Методы распараллеливания алгоритма Полларда решения задачи дискретного логарифмирования для систем с общей памятью

Горбенко И.Д., Качко Е.Г., Погребняк К.А.[1]

[1] *Харьковский национальный университет радиоэлектроники, просп. Ленина, 14, Харьков, Украина*

gorbenkoi@iit.kharkov.ua, ekachko@gmail.com, iitkostya@gmail.com

Аннотация. *В работе предлагается метод распараллеливания алгоритма Полларда решения задачи дискретного логарифмирования в группе точек эллиптической кривой для систем с общей памятью. На основе параллельного алгоритма Ооршота и Вейнера для систем с распределенной памятью строится комбинированный метод нахождения дискретного логарифма, который позволяет использовать преимущества как многопроцессорных, так и многоядерных систем. Далее анализируются известные функции итерирования точек в алгоритме Полларда и строится обобщенный метод Полларда для произвольной функции итерирования и систем с общей памятью. Предложенный подход к распараллеливанию алгоритма Полларда для систем с общей памятью позволил снизить время вычислений на 30-50%.*

Ключевые слова

Криптоанализ, эллиптическая кривая, дискретное логарифмирование, метод ρ-Полларда.

1 Введение

Известно, что стойкость значительной части криптографических примитивов, используемых в информационных системах, основывается на вычислительной сложности решения задачи дискретного логарифма в группе точек эллиптической кривой (ЭК), заданной над конечным полем. Следовательно, оценка стойкости криптографических примитивов заключается в выборе наиболее эффективного метода решения задачи дискретного логарифмирования в группе точек ЭК, из предложенных на данный момент, и дальнейшем получении эмпирических характеристик выбранного метода. Критерием эффективности методов дискретного логарифмирования, как правило, считается минимизация вычислительных ресурсов, то есть пространственно-временных показателей, и как результат удешевление процесса криптоанализа [1].

Методы дискретного логарифмирования в группе точек ЭК принято разделять на два типа. К первому типу относятся методы, которые применимы к ЭК с определенной структурой группы, а ко второму – методы логарифмирования для произвольной структуры группы. Следует отметить, что при правильном выборе ЭК атаки первого типа становятся нереализуемыми. Среди методов логарифмирования второго типа, наиболее эффективным считается вероятностный метод ρ-Полларда [2]. Фактически, метод ρ-Полларда включает в себя алгоритм построения псевдослучайной последовательности точек ЭК и алгоритм обнаружения коллизии. Повышение эффективности криптоанализа методом ρ-Полларда достигается за счет сокращения длины последовательности значений функции итерирования, улучшения алгоритма обнаружения коллизии и распараллеливания вычислений [3]. Идея распараллеливания метода ρ-Полларда состоит в том, что итерирование точек возлагается на клиентские рабочие станции, а обнаружение коллизии – на сервер [4]. Предложенный параллельный метод ρ-Полларда предполагает наличие высокопроизводительных сетей и не учитывает использование многоядерной архитектуры клиентских рабочих станций и сервера. Таким образом, актуальной задачей является исследование известных модификаций метода ρ-Полларда и адаптации их к многоядерной архитектуре рабочих станций.

Целью данной работы является разработка методов распараллеливания алгоритма ρ-Полларда для повышения эффективности использования на многоядерных рабочих станциях.

2 Метод ρ-Полларда

Пусть задана группа точек ЭК, которая будет обозначаться как $E(\mathbb{F}_p)$, такая что $\#E(\mathbb{F}_p) = n \cdot cof$, где n – простое число. Не ограничивая общности, можно предположить, что $p > 3$ и p – простое число. Обозначим подгруппу $E(\mathbb{F}_p)$ порядка n через G и зафиксируем порождающий элемент P.

Для произвольного элемента группы $Q = xP$ задача дискретного логарифмирования заключается в нахождении элемента $1 < x < n$.

2.1 Последовательный метод ρ-Полларда

Группа G представляется в виде объединения $G = S_1 \cup S_2 \cup S_3$, где S_i – произвольные множества приблизительно одинаковой мощности. Функция итерирования $f : G \to G$ определяется как

$$R_{i+1} = f(R_i) = \begin{cases} Q + R_i, & R_i \in S_1 \\ 2R_i, & R_i \in S_2 \\ P + R_i, & R_i \in S_3 \end{cases} \tag{1}$$

Пусть $R_{i+1} = a_i P + b_i Q$, тогда коэффициенты определяются следующим образом

$$a_{i+1} = \begin{cases} a_i, & R_i \in S_1 \\ 2a_i \mod n, & R_i \in S_2 \\ a_{i+1}, & R_i \in S_3 \end{cases} \tag{2}$$

$$b_{i+1} = \begin{cases} b_{i+1}, & R_i \in S_1 \\ 2b_i \mod n, & R_i \in S_2 \\ b_i, & R_i \in S_3 \end{cases} \tag{3}$$

Выбирая произвольное начальное значение R_0, строятся две последовательности $\{R_i\}_{i=0}^m$ и $\{R_{2i}\}_{i=0}^m$ пока не будет обнаружена коллизия $R_m = R_{2m}$.

