УДК 004.421

ПРАКТИЧЕСКОЕ ПРИМЕНЕНИЕ АЛГОРИТМА RSA ДЛЯ ШИФРОВАНИЯ

Г. М. ПОСНИЧЕНКО
(Полоцкий государственный университет имени Евфросинии Полоцкой, Беларусь)

Аннотация. *Представлено практическое применение алгоритма RSA для шифрования текстовой информации в файлах. Рассмотрен анализ стойкости алгоритма RSA.*

Ключевые слова: *шифрование, RSA, алгоритм.*

Алгоритм RSA (Rivert, Shamir, Adleman) известен с 1976 года и получил большое распространение благодаря высокой криптостойкости и лаконичности математического фундамента, на котором он основывается. Алгоритм является ассиметричным. Информация шифруется с помощью открытого ключа, и после передачи адресату расшифровывается открытым ключом.

Ассиметричное шифрование используется во многих криптографических протоколах, в том числе – в https.

Ключевую роль в вычислениях играют теория сравнений и функция Эйлера. Рассмотрим математическую основу алгоритма.

Выбираются два больших простых числа p1 и p2 и вычисляется произведение N = p1*p2. Функция Эйлера ϕ(N) – это количество натуральных чисел, взаимно простых с N и меньших, чем N. Так как числа p1 и p2 простые, то ϕ(N) = (p1–1)*(p2–1).

После этого выбирается число k1, являющееся взаимно простым с числом ϕ(N) и меньшим его. Пара чисел (k1 и N) станет открытым ключом для шифрования. А для закрытого ключа нужно вычислить такое число k2, которое является обратным по модулю ϕ(N) для числа : $k2 = k1^{-1} (\mathrm{mod}\ \phi(N))$. Пара чисел (k2 и N) будет выполнять роль закрытого ключа.

Для нахождения числа k2 существенную роль играет теорема Эйлера, которая гласит, что для взаимно простых чисел a и n выполняется сравнение: $a^{\phi(n)} \equiv 1 (\mathrm{mod}\ n)$.

Отсюда следует, что $a^{\phi(n)-1} \equiv a^{-1} (\mathrm{mod}\ n)$. И для вычисления закрытого ключа получаем : $k2 = k1^{\phi(N)-1} (\mathrm{mod}\ N)$.

И теперь, для использования криптографического алгоритма RSA, нам нужно научиться вычислять по модулю N степени a^e для больших чисел.

Конечно, умножение e раз в цикле res = (res * a) % N даст нам нужный результат, но это хоть и простой, но нерациональный способ. Быстрый способ возведения в степень основан на двоичном представлении показателя степени

e = (e_t, e_{t-1}, …, e_1, e_0) и последовательном вычислении a_2, a_4, a_8, …, a^(2^t). Причем, умножение на число a в нужной степени производится тогда, когда в двоичном представлении числа e в этой позиции стоит 1.

Вычисления можно оформить в виде функции:

```
long long degree(long long a, long long e, long long m) {
        long long res = 1;
        while(e>0)
        {
        if(e%2 == 1) res = res * a % m;
        a = a * a % m;
        e /= 2;
        }
return res;
}
```

По этой функции модульное возведение в степень может быть вычислено не более, чем за 2*t операций, где t – старший бит в двоичном представлении числа e [1].Алгоритм работы функции degree() понятен, но компьютерная арифметика позволяет работать с целыми числами максимальной длины в 8 байт. На ключах такой длины нельзя строить цифровую подпись или иную серьезную систему шифрования. Для надежной криптосистемы применяют ключи длиной 1024 бит, или 2048 бит. И для этого нужны функции для работы с большими числами, представленными в двоичном или 16-ричном виде.

На чем основана стойкость алгоритма RSA? Для взлома шифра нужно разложить на множители число N. И сделать это очень-очень сложно, если N велико. Но чисто умозрительно, это можно сделать, но понадобится очень много времени. Например, по словам дававшего интервью генерала ФСБ, для взлома шифра, применяемого для правительственной связи, понадобится около 90 лет.

А для разового обмена зашифрованной информацией достаточно перед сеансом связи менять ключи и пользоваться малым средством – модульным вычислением в обычной компьютерной арифметике, то есть, применять функцию degree().

Глянем еще раз на функцию. В операторе a = a * a % m; происходит умножение двух целых чисел. И чтобы не потерять разряды и результат умножения остался 64-битным, число a должно быть 32-битным.

Вот, для примера и возьмем два таких простых числа (таблицы простых чисел в большом диапазоне хорошо представлены в интернете):

p1 = 65521, p2 = 65287, N = 4277669527, ϕ(N) = 4277538720.

k1 = 65537, k2 = degree(k1, ϕ(N)-1,N) = 1575251787.

Теперь, для того, чтобы зашифровать число x, вычисляем y = degree(x, k1, N).

Число y передаем адресату, который с помощью закрытого ключа определит значение x следующим образом : x= degree(y, k2, N). Таким образом идет шифрование и расшифровка.

И остался еще вопрос о кодировке текста или байтов из файла. Ведь в шифровании участвуют только целые числа. Значит, текст перед шифрованием нужно представить в числовом виде. Самое простое – использовать ASCII-коды.

Но можно придумать синтаксическую обработку текста с использованием псевдо-алфавита. Зачем нам нужен весь набор ASCII-кодов? Достаточно взять в свой алфавит 33 русские буквы, например строчные, с кодами 1, 2, …, 33. И если в алфавите есть флаг (большие/малые), то отдельно уже не нужны коды для заглавных букв. Достаточно в нужном месте выставить соответствующее значение переключателя. И к тем же позициям можно привязать и латинские буквы со своим переключателем (рус/лат). И еще нужно добавить в алфавит цифры и знаки препинания. В итоге, можно уложиться в 6-битовую кодировку. А дальше, чтобы остаться в границах 32-битовых чисел, пять 6-битовых символов упаковать в одно число и отправить на шифрование. Такая обработка текста усилит стойкость от взлома, но потребует кроме ключей шифрования на обоих сторонах иметь функции, которые упаковывают и распаковывают исходный текст.

ЛИТЕРАТУРА

1. Разинков, Е.В. Теория чисел и ассиметричная криптография : конспект лекций / Е.В. Разинков. – Казань: Казан. ун-т, 2020. – 53 с.