УДК 004.056.55

РЕАЛИЗАЦИЯ Р-МЕТОДА ФАКТОРИЗАЦИИ ПОЛЛАРДА НА ЯЗЫКЕ С++

Смирнов И. А., Черкесова Л. В. , Сафарьян О. А.
Донской государственный технический университет, г. Ростов-на-Дону, Российская Федерация
terran.doatk@mail.ru
chia2002@inbox.ru
safari_2006@mail.ru

Представлен проект реализации ρ-метода факторизации Полларда на языке С++, который работает быстрее стандартного алгоритма на 27%. Это помогает значительно облегчить работу в расшифровывании и криптоанализе в различных шифрах, например, таких как RSA.

Ключевые слова: факторизация, Эвклид, Поллард, алгоритмическая сложность, матрица смежности, матрица достижимости, метрика Маккейба.

UDC 004.056.55

THE REALIZATION OF THE POLLARD FACTORIZATION P – METHOD IN THE PROGRAMMING LANGUAGE C ++

Smirnov I. A., Cherkesova L. V., Safaryan O. A.
Don State Technical University, Rostov-on-Don, Russian Federation
terran.doatk@mail.ru
chia2002@inbox.ru
safari_2006@mail.ru

The paper presents the project implementation of ρ-factor Pollard factorization in C ++, which works faster than the standard algorithm by 27%, which can significantly facilitate the work in deciphering and cryptanalysis of various ciphers such as RSA

Keywords: factorization, Euclid, Pollard, algorithmic complexity, adjacency matrix, reachability matrix, McCabe metric

Введение. Исследования криптоалгоритмов, на сегодняшний день, очень актуально в связи с угрозой кибернетических атак и необходимостью защиты информации на предприятиях различного уровня, в том числе стратегического назначения. Алгоритм шифрования с открытым ключом RSA по сей день является одним из самых широко используемых криптографических алгоритмов. Данный ассиметричный криптографический алгоритм основан на вычислении факторизации больших чисел. В схеме RSA используются открытый и закрытый ключ. Открытый ключ состоит из открытой экспоненты — некоторое число e — и модуля N, который получается произведением простых целых чисел P и Q. После формирования открытых ключей формируется закрытый ключ. Для этого необходимо вычислить функцию Эйлера $\phi_n = (P-1)\times(Q-1)$ и элемент d, который рассчитывается по формуле: $d = e^{-1} \bmod \phi^n$. С помощью элемента d можно расшифровать зашифрованную информацию, т.е. факторизовать модуль N [1].

Для решения задачи факторизации используется декомпозиция — научный метод, разбивающий большую задачу на серии меньших взаимосвязанных задач. В данном случае факторизация — это декомпозиция объекта числа N в произведение двух чисел, которые при перемножении дадут исходный объект. К примеру, число 15 можно факторизовать на простые числа 3 и 5, а полином $x^2 - 81$, соответственно, на (x −9) (x + 9). Благодаря факторизации получаем произведение более простых объектов.

Однако, в криптоалгоритмах используются очень большие числа, что составляет определенные затруднения для факторизации. Для решения таких задач было создано много различных алгоритмов, таких, как [2]:

- алгоритм факторизации Шенкса,
- алгоритм факторизации с помощью эллиптических кривых,

– тест простоты Миллера – Рабина,
– ρ-алгоритм Полларда и др.

Все они, несомненно, имеют свои достоинства и недостатки. В частности, одним из главных недостатков, можно считать довольно низкое быстродействие, что при современных киберугрозах весьма существенно.

Рассмотрим алгоритм Полларда.

Постановка задачи.

Задачей исследования является улучшение алгоритма факторизации ρ-метода Полларда, модификация которого способна увеличить быстродействие реализованного ранее стандартного алгоритма. Улучшенная версия алгоритма прошла проверку тремя критериями надежности программного обеспечения, в том числе и по метрике Маккейба, и показала лучшие результаты.

