldsymbol{to}\ B2\ \boldsymbol{do}$
4. $\quad \boldsymbol{while}\ r^m \leq B1 + 2\sqrt{B1} + 1\ \boldsymbol{do}$
5. $\quad\quad m \leftarrow m + 1$
6. $\quad \boldsymbol{end\ while}$
7. $\quad k \leftarrow k \times r^m$
8. $\quad m \leftarrow 0$
9. $\boldsymbol{end\ for}$
10. $d \leftarrow compute(kP_0)$
11. $\boldsymbol{if}\ 1 < d < n\ \boldsymbol{then}$
12. $\boldsymbol{return}\ d$
13. $\boldsymbol{end\ if}$
14. $\boldsymbol{return}\ FAIL$

Функция $compute(kP_0)$ возвращает делитель числа n в ходе вычисления кратного k точки P_0. Если алгоритм завершился неудачно, нужно выбрать другую эллиптическую кривую $E_{a,b}\ mod\ n$ и точку $P_0 = (x_0, y_0)$ на ней. Идея параллельной реализации заключается в выборе разных эллиптических кривых для каждого отдельного процесса и выполнении алгоритма, описанного выше.

2 РЕАЛИЗАЦИЯ АЛГОРИТМОВ ФАКТОРИЗАЦИИ

Для реализации вышеописанных алгоритмов была выбрана математическая система Pari/GP [14]. Математический пакет Pari/GP был выбран не случайно, в нем хорошо развита «длинная» арифметика, позволяющая работать с большими числами, он является бесплатным, использует совсем небольшое количество памяти для вычислений, в нем есть функции для параллельного вычисления, а также обладает большим набором математических функций для быстрых вычислений в теории чисел. Именно поэтому система имеет хорошую репутацию и определенную популярность в научной среде. По данным Google Scholar за 2015 год было опубликовано более трехсот тематических статей, связанных с вычислениями на Pari/GP. Pari/GP была разработана 1985 году, командой разработчиков, которую возглавлял Henri Cohen в университете Bordeaux I [9].

Пакет PARI GP составлен из двух частей. PARI является библиотекой, которую можно использовать с другими языками программирования, такими как C++, Java, Perl, Python. GP же является интерактивной оболочкой, в которой есть собственный C подобный язык, также есть компилятор, позволяющий преобразовать GP код в код C++.

Для реализации описанных методов была выбрана интерактивная оболочка GP, так как в ее собственном языке есть функции для параллельного вычисления, что является немаловажным фактором для выполнения данной работы.

Для параллельных вычислений в работе используется две основные функции **parfor** и **parforprime**.

Работа с параллельным циклом **parfor**(i = a,{b},expr1,{j},{expr2}) имеет следующие особенности.

Каждый отдельный поток выбирает случайное значение i в промежутке от a до b, и начинает выполнять функцию, вставленную на место expr1, если у нее есть возвращаемое значение, то переменная j примет это значение

после вычисления функции одним из потоков. Для того, чтобы завершить параллельный цикл, нужно вставить условие выхода из него в expr2. Параметры в фигурных скобках можно опустить. Цикл **parforprime** имеет те же особенности разница лишь в том, что он выбирает простые числа i в промежутке от a до b. Именно из-за этих особенностей параллельные реализации методов описанных алгоритмов приходилось разделять на отдельные функции. Также в каждом методе была добавлена детерминированная проверка простоты полученного множителя. Она реализована в GP функцией isprime(n), если этот множитель являлся не простым, алгоритм рекурсивно будет раскладывать данный множитель до нахождения всех простых сомножителей.

2.1 Список используемых математических функций GP

isprime(x) – детерминированная проверка простоты, возвращаемое значение равно 1, если число *x* является простым, иначе равно 0.

round(x) – функция округления числа *x* до целого значения.

issquare(n) – если число n является полным квадратом функция возвращает 1, иначе 0.

sqrt(x) – функция возвращает квадратный корень числа x.

sqrt(x,n) – функция возвращает корень степени n заданного числа x.

gcd(x,y) – функция возвращает наибольший общий делитель чисел x и y.

binary(x) – возвращает вектор, который является представлением заданного числа x в двоичном виде.

length(v) – функция возвращает длину вектора v.

floor(n) – функция возвращает целую часть числа n.

log(x) – функция возвращает натуральный логарифм числа x

ellmul(E,z,n) − функция вычисления кратной точки z на эллиптической кривой E, если точка не найдена то выдается сообщение об ошибке.

ellinit(a,b) − функция инициализации эллиптической кривой вида $y^2 = x^3 + ax + b$

2.2 Метод Ферма

С реализацией этого алгоритма трудностей не возникло, обычная реализация полностью соответствует псевдокоду описанному ранее.

Модифицированный распараллеленный код метода содержит следующие изменения: пробное деление до границы B, которая выбирается по формуле: $[\sqrt{e^{\sqrt{2\ln(n)\ln(\ln(n))}}}]$ [5]. Пробное деление исполняется параллельно с использованием цикла parforprime, в котором параллельные потоки случайно выбирают простое число a в промежутке от 2 до B и проверяют делимость числа n на a. Если число делится на число a, то найден его множитель. Также были распараллелены основные вычисления этого метода параллельным циклом parfor. Из-за особенности выполнения параллельных вычисления таким циклом пришлось разбить метод на четыре отдельных функции, которые повторяют основные идеи простой реализации метода Ферма. С реализацией данного метода на Pari/GP можно ознакомиться в Приложении 1.

2.3 P-1 метод Полларда

Метод является вероятностным, поэтому было принято решение, что метод будет выполнять свою работу до получения искомого делителя. В целом обычная реализация ничем не отличается от псевдокода, представленного выше. Параллельная реализация мало отличается от обычной, она разбита на несколько функций из-за особенностей цикла parfor и прочих особенностей не содержит. С реализацией данного метода на Pari/GP можно ознакомиться в Приложении 1.

2.4 Метод Шенкса

Во время реализации метода Шенкса была обнаружена следующая проблема: в псевдокоде, представленном ранее по источнику [10], указывается, что при вычислении коэффициента r должно происходить округление его до целой части, но если делать именно округление, то данный метод не будет работать: он будет попадать в бесконечный цикл на втором этапе данного

метода. Благодаря примеру из [5, с. 78], было выявлено, что коэффициент r необходимо не округлять, а брать его целую часть. Такое решение все-таки не дало полного избавления от зацикливания. Алгоритм продолжал работать бесконечно долго в некоторых очень редких случаях, поэтому во второй части этого метода было добавлено ограничение в 1000 итераций, которое хоть и дало некоторое замедление в работе, но позволило использовать метод для разложения числа n за конечное время.

Параллельная реализация не сильно отличается от метода, соответствующего последовательному исполнению алгоритма, в ней присутствует лишь случайный выбор множителя b для каждого отдельного потока и вызов обычного метода из цикла parfor. С реализацией данного метода на Pari/GP можно ознакомиться в Приложении 1.

2.5 Метод эллиптических кривых

Благодаря тому, что в PARI GP есть функции для работы с эллиптическими кривыми, реализация довольно сложного метода оказалось очень простой.

Реализация метода немного отличается от псевдокода, изложенного ранее, но полностью сохраняет его идеи. Так как метод является вероятностным и не всегда дает искомый делитель в функции добавлено условие, чтобы она продолжала свою работу, пока не найдет один из делителей числа n.

Была выполнена реализация данного метода без использования встроенных функции PARI GP, но ее скорость факторизации больших чисел оставляла желать лучшего, неудачная реализация этого метода была исключена из данной работы.

Параллельная реализация заключается в вызове отдельным потоком функции нахождения одного из делителей числа n. С реализацией данного метода на PARI GP можно ознакомиться в Приложении 1.

.

3 ОБРАБОТКА ПОЛУЧЕННЫХ РЕЗУЛЬТАТОВ

Наиболее распространенной практикой в криптосистеме RSA является формирование открытого ключа с использованием двух сомножителей близкой разрядности. Поэтому для определения скорости факторизации вышеизложенных алгоритмов были использованы именно такие числа.
При вычислительных тестах также была последовательно увеличена разрядность факторизуемых чисел.