Учитывая, что

$$\begin{aligned} R_m &= a_m P + b_m Q \\ R_{2m} &= a_{2m} P + b_{2m} Q \end{aligned} \tag{4}$$

можно вычислить

$$x = \frac{a_{2m} - a_m}{b_m - b_{2m}} \tag{5}$$

Отметим, что m зависит он начального значения R_0 и определяет вычислительную сложность метода Полларда.

2.2 Распараллеливание метода ρ-Полларда

2.2.1 Метод ρ-Полларда для систем с распределенной памятью

Метод ρ-Полларда для систем с распределенной памятью был предложен в работе [4]. Идея метода состоит в разделении итерирования точек между клиентскими рабочими станциями и поиска коллизии сервером. Пусть задано множество клиентских рабочих станций $S = \{S_i | i = 1..r\}$. Сервер определяет общесистемные параметры, некоторое подмножество $D \subset G$ и выполняет инициализацию рабочих станций. Клиентская рабочая станция S_i строит последовательность точек $\{R_{ij}\}_{j=0}^m \subset D$ и отправляет поэлементно точки на сервер. Если точка не содержится в базе данных, сервер добавляет точку в базу данных, иначе вычисляет значение дискретного логарифма.

2.2.2 Метод ρ-Полларда для систем с общей памятью

Рассмотрим метод распараллеливания алгоритма ρ-Полларда для систем с общей памятью. Идея такого метода состоит в распараллеливании отдельно функции итерирования и алгоритма обнаружения коллизии.

Функция итерирования распараллеливается на этапе вычисления последовательностей $\{R_i\}_{i=0}^{m}$ и $\{R_{2i}\}_{i=0}^{m}$. Следует отметить, что параллельное вычисление R_{i_0} и R_{2i_0} для фиксированного значения i_0 и последующее сравнение является неэффективным, что будет экспериментально показано в разделе 3. Это объясняется тем фактом, что накладные расходы, связанные с использованием потоков, являются вычислительно более затратными, чем вычисление $R_{2i_0} = f(R_{2i_0-2})$.

Таким образом, целесообразно вычислять последовательности так, чтобы накладные расходы нивелировались. Это может быть достигнуто организацией вычислений последовательностей вида

$$\{R_{iw+j}\}_{i=0}^{l} \text{ и } \{R_{2(iw+j)}\}_{i=0}^{l} \tag{6}$$

где w – это размер блока вычислений, $0 \leq j < w$, $l = \lceil \frac{m}{w} \rceil$.

Алгоритм ρ-Полларда для систем с общей памятью для двух ядер приведен на рисунке 1.

```
void IterationFunction( AffinePoint& InitSingleR,
                        AffinePoint& InitDoubleR,
                        AffinePoint* BlockSingleR,
                        AffinePoint* BlockDoubleR,
                        int nIndex)
{
    int nOffset  = nIndex* w;
    BlockSingleR = InitSingleR;
    BlockDoubleR = InitDoubleR;

    for(int i = nOffset; i < nOffset + w; i++)
    {
        #pragma omp parallel sections
        {
            #pragma omp section
            {
                BlockSingleR[i+1] = f(BlockSingleR[i]);
            }
            #pragma omp section
            {
                BlockDoubleR[i+1] = f(BlockDoubleR[i]);
                BlockDoubleR[i+1] = f(BlockDoubleR[i+1]);
            }
        }
    }
}
```

Рис. 1. Алгоритм итерирования точек для систем с общей памятью

Функция обнаружения коллизии в методе ρ-Полларда сравнивает значения R_m и R_{2m}. Такое сравнение можно распараллелить, если использовать алгоритм итерирования для систем с общей памятью. Результатом выполнения функции итерирования точек являются два множества точек $\{R_i\}_{i=0}^{w}$ и $\{R_{2i}\}_{i=0}^{w}$, сравнение которых осуществляется по блокам, то есть

$$\begin{aligned} R_i &= R_{2i}, i = 1 \ldots \frac{w}{2} \\ R_i &= R_{2i}, i = \frac{w}{2} \ldots w \end{aligned} \tag{7}$$

2.2.3 Комбинированный метод ρ-Полларда

Метод ρ-Полларда для систем с распределенной памятью [4] может быть расширен для использования на многоядерных рабочих станциях. Идея метода состоит в том, что итерирование точек клиентскими рабочими станциями происходит в соответствии с алгоритмом, приведенным на рисунке 1, что есть клиентская рабочая станция S_i строит последовательность точек $\{R_{ij}\}_{j=0}^{w}$. Далее рабочая станция S_i выбирает подмножество точек $\{R_{ij}\}_{j=0}^{w} \cap D$ и отправляет его серверу.