Основная часть

Для факторизации целых чисел Джон Поллард в 1975 году изобрел собственный алгоритм, который получил название ρ-алгоритм Джона Полларда. Фундаментом данного алгоритма является алгоритм Роберта Флойда, изобретенный им в конце 60-х годов 20 века и который эффективен для поиска длины цикла в последовательности. Такие математики, как Джон Поллард, Дональд Кнут и другие, провели подробный анализ этого алгоритма и предложили несколько модификаций и улучшений алгоритма [3]. Наиболее работоспособным при факторизации составных чисел с малыми множителями в разложении оказался ρ-алгоритм. Особенностью данного алгоритма является построение числовой последовательности, в которой с некоторого номера n элементы этой последовательности образуют цикл (рис.1).

Эта особенность представлена в виде греческой буквы ρ, что как раз и послужила основанием для названия всего семейства методов Полларда.

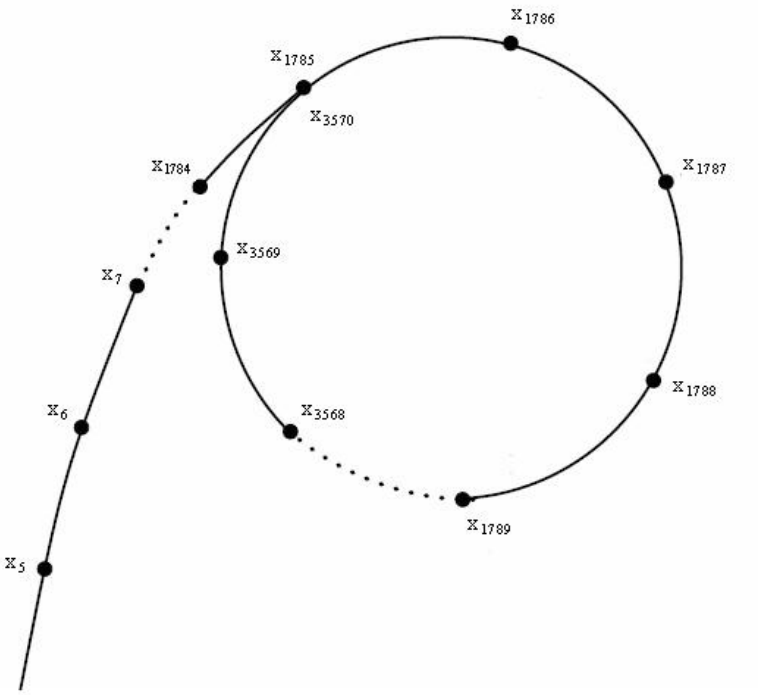


Рис. 1. Числовая последовательность зацикливается начиная с некоторого n

В 1981 году Ричард Брент и Джон Поллард с помощью этого алгоритма нашли наименьшие делители чисел Ферма $F_n = 2^{2^n} + 1$ при $5 \le n \le 13$.

Так, $F_8 = 1238926361552897 \cdot p_{62}$, где p_{62} — простое число, состоящее из 62 десятичных цифр. В рамках проекта "Cunningham project", который был запущен в 1925 году, алгоритм Полларда помог найти делитель длиной 19 цифр числа $2^{2386}+1$.

Делители еще больших чисел также можно было бы найти, но создание и использование эллиптических кривых для факторизации сделало алгоритм Полларда менее конкурентоспособным и востребованным.

Алгоритм ρ-метод Полларда

1. Рассмотрим последовательность целых чисел x_n, такую, где каждое следующее число $x_{i+1} = \left(x_n^2 - 1\right) \bmod N$, а $n = 0,1,2,...$ В этой последовательности x_0 — небольшое число.
2. На каждом шаге будем вычислять значение $d = НОД\left(n, \left|x_i - x_j\right|\right)$,где j < i.
3. Если $d \neq 1$, то вычисления заканчиваются. Найденное число d является делителем числа n. Если n/d не является простым числом, то процедуру можно продолжить, взяв вместо n число n/d [1].

Взамен функции $F(x) = (x^2 - 1) \bmod n$ для вычисления x_{n+1} можно взять иной многочлен, к примеру, $x^2 + 1$ или иной другой многочлен 2-й степени $F(x) = ax^2 + bx + c$ [1].