Для каждого открытого ключа разрядностью 2^k где $k = 10,20,30 \dots 80$, определим выборку чисел $n = p \cdot q$ размером в 50 чисел. Множители p и q являются простыми числами и не должны отличаться более чем на 1 порядок. Числа p и q брались случайным образом.

Для формирования данных был использован математический пакет PARI GP, с помощью которого была написана функция для решения поставленной задачи gen(n,k) где n – количество сгенерированных RSA ключей, а k – разрядность ключа n.

Полный исходный код функции представлен в приложении 1.

Для чистоты эксперимента для всех алгоритмов факторизации применены одинаковые данные. Для каждой полученной выборки было рассчитано среднее время разложения по формуле $t_{\text{ср}} = \frac{1}{m}\sum_{i=1}^{m} t_i$, где t_i – время разложения RSA ключей одной и той же разрядности, m – размер выборки.

Также были рассчитаны доверительные интервалы для среднего времени работы. Так как дисперсия для полученных опытных данных неизвестна, будем использовать ее несмещенную оценку, которая рассчитывается по формуле:

$$S = \sqrt{\frac{1}{m-1}\sum_{i=1}^{m}({t_i}^2 - t_{\text{ср}})}$$

Случайная величина t_i имеет распределение Стьюдента (m-1) степенью свободы. Тогда доверительный интервал для оценки $t_{\text{ср}}$, есть[13]

$$Q = [t_{\text{ср}} - \frac{S\ K_{St}}{\sqrt{m}}, t_{\text{ср}} + \frac{S\ K_{St}}{\sqrt{m}}]$$

где K_{St} – коэффициент Стьюдента для 90% вероятности.

Все полученные данные были рассчитаны в математической среде Octave, для вычисления коэффициента Стьюдента с заданной степенью свободы использовалась функция tinv(). Остальные данные были рассчитаны по вышеописанным формулам.

В математической среде Octave были построены графики для полученных результатов. В приложении 1 приведены функции для расчета среднего времени факторизации и доверительных интервалов.

Вычисления проводились на компьютере с 2-х ядерным процессором и следующими системными характеристиками «Intel(R) Atom(TM) CPU D525 1.80Ghz 32 разряда, Win7 SP1, объем ОЗУ – 4,00 Гб»

Во время проведения испытаний было обнаружено, что результаты вычислений параллельных методов дают замедление, оказалось, что стоимость коммуникаций между 2 параллельными процессами слишком велика и не дает выигрыша во времени разложения, а наоборот дает замедление вычислений в среднем на 5% что совпадает с информацией, изложенной в [9].

В дальнейшем для проверки гипотезы об ускорении факторизации параллельными методами было принято решение использовать компьютер, с 4-х ядерным процессором со следующими системными требованиями: Intel Core i7-2630QM CPU 2.00 ГГц, 64 разряда, Win7 SP1, объем ОЗУ – 4,00 Гб.

Далее на графиках изображенных на Рис 3.1-3.6 синим цветом будет представлена зависимость веремени работы непараллельной модифицкациии метода факторизации для 2-х ядерного процессора Intel Atom 1.80Ghz от разрядности раскладываемого числа, красным – время работы

непараллельной модифицкациии для 4-х ядерного процессора Intel Core i7-2630QM CPU, зеленным – время работы параллельной модификации для этого же процессора.

.

3.1 Метод Ферма

По результатам работы метода факторизации Ферма, были получены зависимости времени работы от разрядности раскладываемого числа, которые представлены на Рис 3.1.

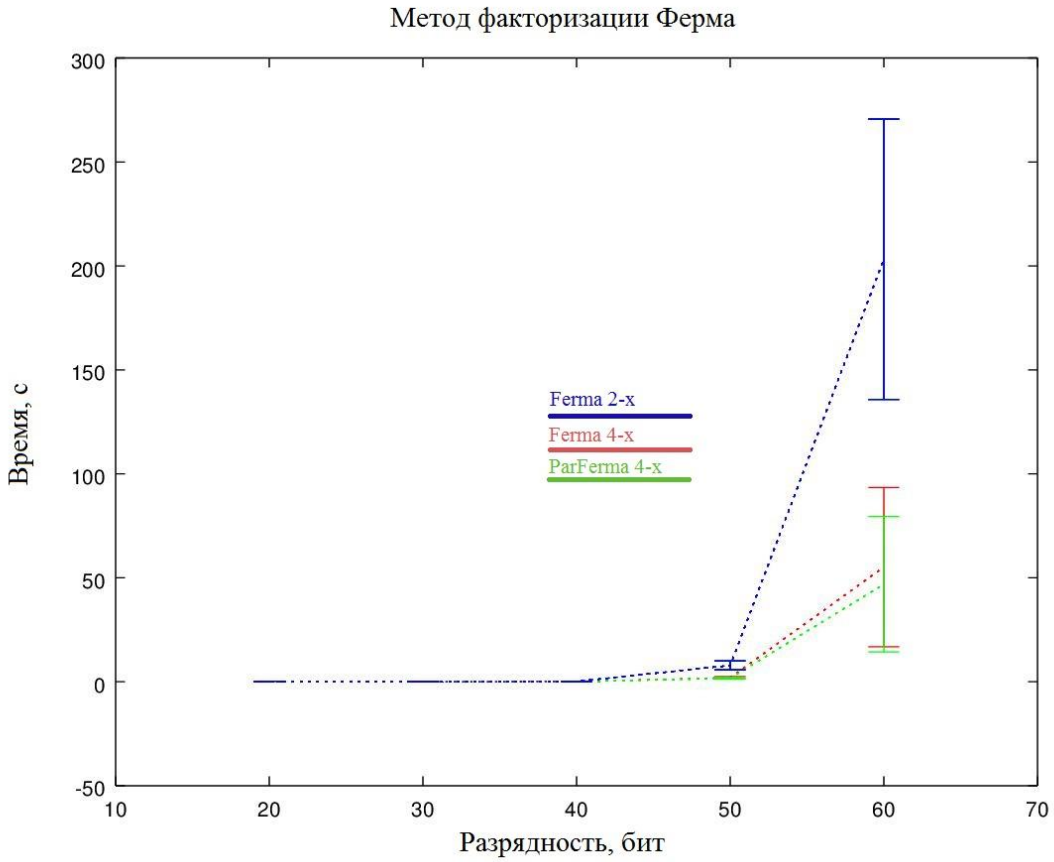

Рис. 3.1. Результаты измерений метода Ферма

Как видно из графика, изображенного на Рис 3.1, с помощью паралельной реализации метода удалось получить ускорение, равное примерно 9 процентам для 4-х ядерного процессора на числах, разрядность которых равна 60 бит. До этого момента разница во времени не столь существенна, это связано с тем, что на числах меньшой разрядности алгоритм работает

довольно быстро. К сожалнению, данный результат поподает в доверительные интервалы непаралельной версии метода, поэтому нельзя с полной уверенность утверждать, что было получено сколь-либо небольшое ускорение. Зависимость времени работы метода от разрядности факторизуемых чисел является экспоненциальной, что соответсвует теоритическим данным, известным для этого метода [5].

3.2 ρ - метод Полларда

В результате работы 2 ρ - метода Полларда, были получены зависимости времени работы от разрядности раскладываемого числа, которые представлены на Рис 3.2.