Проверка на принадлежность подмножеству осуществляется в параллельном режиме:

$$\begin{aligned} R_{ij} \in D, i = 1 \dots \frac{w}{2} \\ R_{ij} \in D, i = \frac{w}{2} \dots w. \end{aligned} \tag{8}$$

Сервер добавляет точки $\{R_{ij}\}_{j=0}^{w} \cap D$ в базу данных до тех пор, пока не найдет уже существующую точку.

2.2.4 Обобщение метода ρ-Полларда для систем с общей памятью на произвольную функцию итерирования

Рассмотрим известные модификации функции итерирования для последовательного алгоритма ρ-Полларда, а именно обобщенную функцию итерирования Полларда [3], функцию итерирования Теске [5] и смешанную функцию итерирования Теске [5].

Обобщенная функция итерирования Полларда представляется в виде:

$$R_{i+1} = f_{PG}(R_i) = \begin{cases} W_1 + R_i, & R_i \in S_1 \\ 2R_i, & R_i \in S_2 \\ W_2 + R_i, & R_i \in S_3 \end{cases} \tag{9}$$

где $W_1 = t_1 P$, $W_2 = t_2 Q$, t_1,t_2 - случайным образом выбранные числа, такие что $0 < t_1, t_2 \leq n$.

Предположим, что предварительно вычислено r значений $W_i = t_i^1 P + t_i^2 Q$, $i = 1..r$. Определим функцию $\nu : G \to \{1..r\}$, тогда функция итерирования Теске может быть описана следующим образом:

$$R_{i+1} = f_{TA}(R_i) = R_i \cdot W_{\nu(R_i)}, \nu(R_i) \in 1..r \tag{10}$$

Смешанная функция итерирования Теске записывается как:

$$R_{i+1} = f_{TM}(R_i) = \begin{cases} R_i \cdot W_{\nu(R_i)}, & \nu(R_i) \in 1..r \\ 2R_i, & \nu(R_i) \notin 1..r \end{cases} \tag{11}$$

Отметим, что независимо от выбора функции итерирования $F \in \{f, f_{PG}, f_{TA}, f_{TM}\}$, последовательно вычисляются множества

$$\begin{aligned} F(R_0), F(R_1) \dots F(R_m) \\ F(R_0), F(R_1) \dots F(R_{2m}) \end{aligned} \tag{12}$$

Следовательно, тривиальным образом можно распространить идею алгоритма ρ-Полларда для систем с общей памятью на произвольную функцию итерирования как вычисления вида:

$$\begin{aligned} \{F(R_{ij})\}_{j=0}^{w}, i = 0 \dots l \\ \{F(R_{ij})\}_{j=0}^{2w}, i = 0 \dots 2l \end{aligned} \tag{13}$$

3 Оценка вычислительной сложности метода ρ-Полларда для систем с общей памятью

Для реализации описанных выше алгоритмов использовались: процессор Intel (R) Core (TM)2 Duo CPU E6850 3.00GHz, ОЗУ – 2Gb , ОС – Windows 7. В таблице 1 приведены теоретические оценки времени выполнения и требуемой памяти для классического и параллельного методов Полларда для двуядерных рабочих станций. Ограничение случаем для двуядерных процессоров вызвано тем, что итерируются параллельно не более двух последовательностей. Через t обозначено время выполнения функции итерирования, а через θ – вероятность того, что точка ЭК удовлетворяет заданному критерию.

Таблица 1. Оценка классического и параллельного методов Полларда

№	Методы Полларда	Время выполнения	Количество памяти
1	Классический метод	$t\sqrt{\frac{\pi n}{2}}$	1
2	Параллельный метод с общей памятью	$\frac{t}{2}\sqrt{\frac{\pi n}{2}}$	1
3	Параллельный метод с разделенной памятью	$t\left(\sqrt{\frac{\pi n}{2}}+\frac{1}{\theta}\right)$	$\theta\sqrt{\frac{\pi n}{2}}$
4	Комбинированный метод	$\frac{t}{2}\left(\sqrt{\frac{\pi n}{2}}+\frac{1}{\theta}\right)$	$\theta\sqrt{\frac{\pi n}{2}}$

В таблице 2 приведено время выполнения в секундах (δ_t) для последовательного и параллельного методов Полларда для систем с общей памятью. Временная оценка проводится в зависимости от длины блока согласно алгоритму на рисунке 1.