Главным недостатком данного метода является выделение дополнительной памяти для хранения предыдущих значений x_j. Заметим, что если $\left(x_i - x_j\right) \equiv 0 \left(\bmod p\right)$, то $\left(f(x_j) - f(x_i)\right) \equiv 0 \left(\bmod p\right)$, поэтому, если пара $\left(x_i, x_j\right)$ дает нам решение, то решение даст любая пара $\left(x_i + k, x_j + k\right)$ [1].

В связи с этим нет потребности проверять все пары $\left(x_i, x_j\right)$, а есть возможность всего лишь ограничиться парами вида $\left(x_i, x_j\right)$, где $j = 2^k$, и k проходит набор последовательных значений 1, 2, 3,... , а i будет принимать значения из промежутка $\left[2^k + 1; 2^{k+1}\right]$.

Еще одна модернизация ρ-метода Полларда была создана Флойдом, согласно которой значение y изменяется на каждом шаге по формуле $y = F_2(y) = F(F(y))$. В связи с этим на шаге i будут найдены значения $x_i = F^i(x_0)$, $y_i = x_{2i} = F^{2i}(x_0)$ и Н.О.Д на этом шаге вычисляется между n и y−x [1].

Обоснование ρ-метода Полларда

Рассмотрим данный метод и рассчитаем его трудоемкость. Такая оценка базируется на известной теореме «парадоксе дня рождения».

Теорема

Пусть λ > 0. Для произвольной выборки из l + 1 элементов, каждый из которых меньше q, где $l = \sqrt{2\lambda q}$, вероятность того, что два элемента будут равными $p > 1 - e^{-\lambda}$.

Вероятность $p = 0,5$ в парадоксе дня рождения получается при λ ≈ 0, 69.

Допустим порядок элементов $\{u_n\}$ состоит из разностей $\left|x_i - x_j\right|$ проверяемых в процессе работы алгоритма. Установим новую последовательность $\{z_n\}$, где $z_n = u_n \bmod q$. q — меньший из делителей n. Все элементы последовательности $\{z_n\}$ меньше $\sqrt{n}$. Если рассматривать $\{z_n\}$ как случайную последовательность чисел, меньших q, то, соответственно парадоксу близнецов, вероятность того, что в числе первых l + 1 её членов попадутся два идентичных, превысит 1/2 при λ ≈ 0, 69, тогда l должно быть не меньше $\sqrt{2\lambda q} \approx \sqrt{1.4q} \approx 1.18\sqrt{q}$.

Если $z_i = z_j$, тогда $x_i - x_j \equiv 0 \bmod q \rightarrow x_i - x_j = kq$ для некоторого k∈Z. Если $x_i = x_j$, что осуществляется с огромной вероятностью, то искомый делитель q числа n будет найден как Н.О.Д.$\left(n, x_i - x_j\right)$. Ввиду того, что $\sqrt{q} \le n^{1/4}$, то с вероятностью превосходящей 0,5, делитель n может быть найден за 1,18 $n^{1/4}$ итераций.

Видно, что ρ-метод Полларда является вероятностным методом, который позволяет найти нетривиальный делитель q числа n за $O(q^{1/2}) \le O(n^{1/4})$ итераций. Сложность нахождения нетривиального делителя в этом методе зависит только от размера этого делителя, а не от размера числа n. В связи с этим, ρ-метод Полларда применяется в тех случаях, когда иные методы факторизации, которые зависят от размера n, являются низкоэффективными.

В иных случаях, последовательность $\{y_n\}$ будет зацикливаться (т.е. на определенном шаге t появляется $x_t = x_0$, затем последовательность повторяется), тогда следует заменить текущий элемент x_0 или полином F(x) на какой-нибудь другой [1].