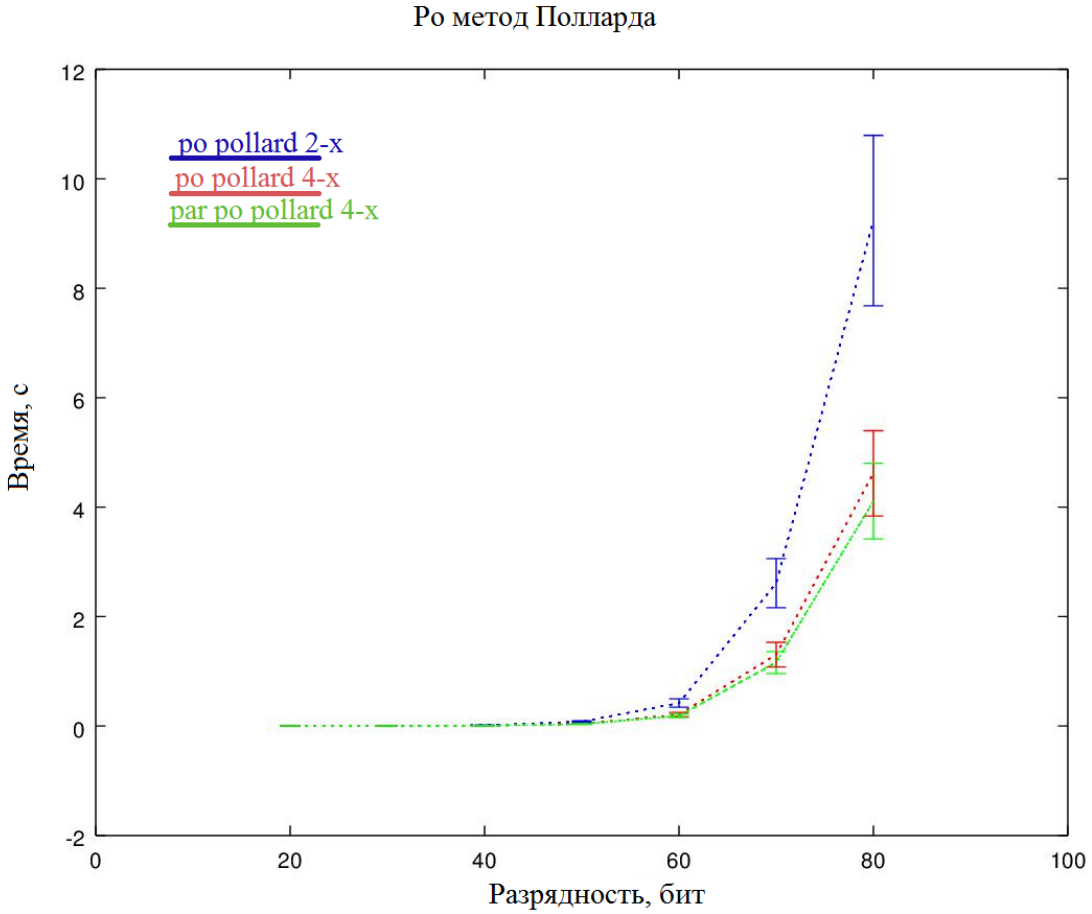

Рис. 3.2. Результаты измерений Ро метода Полларда.

Из полученных данных времени разложения на Рис. 3.2, можно увидеть что паралельная реализация ρ-метода и обычная выдает приблизительно одинаковый результат на числах одной разрядность которых менее 70 бит .

Также стоит отметить что реализация этого метада получилась быстрейшей из экспоненциальных алгоритмов факторизации. Паралельная реаизация быстрее обычной на 11 процентов для разрядности раскладываемого числа равной 80 бит, но среднее время попадает в доверительные интервалы обычной реализации метода, поэтому нельзя утверждать что паралеллизация привела к сколь-либо существенному ускорению.

3.3 p-1 метод Полларад

В результате работы p-1 метода Полларда, были получены зависимости времени работы от разрядности раскладываемого числа, которые представлены на Рис 3.3.

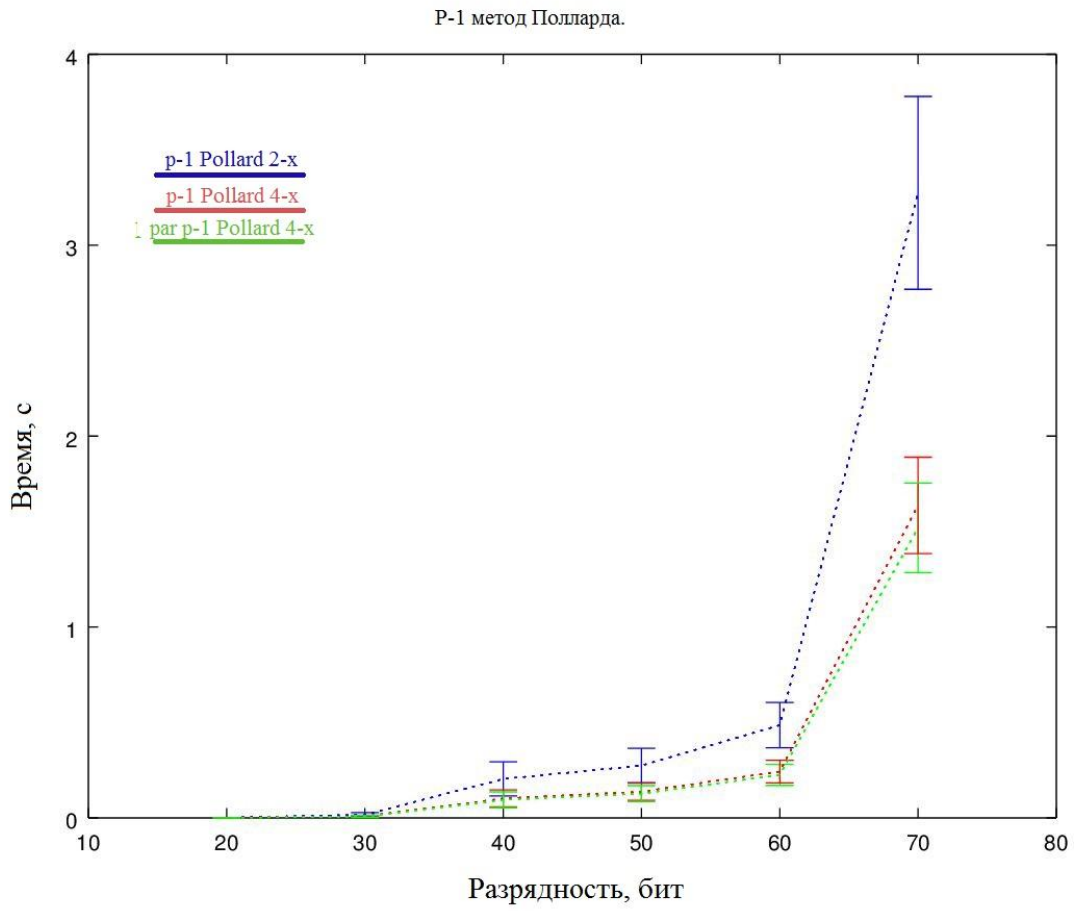

Рис. 3.3. Результаты измерений p-1 метода Полларда.

Из полученных данных времени разложения на Рис. 3.3, можно увидеть, что обычная реализация и паралельная реализация P-1 метода

выдает приблизительно одинаковый результат на числах разрядность которых не привышает 60 бит. Паралельная реализация быстрее обычной на 8,2 процента, для разрядности числа равной 70, но среднее время попадает в доверительные интервалы обычной реализации метода, поэтому нельзя утверждать что досигнуто значимое ускорение.

3.4 Метод Шермана Лемана

В результате работы метода Лемана, были получены зависимости времени работы от разрядности раскладываемого числа, которые представлены на Рис 3.4.