Таблица 2. Временные показатели методов Полларда

№	Методы Полларда	δ_t (сек)	
		21 бит	27 бит
1	Классический метод	9.59973	354.191
2	Параллельный метод (w=1)	8.50658	344.427
3	Параллельный метод (w=4)	7.22392	308.106
4	Параллельный метод (w=8)	7.10515	278.301
5	Параллельный метод (w=128)	6.95188	246.382
6	Параллельный метод (w=512)	6.93911	245.298
7	Параллельный метод (w=1024)	6.98354	245.65
8	Параллельный метод (w=2048)	7.08218	256.072
9	Параллельный метод (w=4096)	7.21804	251.537

Отметим, что на последней итерации решение может быть найдено уже на первом элементе блока, однако, при распараллеливании значения функции итерирования (I) и последующие сравнения (C) будут вычислены для всего блока. Тем не менее, относительно общих вычислительных затрат, время вычисления избыточных операций (Δ_t) является незначительным, что показано в таблице 3.

Таблица 3. Временные показатели избыточных операций метода Полларда

№	Методы Полларда	Δ_t (мк)			
		21 бит		27 бит	
		I	C	I	C
1	Параллельный метод (w=4)	32	4,8	32,6	4,89
2	Параллельный метод (w=8)	64,5	9,4	87,1	9,24
3	Параллельный метод (w=128)	1000	170	1031	139
4	Параллельный метод (w=512)	4041	545	4050	548
5	Параллельный метод (w=1024)	8047	1115	8169	1148
6	Параллельный метод (w=2048)	16162	2411	16488	2477
7	Параллельный метод (w=4096)	32582	4881	33039	4981

4 Заключение

В работе предложен метод распараллеливания алгоритма Полларда решения задачи дискретного логарифмирования в группе точек эллиптической кривой для систем с общей памятью. На основе параллельного алгоритма Ооршота и Вейнера [4] для систем с распределенной памятью предложен комбинированный метод нахождения дискретного логарифма, который позволяет использовать преимущества как многопроцессорных, так и многоядерных систем. Проанализированы известные функции итерирования точек в алгоритме Полларда и построен обобщенный метод Полларда для произвольной функции итерирования и систем с общей памятью. Получены аналитические выражения для предложенных параллельного метода Полларда и комбинированного метода Полларда для двуядерных процессоров. В работе приведены эмпирические временные показатели для параллельного метода Полларда для систем с общей памятью, которые согласуются с аналитическими выражениями. Следует отметить, что моделирование проводилось для двуядерных процессоров, что обусловлено наличием двух последовательностей в алгоритме обнаружения цикла. Предложенный подход к распараллеливанию алгоритма Полларда позволил снизить время вычислений на $30 - 50\%$. Исследованы временные показатели избыточных операций для параллельного алгоритма Полларда для систем с общей памятью и эмпирически показана их низкая вычислительная нагрузка.

В дальнейшем планируется провести моделирование для комбинированного метода и для различных итеративных функций, обобщить полученные результаты на случай произвольного числа ядер и на кривые с большей битовой длиной порядка подгруппы.

Отметим также, что анализировался только один алгоритм обнаружения цикла, а именно алгоритм Флойда [3]. Необходимо также проанализировать альтернативные алгоритмы, например, алгоритм Брента [3]. Структура алгоритма Брента смогла бы позволить построить конвейер вычислений, на котором при вычислении следующего блока точек ЭК на одном ядре, происходит поиск коллизии на другом.

В дальнейшем также необходимо исследовать влияние начального значения для функции итерирования на выбор длины блока. Отметим, что исходя из экспериментальных данных, оптимальной длиной блока является $w = 1024$ или $w = 2048$.

Список литературы

[1] Качко Е.Г., Погребняк К.А. Параллельный метод Полларда решения задачи дискретного логарифмирования в группе точек эллиптической кривой // Параллельные вычислительные технологии (ПАВТ–2012): труды международной научной конференции (Новосибирск, 26 – 30 марта, 2012 г.). Челябинск: Издательский центр ЮУрГУ, 2012. – С. 723.

[2] Pollard J.M. Monte Carlo methods for index calculus computation (mod p) // Mathematics of Computation. July 1978. Vol. 32, No. 143. P. 918-924.

[3] Bai S., Brent R. P., On the efficiency of Pollard's rho method for discrete logarithms // Fourteenth Computing: The Australasian Theory Symposium (CATS 2008), January 22–25, 2008, Wollongong, NSW, Australia, Proceedings. CRPIT, 77. Harland, J. and Manyem, P., Eds. ACS. P. 125–131.

[4] P. Van Oorschot, Wiener M. Parallel collision search with cryptanalytic applications. // Journal of Cryptology. 1999. Vol. 12. P. 1–28.

[5] Teske E. Speeding up Pollard's rho method for computing discrete logarithms // Algorithmic Number Theory, Third International Symposium, ANTS-III, LNCS 1423, Springer, 1998. P. 541–554.