Программная реализация и анализ алгоритма

Представим данный алгоритм в виде схемы и рассчитаем его структурную сложность. Оригинальный алгоритм Полларда представлен на рис. 1.

```
int ρ-Pollard (int n)
{ int x = random (1, n-2);
int y = 1; int i = 0; int stage = 2;
while(Н.О.Д.(n, abs(x − y)) = 1)
{
  if (i = = stage ){
    y = x;
     stage = stage*2; }
  x=x * x + 1(mod n);
  i=i + 1;
}
return Н.О.Д.(n, abs(x − y)); }
```

Рис.1. Оригинальный алгоритм Полларда

Разработка

Для быстрого нахождения Н.О.Д воспользуемся бинарным алгоритмом Эвклида. Его преимущество перед обычным алгоритмом заключается в использовании побитовых сдвигов, которые по разным оценкам имеют преимущество в скорости до 30%. Поэтому ожидается, что при использовании бинарного алгоритма Эвклида получим факторизацию чисел на тридцать процентов быстрее, чем получили бы при использовании обычного алгоритма.

Модификация алгоритма Полларда

Разработана модификация алгоритма, основанная на рекурсивном методе подсчёта факторизации числа, который работает на 27% быстрее, чем обычный оригинальный алгоритм Полларда, основанный на итерациях.

Для качественного анализа и сравнения двух алгоритмов, рассмотрим блок-схемы, представленные на рис. 2 и 3.