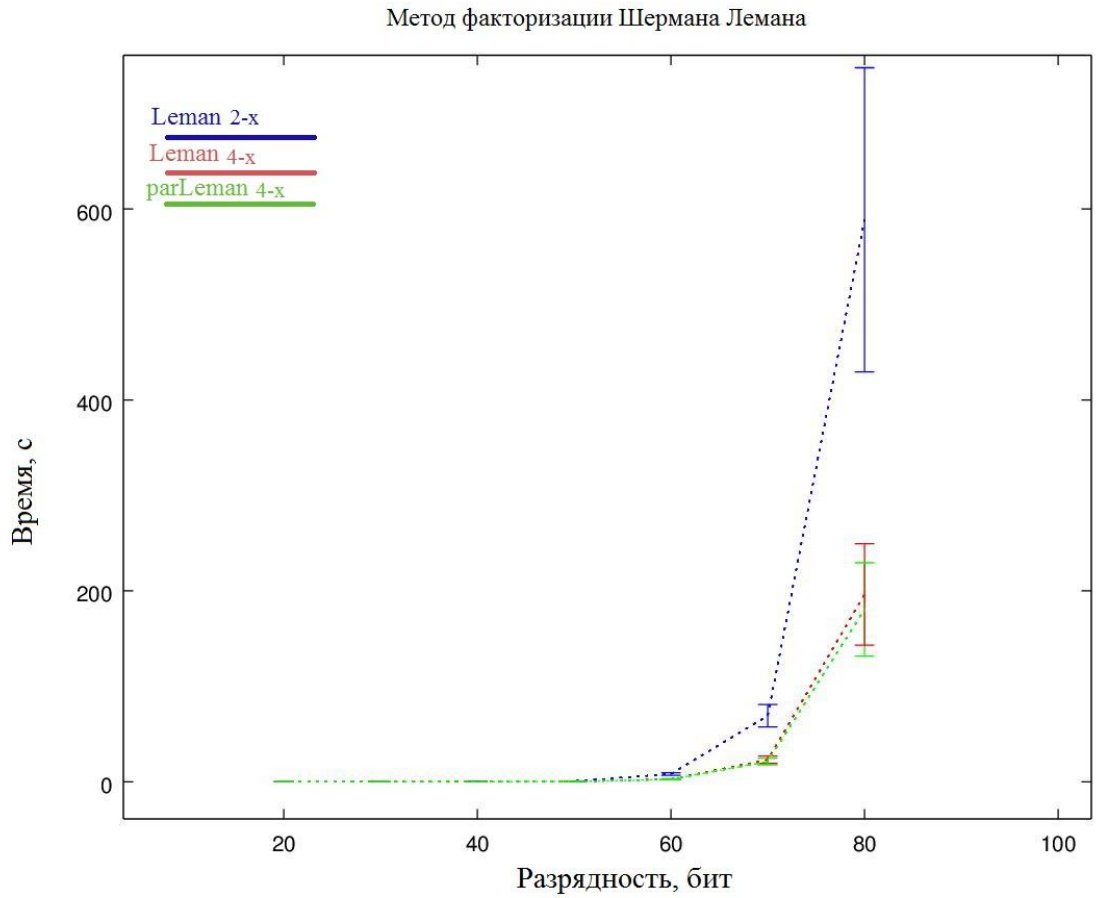

Рис. 3.4. Результаты измерений метода Шермана Лемана.

Реализация параллельного метода Лемана дала ускорение, равное 7,3% для разрядности числа равной 80 ,по сравнению с обычным алгоритмом, но и в этом случае среднее время вычислений параллельным методом попадает в

доверительные интервалы для времени работы последовательной версии алгоритма.

3.5 Метод Шенкса

В результате работы метода Шенкса, были получены зависимости времени работы от разрядности раскладываемого числа, которые представлены на Рис 3.5.

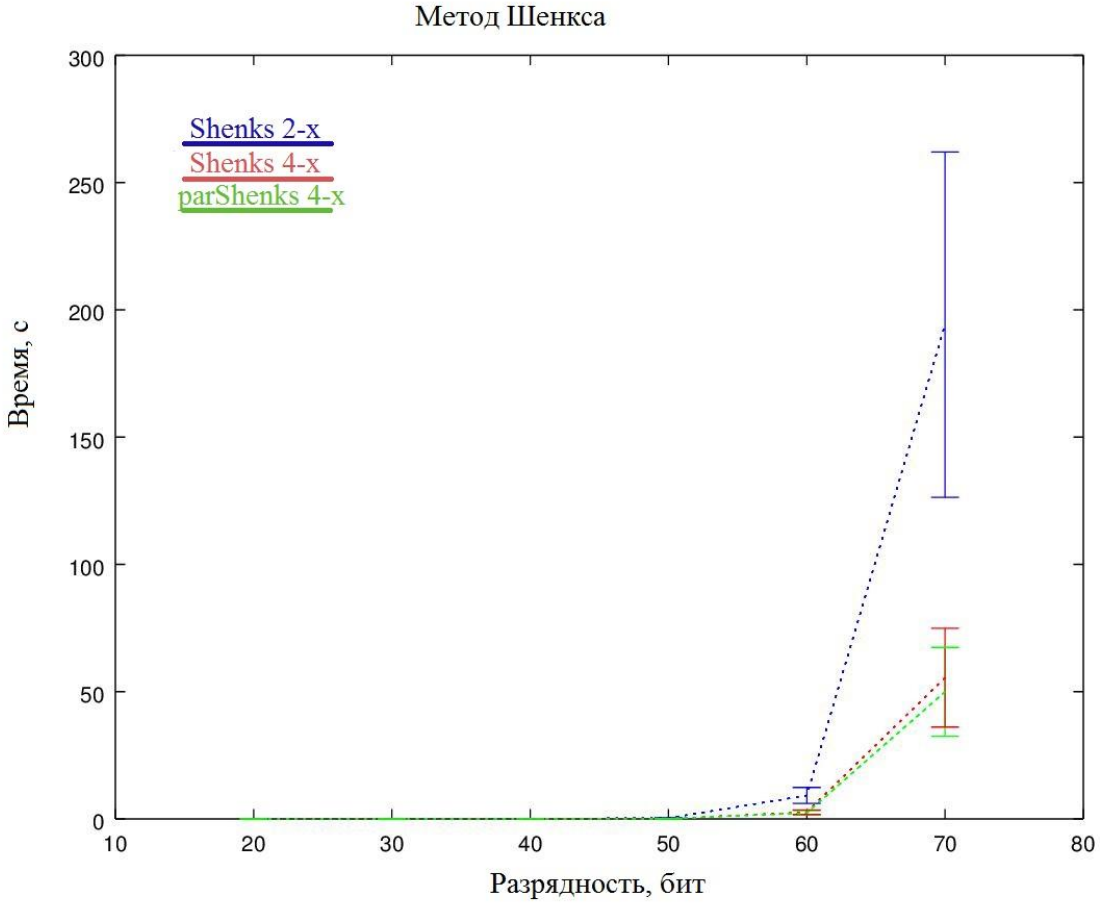

Рис. 3.5. Результаты измерений метода Шенкса.

Реализация параллельного метода Шенкса дала ускорение около 10 процентов для рарядности раскладываемого числа равной 70. Данный результат попадает в доверительный интервал обычной реализации, поэтому нельзя утверждать, что достигнуто ускорение.

3.6 Метод эллиптических кривых

В результате работы метода эллиптических кривых, были получены зависимости времени работы от разрядности раскладываемого числа, которые представлены на Рис 3.6.