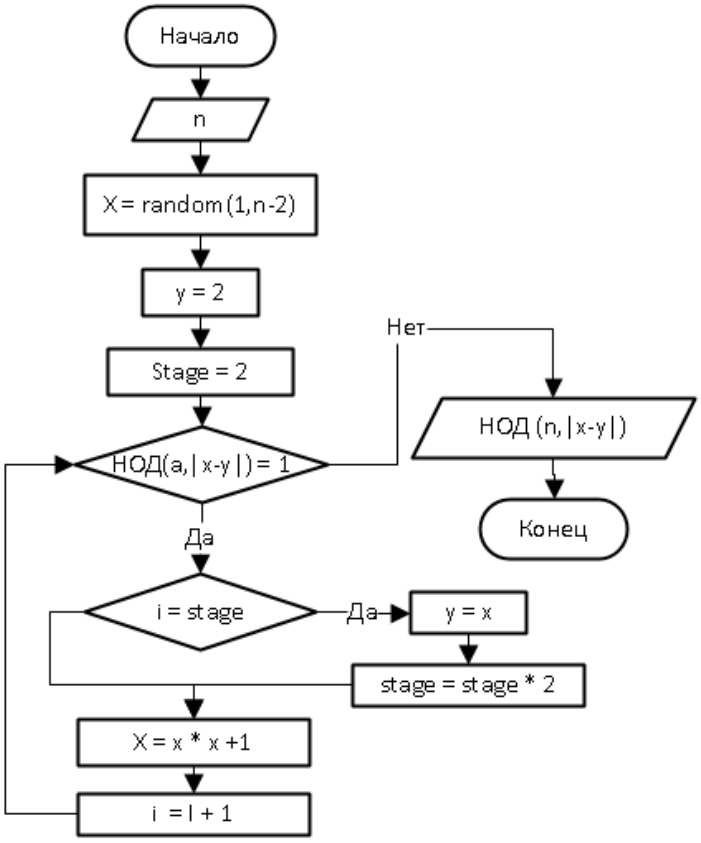


Рис. 2. Итерационный алгоритм Полларда

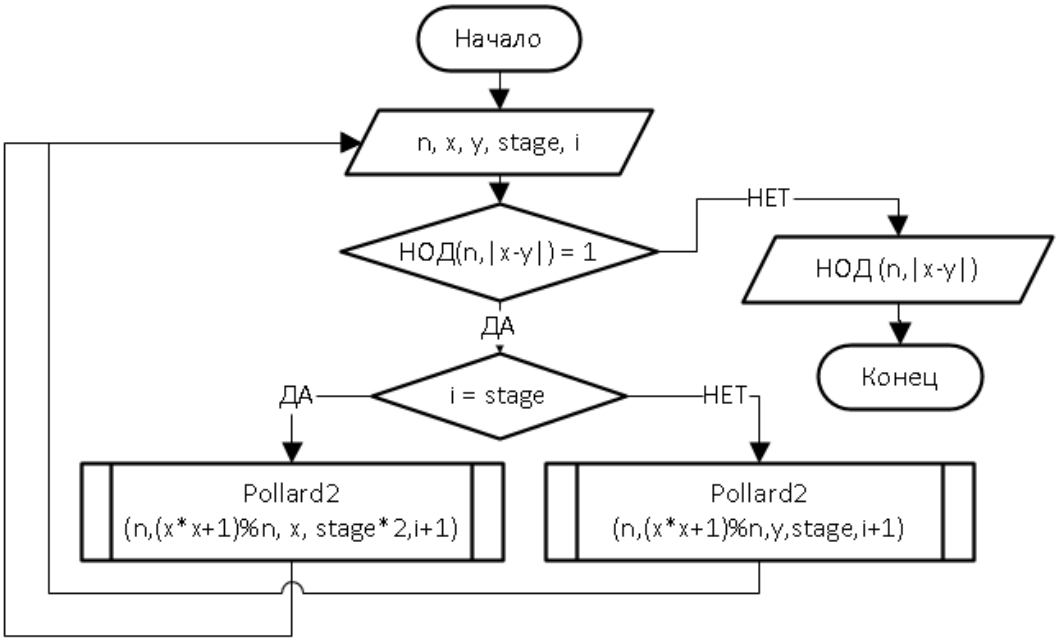


Рис. 3. Рекурсивный алгоритм Полларда

В таблице 1 представлен программный код двух алгоритмов.

Таблица 1

Программный код двух алгоритмов

Итерационный алгоритм Полларда	Рекурсивный алгоритм Полларда
Программный код	

intPollard3(intn,intx1)
{srand(time(NULL));
int y = 1; inti = 0; int stage=2;
while (gcd_bynary(n, abs(x1-y))==1)
{
if (i == stage)
{y = x1;
stage = stage * 2;}
x1 = (x1*x1 + 1) % n;
i = i + 1;}
Return gcd_bynary(n, abs(x1 - y));}
	int Pollard2(intn, intx, inty, intstage, inti)
	{if (gcd_bynary(n, abs(x - y)) == 1)
	{
	if (i == stage)
	{
	return Pollard2(n,(x*x+1)% n, x, stage*2, i+1);
	}
	return Pollard2(n,(x*x+1)%n, y, stage, i+1);
	}
	Else Return gcd_bynary(n, abs(x - y));
	}

Теперь сделаем оценку структурной сложности программного обеспечения по трем критериям, чтобы убедиться в преимуществе разработанного метода перед стандартной реализацией.

Критерий 1

Согласно данному критерию, граф потока управления программой должен быть проверен по самому малому набору маршрутов, проходящих через каждый оператор ветвления по каждой дуге [2].

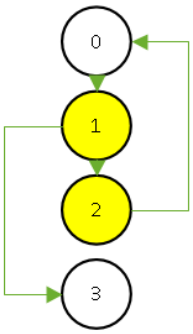
Прохождение по каждому маршруту происходит не более одного раза, т.к. повторная проверка дуг считается избыточной. В ходе проверки обеспечивается выполнение всех передач управления среди операторов программы и каждого оператора не меньше одного раза. Стоит отметить, что существуют алгоритмы, которые позволяют улучшить процесс получения минимального множества маршрутов по данному критерию [2].

В таблице 2 продемонстрированы графы и программные коды двух алгоритмов.

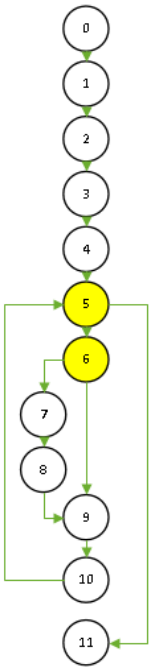
Таблица 2

Сравнение итерационного и рекурсивного алгоритмов Полларда

Рекурсивный алгоритм Полларда	

	int Pollard2(intn, intx, inty, intstage, inti) //0
	{
	if (gcd_bynary(n, abs(x - y)) == 1) //1
	{
	if (i == stage) //2
	{
	return Pollard2(n,(x*x+1)% n, x, stage*2, i+1); }
	return Pollard2(n,(x*x+1)%n, y, stage, i+1);
	}
	Else Return gcd_bynary(n, abs(x - y))}; //3

Итерационный алгоритм Полларда




```
int Pollard3(intn,intx1) //0
{
    srand(time(NULL));
    intx1 = rand() % n - 2 + 1; //1
    int y = 1; //2
    inti = 0; //3
    int stage=2; //4
    while (gcd_bynary(n, abs(x1-y))==1) //5
     {
          if (i == stage) //6
           {
                    y = x1; //7
                    stage = stage * 2; //8
           }
          x1 = (x1*x1 + 1) % n; //9
          i = i + 1; //10
     }
    Return gcd_bynary(n, abs(x1 - y));//11
}
```

Оценка алгоритмической сложности

Определим минимальный набор маршрутов, проходящих через всякий оператор ветвления и по каждой дуге, как представлено в таблице 3.

Таблица 3

Определение минимального набора маршрутов

int Pollard3(int n, int x1)	int Pollard2(int n, int x, int y, int stage, inti)
m1: 0-1-2-3-4-5-11; p1 = 1 m2: 0-1-2-3-4-5-6-7-8-9-10-5; p2 = 3 m3: 0-1-2-3-4-5-6-9-10-5; p3 = 3	m1: 0-1-3; p1 = 1; m2: 0-1-2-0-1-3; p2 = 2;

В соответствии с первым критерием оценки алгоритмической сложности, необходимое количество маршрутов для стандартной реализации ρ-метода Полларда равно трем, а для разработанного алгоритма — двум. Уровень сложности определяет количество вершин ветвления в графах:

$S1 = p_1 + p_2 = 1 + 2 = 3$ - *int Pollard2 (int n, int x, int y, int stage, int i)*

$S2 = p_1 + p_2 + p_3 = 1 + 3 + 3 = 7$ - *int Pollard3 (int n, int x1)*

Критерий 2

Данный критерий основан на анализе базовых маршрутов в программе, которые формируются и оцениваются на основе цикломатического числа графа потока управления программы. Для каждого линейно независимого цикла и ациклического участка программы определим число проверок. Число проверок определяется цикломатическим числом графа, которое определяется следующим соотношением:

$Z = n_B + 1 = 1 + 1 = 2$ - *int Pollard2(int n, int x, int y, int stage, int i)*

$Z = n_B + 1 = 2 + 1 = 2$ - *int Pollard3(int n, int x1)*

где n_B — число вершин ветвления. Далее необходимо выделить маршруты на заданном графе. Результаты представлены в таблице 4.

Таблица 4

Расчет маршрутов на заданных графах

intPollard3(intn, intx1)	intPollard2(int n, int x, int y, int stage, int i)
ациклические	
m_1: 0-1-2-3-4-5-11; p1 = 1;	m1: 0-1-3; p1 = 1;
циклические	
M_2: 5-6-7-8-9-10; p2 = 2;	M_2: 1-2; p2=2

Тестирование программы по указанным маршрутам позволит проверить все операторы ветвления программы. Метрику структурной сложности определим по следующему соотношению:

$Z = n_B + 1 = 2 + 1 = 2$ - *int Pollard3(int n, int x1)* $S_2 = \rho_1 + \rho_2 = 1 + 2 = 3$

$Z = n_B + 1 = 1 + 1 = 2$ - *int Pollard2(int n, int x, int y, int stage, int i)* $S_2 = \rho_1 + \rho_2 = 1 + 2 = 3$

Следующим шагом является построение матриц смежности, с помощью которых строится анализ графов. В этих матрицах содержится информация о структуре проверяемой программы.

Матрица смежности — квадратная матрица, в соответствующих ячейках которых располагаются единицы, если в графе потока управления программой имеется соответствующая дуга. В другом случае эта ячейка не заполняется.

Матрицы смежности двух алгоритмов показаны в таблицах 5 и 6.