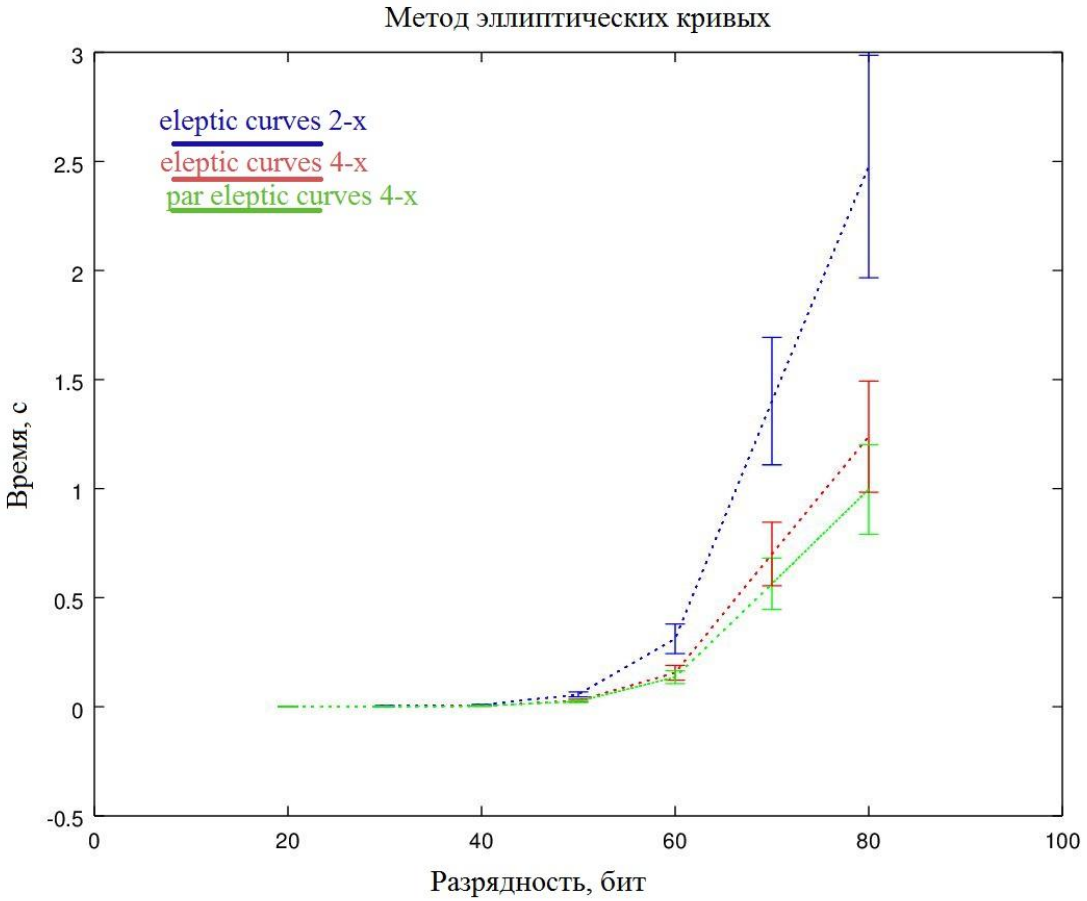

Рис. 3.6. Результаты измерений метода эллиптических кривых.

Для получения оптимальных результатов не достаточно было просто использовать вышеописанный алгоритм, правильный выбор границ B1 и B2 позволял получить самое быстрое время работы алгоритма. Для правильного выбора таких границ воспользуемся таблицей Бренда [8], в которой указаны рекомендуемые значения B1 и B2 для близких чисел определенной разрядности.

Как видно из графика метод является субэкспоненциальным. Скорость разложения целых чисел является наивысшей среди всех алгоритмов, представленных в этой работе, но, как и говорилось ранее в других алгоритмах, результаты полученной выборки слишком малы и их нельзя

считать достоверными, хотя и было получено ускорение параллельного метода порядка15% для чисел разрядностью равной 80 бит.

3.7 Сравнение алгоритмов факторизации

Так как скорость алгоритмов факторизации оказалась, очень различной сравнение скорости было представлено на двух отдельных графиках, на Рис 3.7 было представлено сравнение методов Ферма, Лемана и Шенкса, а на Рис. 3.8 сравнение

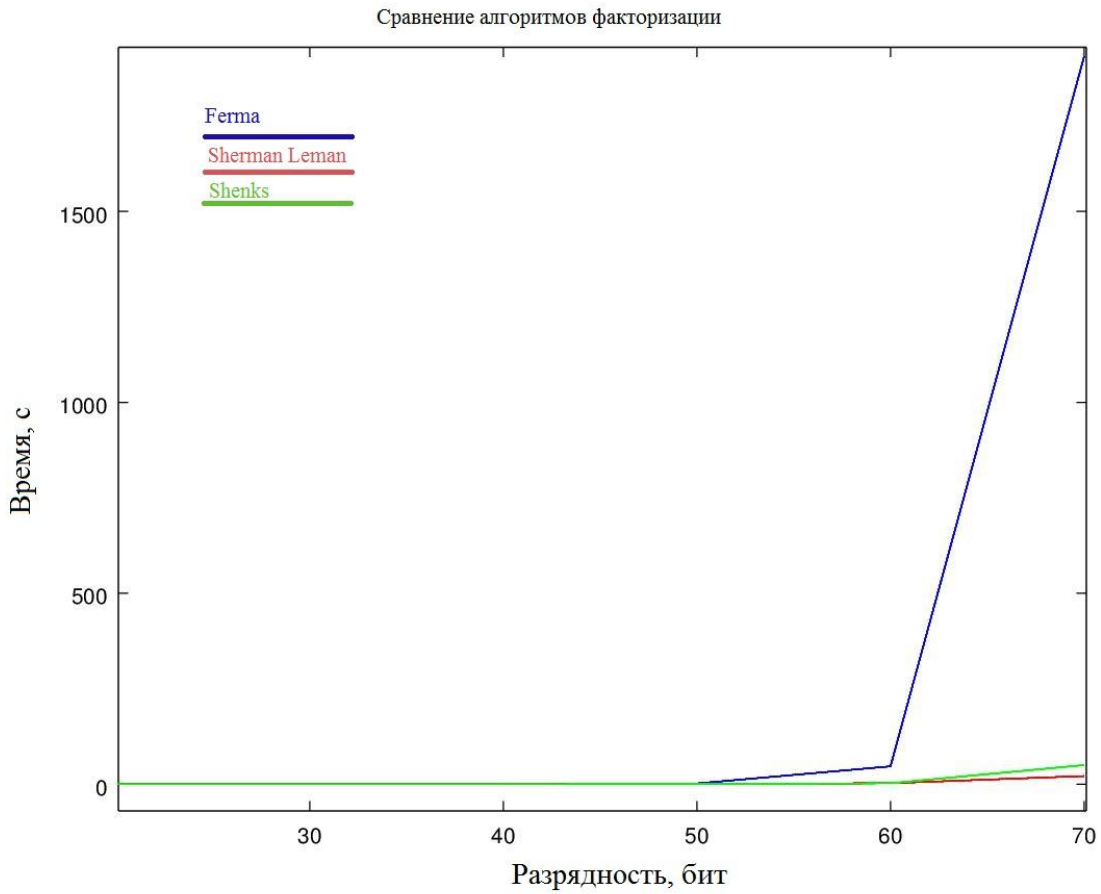

Рис.3.7. Сравнение скорости работы реализованых алгоритмов.

Из этого графика можно сделать вывод, что метод Ферма имеет наименьшую скорость разложения среди всех алгоритмов факторизации, за ним следуют метод Шенкса и метод Шермана лемана. Метод Ферма не рекомендуется использовать на пракике даже в составных програмных модулях.

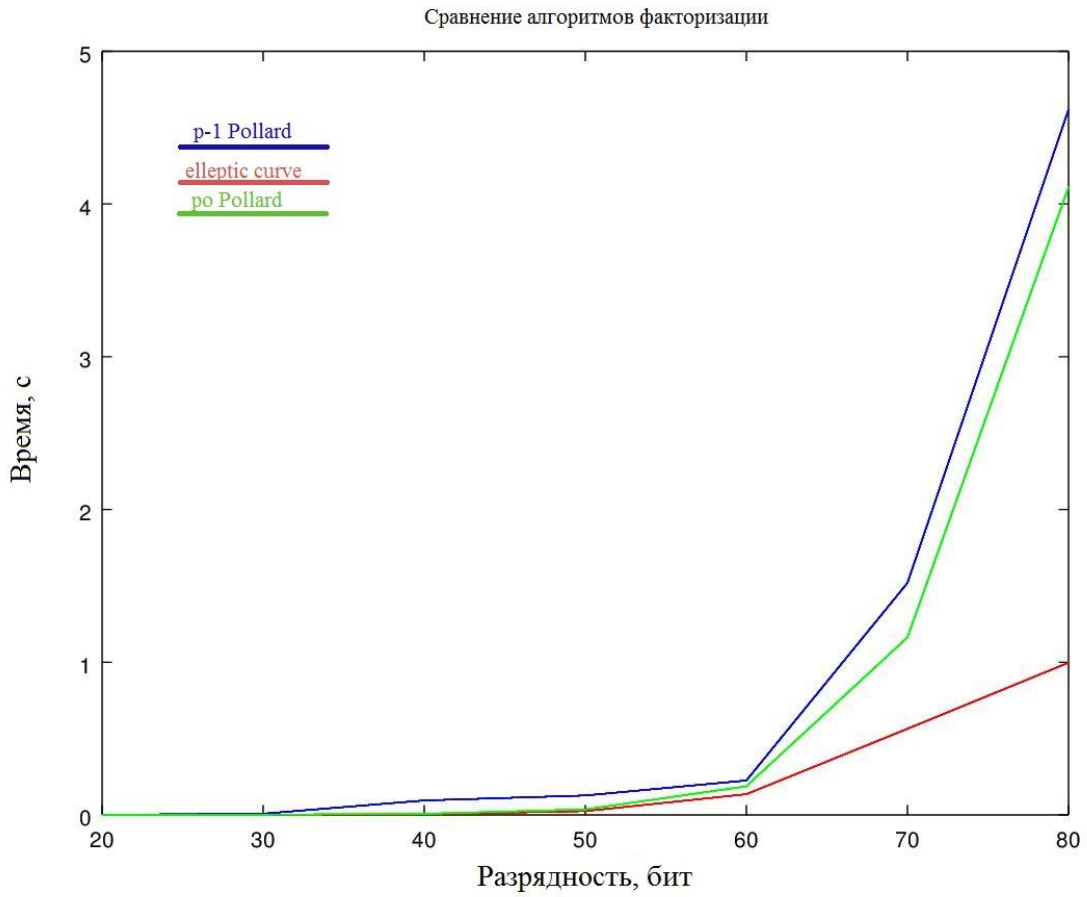

Рис. 3.8. Сравнение скорости работы реализованых алгоритмов.

Как данного графика алгоритмы р и Р-1 метод Полларда показывают приблизительно равное время работы, Р-1 метод , лишь слегка уступает Ро методу , который является наибыстрейшим алгоритмом факторизации среди простых, он уступает только методу эллиптических кривых, который в свою очередь является субэкспоненциальным. Все 3 метода показывают приблизительно одинаковую скорость на числах, разрядность которых не превышает 60. В дальнейшем наблюдается значительный выигрыш алгоритма факторизации методом эллиптических кривых

4 ДОСТУПНОСТЬ ОБЛАЧНЫХ ВЫЧИСЛЕНИЙ

В настоящее время злоумышленник может легко получить очень большой объем вычислительных мощностей. Если бы удалось получить линейное ускорения для одного из алгоритмов, то можно было воспользоваться облачными вычислениями, которые предоставляет, например, сервис Amazon [11].

Согласно исследованиям, связанным с параллельной реализацией методов факторизации в системах с распределенной памятью [10], для алгоритма факторизации методом эллиптических кривых было получено, практически линейное ускорение. Что можно наблюдать на приведенном ниже графике Рис. 4.1, построенном по данным работы [10].

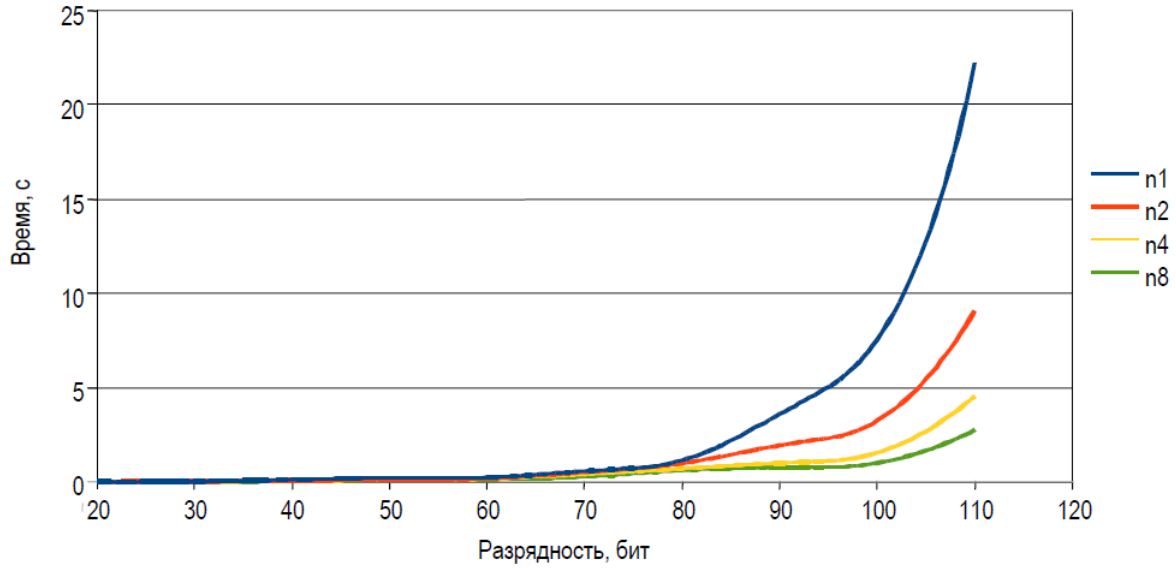

Рис. 4.1. Результаты исследований для метода эллиптических кривых

На графике, который представлен на Рис.4.1, происходит сравнение скорости разложения методом эллиптических кривых от количества используемых процессоров. Результаты измерений были получены с помощью распределенной сети, состоящей из 8 компьютеров, системная конфигурация которых имеет следующие характеристики: CPU – Dual Core E5200 2,5 ГГц, ОЗУ – 3,24 Гб; ОС – Mandriva Linux 2008. Локальная вычислительная сеть – 100 Мбит/с.

Применим грубую оценку результатов данных исследований для того, чтобы выяснить, сколько потребуется ресурсов потребуется злоумышленнику для взлома за приемлемое время RSA ключей, которые на данный момент всё ещё являются актуальными.

Как видно из графика на Рис 4.1, скорость факторизации от увеличения количества потоков в 2 раза возрастает практически в 2 раза. Предположим, что линейное ускорение сохранится и при дальнейшем увеличении количества запущенных процессов. Тогда экстраполируем эти данные на вычислительные мощности, которые предоставляет Amazon.

По данным с сайта Amazon [11] выберем вычислительную машину, близкую по характеристикам к компьютерам, которые использовались в приведенном исследовании. По этим данным была выбрана конфигурация: M1 General Purpose Large ЦП: 2 cores 4 units ОЗУ: 7,5 Гб ОС: Linux. Здесь 1 unit эквивалентен 1.2 ГГц.

Цена использования данной вычислительной машины в час составляет \$0.175. Для оценки времени разложения воспользуемся формулой $\exp\left(\frac{\sqrt{2}}{2} + o(1)\sqrt{\ln(p)\,lnln(p)}\right)$ [5], где p – наименьший множитель раскладываемого числа, пусть будем считать p приблизительно равным 2^{512}, тогда приблизительное время, требуемое для разложения одного RSA ключа, размерность которого составляет 1024 бита, займет примерно

4300401739674 лет для одного процессора. Для разложения данного числа за один год нам потребуется 2^{42} отдельных вычислительных машин, что данный облачный сервис нам не способен предоставить и даже если бы мог, то такие вычисления потребовали бы затрат в 6742205301522432 долларов США.

То есть даже теоретически при соблюдении всех благоприятных условий обычный пользователь не сможет взломать данную криптосистему, так как это слишком дорого и неприемлемо долго.

ЗАКЛЮЧЕНИЕ

Результаты работы не позволяют свидетельствовать о существенном ускорении работы методов факторизации с использованием их параллельных модификаций. Возможно, это связано с малым объемом полученной выборки значений времени факторизации. Увеличить данный объем не представлялось возможным на использованных вычислительных машинах из-за долгой скорости факторизации отдельных методов.