Таблица 5

Матрица смежностиитерационного алгоритма intPollard3(intn, intx1)

	0	1	2	3	4	5	6	7	8	9	10	11
0												
1	1											
2		1										
3			1									
4				1								
5					1							
6						1						
7							1					
8								1				
9							1		1			
10										1		
11						1					1	

Таблица 6

Матрица смежности рекурсивного алгоритма intPollard2(intn, intx, inty, intstage, inti)

	0	1	2	3
0			1	
1	1			
2		1		
3		1		

После построения матриц смежности строится матрица достижимости. В ячейках этой матрицы единицы располагаются в позиции, соответствующей дуге (i,j). С помощью средств ЭВМ такую матрицу можно получить, возведя предыдущую матрицу смежности в степень, величина которой равна числу вершин без последней в исходном графе потока управления.

С помощью матрицы достижимости можно выделить циклы, отмечая диагональные элементы, равные единице, и идентичные строки. Матрицы достижимости двух алгоритмов показаны в таблицах 7 и 8.

Таблица 7

Матрица достижимости итерационного алгоритма intPollard3(intn, intx1)

	0	1	2	3	4	5	6	7	8	9	10	11
0												
1	1											
2	1	1										
3	1	1	1									
4	1	1	1	1								
5	1	1	1	1	1	1	1	1	1	1	1	
6	1	1	1	1	1	1	1	1	1	1	1	
7	1	1	1	1	1	1	1	1	1	1	1	
8	1	1	1	1	1	1	1	1	1	1	1	
9	1	1	1	1	1	1	1	1	1	1	1	
10	1	1	1	1	1	1	1	1	1	1	1	
11	1	1	1	1	1	1	1	1	1	1	1	

Таблица 8

Матрица достижимости рекурсивного алгоритма intPollard2(intn, intx, inty, intstage, inti)

	0	1	2	3
0	1	1	1	
1	1	1	1	
2	1	1	1	
3	1	1	1	

Критерий 3

С помощью этого критерия формируется полный состав базовых структур графа потока управления программой и анализируется каждый циклический и ациклический маршрут исходного графа программы, достижимого из всех этих маршрутов.

Согласно этому критерию были выделены все реально возможные маршруты управления. Расчет их представлен в таблице 9.

Таблица 9

Расчет всех возможных маршрутов управления

int Pollard3 (int n, int x1)	*int Pollard2 (int n, int x, int y, int stage, int i)*
m1: 0-1-2-3-4-5-6-7-8-9-10-5-11; p1 = 3; m2: 0-1-2-3-4-5-6-9-10-5-11; p2 = 3 m3: 0-1-2-3-4-5-6-7-8-9-10-5-6-9-10-5-11; p3 = 5 m4: 0-1-2-3-4-5-6-9-10-5-6-7-8-9-10-5-11; p4 = 5	m1 :0-1-2-0-1-3 p=2; m2: 0-1-3; p=1

Оценка структурной сложности программы для соответствующего алгоритма имеет следующий вид:

int Pollard3 (int n, int x1): $S_3 = p_1 + p_2 + p_3 + p_4 = 3+3+5+5 = 16$

int Pollard2 (int n, int x, int y, int stage, int i): $S_3 = p_1 + p_2 = 1+2 = 3$

Вывод по оценке структурной сложности

intPollard3(intn, intx1)

Исходя из полученных результатов расчёта метрик функции по трем критериям выделения маршрутов, можно сделать вывод, что стандартная функция ρ-методом Полларда имеет более высокую алгоритмическую сложность, чем разработанная и количество условных операторов в стандартной функции потребует не менее двух-трех тестовых вариантов исходных данных.

intPollard2(intn, intx, inty, intstage, inti))

Исходя из полученных результатов расчёта метрик функции по трем критериям выделения маршрутов, можно сделать вывод, что разработанная функция имеет более низкую алгоритмическую сложность по сравнению с предыдущей, из-за того, что используется один условный оператор, для которого достаточно проверить от одного до двух тестовых вариантов исходных данных.