Самыми простыми в параллельной реализации оказались ρ-метод Полларда и метод эллиптических кривых. Дальнейшие исследования в этой области могут заключаться: в реализации более сложных алгоритмов факторизации, таких как квадратичное решето и решето числового поля и их модификаций; поиску новых стратегий для реализации параллельных вычислений, а также в использовании кластерных, облачных или распределенных вычислений или исследовании того, дают ли выше описанные стратегии параллельных вычислений линейное ускорение для процессоров с большим количеством физических ядер.

Также стоит отметить, что безопасность открытого RSA ключа не вызывает сомнений без существенного прорыва в методах факторизации числа его взлом за приемлемое время не осуществим. И даже при определенном прорыве для достижения конфиденциальности данных будет достаточно увеличить размер RSA ключа.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. The Alternative History of Public-Key Cryptography [Электронный ресурс] URL: http://cryptome.org/ukpk-alt.htm (дата обращения 12.02.2015) Формат доступа: свободный

2. RSA Challenge [Электронный ресурс] URL: http://www.emc.com/emc-plus/rsa-labs/historical/the-rsa-challenge-numbers.htm (дата обращения 13.03.2015) Формат доступа: свободный

3. Factorization of a 768-bit RSA modulus http://eprint.iacr.org/2010/006.pdf (дата обращения 21.03.2015) Формат доступа: свободный

4. Маховенко, Е. Б. Теоретико-числовые методы в криптографии: Учебное пособие / Е. Б. Маховенко. – М.: Гелиос АРВ, 2006. – 320 с., ил.

5. Методы факторизации натуральных чисел: учебное пособие / Ш.Т. Ишмухаметов.– Казань: Казан. ун. 2011.– 190 с.

6 ρ-метод Полларда Лектор: Н.Ю. Золотых Записал: А. Фель 22 ноября 2008 [Электронный ресурс] URL: http://www.uic.unn.ru/~zny/compalg/Lectures/ca_11_RhoPollard.pdf (дата обращения 12.04.2015) Формат доступа: свободный

7. LehmanR.S.Factoring large integers//Math.Comp.1974.V.28.

8. Brent, Richard P. (1999). "Factorization of the tenth Fermat number". Mathematics of Computation 68 (225): 429–451. [Электронный ресурс] URL: http://www.ams.org/journals/mcom/1999-68-225/S0025-5718-99-00992-8/S0025-5718-99-00992-8.pdf (дата обращения 21.05.2015) Формат доступа: свободный

9. User's Guide to PARI / GP (version 2.3.3) C. Batut, K. Belabas, D. Bernardi, H. Cohen, M. Olivier [Электронный ресурс] URL: http://pari.math.u-bordeaux.fr/pub/pari/manuals/2.3.3/users.pdf (дата обращения 12.05.2015) Формат доступа: свободный

10. Параллельная реализация и сравнительный анализ алгоритмов факторизации с распределенной памятью / Макаренко А.В. Пыхтеев А.В. Ефимов С.С. [Электронный ресурс] URL:

http://cyberleninka.ru/article/n/parallelnaya-realizatsiya-i-sravnitelnyy-analiz-algoritmov-faktorizatsii-v-sistemah-s-raspredelyonnoy-pamyatyu (дата обращения 06.04.2015)
11. Easy Amazon EC2 Instance Comparison [Электронный ресурс] URL: http://www.ec2instances.info/ (дата обращения 15.05.2015) Формат доступа: свободный
12. Математический пакет PARI GP [Электронный ресурс]. http://pari.math.u-bordeaux.fr/ (дата обращения 01.02.2015). Формат доступа: свободный.
13. Солопченко, Г. Н. Теория вероятностей и математическая статистики: Учебное пособие / Г. Н. Солопченко, – СПБ.: Издание политехнического университета, 2010. – 212 с., ил.

ПРИЛОЖЕНИЕ 1

Листинги реализованных методов и функций

Реализация метода факторизации Ферма.

```
ferma(n)=
{
my(x,y,t,s);
x=round(sqrt(n));
y=x*x-n;
while(issquare(y)==0,
x=x+1;
y=x*x-n;
);
s=round(x-sqrt(y));
t=round(x+sqrt(y));
if(isprime(s),print("is prime factor "s),ferma(s));
if(isprime(t),print("is prime factor "t),ferma(t));
}
```

Параллельная реализация метода Ферма.

```
fermathread(i,n)= /*функция основных вычисления для каждого отдельного потока*/
{
my(x,y,ans);
x=i;
y=x*x-n;
while(issquare(y)==0,x=x+1;
y=x*x-n);
ans=vector(2);
ans[1]=x;
ans[2]=y;
return(ans);
}


dellimost(a,n)= /*метод проверики делимости числа*/
{
return(n%a);
}

ansParferma(n)= /*функция нахождения 2 сомножителей числа n*/
{
my(B,d);
B=round(sqrt(exp(1)^((sqrt(2*log(n)*log(log(n)))))); /*расчет границы B*/
parforprime(a=2,B,dellimost(a,n),del,if(del==0,return(a))); /*цикл проверки делимости*/
d=round(sqrt(n));
parfor(a=d,d+d/2,
fermathread(a,n),ans,
if(issquare(ans[2])==1,return(ans));
);
}

parferma(n)= /* функция нахождения всех простых делителей числа n*/
{
```

```
my(answer,s,t);
answer=ansParferma(n);
s=round(answer[1]-sqrt(answer[2]));
t=round(answer[1]+sqrt(answer[2]));
if(isprime(s),print("is prime factor "s),parferma(s));
if(isprime(t),print("is prime factor "t),parferma(t));
}
```

Реализация Po метода Полларда.

```
poPollard(n)= /*функция нахождения одного из множителей числа n */
{
my(x,y,i,stage,d);
until(x,x=random(n-2));
y=1;
i=0;
stage=2;
while(gcd(n,round(x-y))==1,
if(i==stage,y=x;stage=stage*2);
x=(x^2+1)%n;
i=i+1;
);
d=gcd(n,round(x-y));
return(d);
}

Pollard(n)= /*Функция нахождения всех простых сомножителей числа n */
{
my(a,fact);
a=poPollard(n);
if(isprime(a),print("is prime factor " a),Pollard(a));
fact=n/a;
if(isprime(fact),print("is prime factor " fact),Pollard(fact));
}
```

Паралельная реализация Po метода Полларда.

```
parpoPollard(n)=  /*параельная функция нахождения одного множителя */
{
parfor(x=1,n-2,parPoIter(x,n),ans,if(ans>1 && ans<n,return(ans)));
}

parPoIter(x,n)= /*функция основных вычислений для каждого отдельного потока*/
{
my(y,i,stage,d);
y=1;
i=0;
stage=2;
while(gcd(n,round(x-y))==1,
if(i==stage,y=x;stage=stage*2);
x=(x^2+1)%n;
i=i+1;
);
d=gcd(n,round(x-y));
return(d);
}
```

```
parPollard(n)=/*функция нахождения всех множителей числа n*/
{
my(a,fact);
a=parpoPollard(n);
if(isprime(a),print("is prime factor " a),parPollard(a));
fact=n/a;
if(isprime(fact),print("is prime factor " fact),parPollard(fact));
}
```

Реализация p-1 метода Полларда

```
P_1Pollard(n)=/*функция нахождения всех множителей числа n */
{
my(a,fact,B);
B=round(sqrtn(n,6));
until(a!=FAIL,a=pollardP_1(n,B););
if(isprime(a),print("is prime factor " a),P_1Pollard(a));
fact=n/a;
if(isprime(fact),print("is prime factor " fact),P_1Pollard(fact));
}

pollardP_1(n,B)= /*функция нахождения одного множителя*/
{
        my(m,a,p,j,d,g,lgn);
   until(a,a=random(n));
   g=gcd(a,n);
   if(g>1,return(g),
   p=3;j=1;d=1;lgn=log(n);
   while(d==1&&j<=B,
        a=lift(Mod(a,n)^(p^floor(lgn/log(p))));
        d=gcd(a-1,n);
        if(d>1&&d<n,return(d));
        j=j++;
        p=nextprime(p++)
        );
        m=1;
   while(d==1&&m<=floor(lgn/log(2)),
                        a=lift(Mod(a,n)^2);
                        d=gcd(a-1,n);
                        if(d>1&&d<n,return(d));
                        m=m++
                        );
      if(d==1||d==n,return(FAIL)))
}
```

Параллельная реализация p-1 метода Полларда.

```
parP_1pollard(n)= /*функция нахождения всех простых множителей n*/
{
my(a,fact);
a=pariterP_1Pollard(n);
if(isprime(a),/*print("is prime factor " a)*/,P_1Pollard(a));
```

```
fact=n/a;
if(isprime(fact),/*print("is prime factor " fact)*/,P_1Pollard(fact));
}

pariterP_1Pollard(n)=  /*параллельная функция нахождения одного множителя числа n*/
{
B=round(sqrtn(n,6));
parfor(i=1,n,pollardP_1(n,B),ans,if(1<ans && ans<n,return(ans)))
}
```

Реализация метода Шермана Лемана.

```
sherman_Leman(n)= /*функция нахождения одного из множителей числа n*/
{
my(a,A,B,gcdSum,gcdRaz);
for(a=2,round(sqrtn(n,3)),
if(n%a==0,
return(a);)
);
for(k=1,round(sqrtn(n,3)),
for(d=0,round(sqrtn(n,6)/(4*sqrt(k))),
if(issquare(sqr(round(sqrt(4*k*n))+d)-4*k*n),
A=round(sqrt(4*k*n))+d;
B=round(sqrt(sqr(A)-4*k*n));
gcdSum=gcd(A+B,n);
gcdRaz=gcd(A-B,n);
if(1<gcdSum && gcdSum<n,
return(gcdSum));
if(1<gcdRaz && gcdRaz<n, return(gcdRaz)););););
}

Leman(n)=/*функция нахождения всех множителей n*/
{
my(k,newn);
k=sherman_Leman(n);
if(isprime(k),/*print("is prime factor "k)*/,Leman(k));
newn=n/k;
if(isprime(newn),/*print("is prime factor "newn)*/,Leman(newn));
```

Паралельная реализация метода Шермана Лемана.

```
shermamiter(k,n)=/*функция основных вычислений*/
{
my(gcdSum,gcdRaz);
for(d=0,round(sqrtn(n,6)/(4*sqrt(k))),
if(issquare(sqr(round(sqrt(4*k*n))+d)-4*k*n),
A=round(sqrt(4*k*n))+d;
B=round(sqrt(sqr(A)-4*k*n));
gcdSum=gcd(A+B,n);
gcdRaz=gcd(A-B,n);
```

```
if(1<gcdSum && gcdSum<n,
return(gcdSum));
if(1<gcdRaz && gcdRaz<n,
return(gcdRaz));
);
);
}

parSherman_Leman(n)= /*паралельная функция нахождения одного множителя*/
{
my(sqrtn3);
sqrtn3=round(sqrtn(n,3));
parfor(a=2,sqrtn3,l=n&a,l,if(l==0,return(a)));
parfor(k=1,sqrtn3,shermamiter(k,n),ans,
if(1<ans && ans<n,return(ans)))
}

parLeman(n)= /*Функция нахождения всех множителей*/
{
my(aka,newn);
aka=parSherman_Leman(n);
if(isprime(aka),/*print("is prime factor "aka)*/,parLeman(aka));
newn=n/aka;
if(isprime(newn),/*print("is prime factor "newn)*/,parLeman(newn));
}
```

Реализация метода Шенкса.

```
Shenks(n,b)= /*функция нахождения одного из множителей n*/
{
my(P0,Q0,Q1,P1,r0,r1,g,counter);
if(issquare(n),return(round(sqrt(n))));
P0=0; Q0=1; r0=floor(sqrt(b*n)); P1=r0;
Q1=b*n-r0^2; r1=floor(2*r0/Q1);
until(issquare(Q1),
P2=r1*Q1-P1;
Q2=Q0+(P1-P2)*r1;
r2=floor((P2+floor(sqrt(b*n)))/Q2);
r0=r1; r1=r2; P0=P1; P1=P2; Q0=Q1; Q1=Q2;
);
P0=-P1; Q0=floor(sqrt(Q1));
r0=floor((P0+floor(sqrt(b*n)))/Q0);
P1=r0*Q0-P0; Q1=(b*n-P1^2)/Q0;
r1=floor((P1+floor(sqrt(b*n)))/Q1);
counter=1;
until(P0==P1,
P2=r1*Q1-P1;
Q2=Q0+(P1-P2)*r1;
r2=floor((P2+floor(sqrt(b*n)))/Q2);
r0=r1; r1=r2; P0=P1; P1=P2; Q0=Q1; Q1=Q2;
counter++;
if(counter>1000,return(FAIL)); /*ограничение в 1000 итераций*/
);
g=gcd(Q0,n);
```

```
if(1<g && g<n,return(g),return(FAIL));
}

shenks(n)= /*функция нахождения всех множителей */
{
i=1;
until(k!=FAIL,
k=Shenks(n,i);i++;);
if(isprime(k),print("is prime factor "k),shenks(k));
newn=n/k;
if(isprime(newn),print("is prime factor "newn),shenks(newn));
}
```

Паралельная реализация метода Шекса.

```
parshenks(n)= /*функция нахождения всех множителей*/
{
my(newn,ans);
ans=ParShenks(n);
if(isprime(ans),/*print("is prime factor "ans)*/,parshenks(ans));
newn=n/ans;
if(isprime(newn),/*print("is prime factor "newn)*/,parshenks(newn));
}

ParShenks(n)= /*параллельный вызов функции вычисления одного множителя*/
{
parfor(i=2,n,Shenks(n,i),k,if(k!=FAIL,return(k)));
}
```

Реализация метода эллиптических кривых.

```
ecm(n)= /*функция нахождения всех делителей числа*/
{
my(ans,newn);
until(ans,ans=ECM(n));
if(isprime(ans),/*print("is prime factor "ans)*/,ecm(ans));
newm=n/ans;
if(isprime(newm),/*print("is prime factor "newm)*/,ecm(newm));
}

ECM(N, B = 1000!, nb = 100)= /*функция нахождения одного делителя числа N*/

    {
     for(a = 1, nb,
       iferr(ellmul(ellinit([a,1], N), [0,1], B),
         E, return(gcd(lift(component(E,2)),N)),
         errname(E) == "e_INV"));
    }
```

Параллельная реализация метода эллиптических кривых.

```
parecm(n)=/*функция нахождения всех множителей числа n*/
```

```
{
my(ans,newn);
until(ans,ans=parECM(n));
if(isprime(ans),/*print("is prime factor "ans)*/,ecm(ans));
newm=n/ans;
if(isprime(newm),/*print("is prime factor "newm)*/,ecm(newm));
}

parECM(N, B = 1000!, nb = 100)=/*функция нахождения одного делителя числа*/
{
parfor(a = 1, nb,ecmiter(N,a,B),ans,if(1<ans && ans<N,return(ans)));
}
```

Функция для формирования исходных данных

```
genN(n,k)=
{
for(i=1,n,
until(p,p=precprime(random(2^(k\2))));
until(q,q=precprime(random(2^(k\2))));
openkey=p*q;
print("q="q ,"p=" p ,"openkey=" openkey))
}
```

Функция расчета среднего значения в Octave

```
function tcp=t(X)
ti=max(size(X));
tcp=sum(X)/si;
end;
```

Функция расчета доверительных интервалов в Octave

```
function Ic=Q(X)
buff=X;
ti=max(size(X));
tcp=sum(X)/si;
for i = 1:1:si
buff(i) = (X(i)- tcp)*(X(i)- tcp);
end;
S = sqrt(sum(buff)/(si-1));
Ic = [ tcp - S/sqrt(si)*tinv(0.90,si-1); tcp + S/sqrt(si)*tinv(0.90,si-1)];
end;
```