Метрика Маккейба

Данная метрика позволяет оценить структурную сложность программных средств построенная на анализе потока управления от одного оператора к другому. Это поможет учесть логику построения программы при оценке ее сложности.

В соответствии с управляющим графом, число дуг m, число вершин n. Тогда цикломатическое число Маккейба равно:

intPollard3(intn, intx1): $Z = m - n + 2 = 13 - 12 + 2 = 3$

intPollard2(intn, intx, inty, intstage, inti): $Z = m - n + 2 = 4 - 4 + 2 = 2$

Тестирование

После реализации алгоритма было проведено тестирование на 3-х компьютерах с разной частотой процессора. Результаты тестирования представлены на таблице 10.

Таблица 10

Результат работы алгоритмов на различных процессорах

Тип процесса	Числа				Средняя эффективность по процессору	Общая средняя эффективность
	11	22	4567891	4567884		
Intel® Core(TM) i5-4200U CPU 2.3 GHz	1832/2597 30%	911/1235 26%	20098/22743 12%	827/1177 30%	27%	27,6%
Intel ® Core(TM) i3-2330M 2.20GHz	2845/4960 43%	3429/6641 49%	27443/44848 39%	4090/8468 52%	45.75%	
AMD Phenom(tm) II Quad-Core Processor 1.80HGz	3209/3485 8%	3057/3545 14%	11242/11735 5%	3057/3545 14%	10.25%	

В таблице 10 показано время выполнения функций высокочастотного счетчика. В числителе данной таблицы показаны данные по модифицированному авторами рекурсивному методу, а в знаменателе — по итерационному методу Полларда. Процент под счетчиком вычислениями показывает эффективность первого метода перед вторым.

Заключение

Разработанный автором рекурсивный алгоритм факторизации ρ-методом Полларда эффективнее на 27% по сравнению со стандартным итерационным алгоритмом. Модификации метода имеют схожий по функционалу, но разный уровень оценки структурной сложности, что было получено методом вычислительного эксперимента для трёх метрик структурной сложности программного обеспечения, в том числе, и по метрике Маккейба.

Исходя из этого, алгоритм факторизации ρ-методом Полларда, разработанный авторами и реализованный через рекурсию, надежнее и быстрее реализации стандартного алгоритма, что экспериментально подтверждено на практике. Использование данного алгоритма факторизации в шифре RSA позволит усложнить криптоанализ, который может быть основан на анализе вычисления времени выполнения операций в протоколе OpenSSL, а также, атак, основанных на анализе затрат электроэнергии [6].

Использование данного алгоритма позволит защитить данные от некоторых кибернетических атак, что способствует защите информации на предприятиях различного уровня, в том числе стратегического назначения.

Библиографический список

1. Ишмухаметов, Ш. Т. Электронное учебное пособие для студентов института вычислительной математики и информационных технологий / Ш. Т. Ишмухаметов, Р. Г. Рубцова. — Казань: Казанский федеральный университет, 2012. — 138 с.

2. Файловый архив студентов. Чувашский государственный педагогический университет им. И. Я. Яковлева [электронный ресурс] / И. С. Черников. — Оценка качества программного обеспечения. Практикум. Режим доступа: https://studfiles.net/preview/5850014/page:12/ (дата обращения 15.02.2018).

3. Википедия [электронный ресурс] / Ро-алгоритм Полларда — Режим доступа: https://ru.wikipedia.org/wiki/Ро-алгоритм_Полларда (дата обращения 12.02.2018).

4. Косяков, М. С. Введение в распределенные вычисления / М. С. Косяков. —Санкт-Петербург:НИУ ИТМО, 2014. — 155 с.

5. Желтов, С. А. Эффективные вычисления в архитектуре CUDA в приложениях информационной безопасности : дис. канд. техн. наук / С. А. Желтов. — Москва: ИИНТБ РГГУ, 2014. — 145 с.

6. Сонг, Я. Й. Криптоанализ RSA / Й. Я. Сонг. — Москва, Ижевск: НИЦ «Регулярная и хаотическая динамика», 2011. — 312 с.