**RSA – алгоритм шифрования с открытым ключом**

В данной статье описан асимметричный алгоритм шифрования RSA. Рассмотрено описание RSA, и всё, что требуется для реализации RSA: рекомендации по выбору размеров блока данных и ключей, работа с числами большой разрядности, алгоритмы выработки простых чисел, нахождения наибольшего общего делителя, возведения числа в большую степень, работа с отрицательными числами.

RSA относится к так называемым асимметричным алгоритмам, у которых ключ шифрования не совпадает с ключом дешифровки. Один из ключей доступен всем (так делается специально) и называется открытым ключом, другой хранится только у его хозяина и неизвестен никому другому. С помощью одного ключа можно производить операции только в одну сторону. Если сообщение зашифровано с помощью одного ключа, то расшифровать его можно только с помощью другого. Имея один из ключей невозможно (очень сложно) найти другой ключ, если разрядность ключа высока.

**Описание RSA**

Алгоритм RSA состоит из следующих пунктов:

1. Выбрать простые числа p и q
2. Вычислить n = p \* q
3. Вычислить m = (p - 1) \* (q - 1)
4. Выбрать число d взаимно простое с m
5. Выбрать число e так, чтобы e \* d = 1 (mod m)

Числа e и d являются ключами RSA. Шифруемые данные необходимо разбить на блоки - числа от 0 до n - 1. Шифрование и дешифровка данных производятся следующим образом:

* Шифрование: b = ae (mod n)
* Дешифровка: a = bd (mod n)

Следует также отметить, что ключи e и d равноправны, т.е. сообщение можно шифровать как ключом e, так и ключом d, при этом расшифровка должна быть произведена с помощью другого ключа.

**Нахождение простых чисел**

В первом пункте алгоритма RSA сказано, что необходимо выбрать два простых числа p и q. Как это сделать, если числа имеют большую разрядность? Простой способ - деление предполагаемого простого числа на все числа меньшие его не работоспособен уже с 32-битными числами (требуется очень много времени на выполнение).

В данном случае, для выработки простых чисел используют вероятностные методы, один из которых будет здесь представлен. Вероятностные методы не дают полной гарантии, что найденное число простое, но при достаточно небольшом количестве операций позволяют получить очень высокую вероятность этого.

**Алгоритм поиска простых чисел**

1. N - нечетное число. Найти s и t, удовлетворяющие уравнению: N - 1 = 2s \* t
2. Случайным образом выбрать число a, 1 < a < N
3. Если N делится на a, перейти к пункту 6
4. Если условие at = 1 (mod N) выполняется, перейти к пункту 2
5. Если найдется такое k, 0 <= k < s, что a2k \* t = -1 (mod N), перейти к пункту 2
6. Число N - составное: выбрать другое нечетное число N, перейти к пункту 1

Если для какого-либо числа N проверено m чисел a, то математически доказанная вероятность того, что число является составным будет равняться 4-m (на самом деле вероятность намного меньше этого значения). Исходя из этого, считаю нужным для числа N, состоящего из p бит проверить p различных значений a. Если во время этого не обнаружится, что N - число составное, то вероятно (это моя личная оценка), что число N является простым.

Замечу, что число s не может быть большим количества бит в числе. Числа s и t находятся при помощи двоичного сдвига числа N - 1, пока младший разряд не станет 1. В результате s - количество сдвигов, t - число N - 1 после сдвига.

**Нахождение взаимно простых чисел**

На шаге 4 алгоритма RSA необходимо найти число d взаимно простое с m, т.е. не имеющее общих делителей с ним, кроме единицы. Число d должно быть меньше m, т.о. разрядность числа d равна сумме бит в числах p и q. Для нахождения взаимно простых чисел используется алгоритм Евклида, который находит наибольший общий делитель двух чисел. Если найденный делитель больше единицы, то необходимо выбрать другое число d и повторить проверку.

**Алгоритм Евклида**

1. Исходные числа a и b
2. Вычислить r - остаток от деления a на b: a = b \* q + r
3. Если r = 0, то b - искомое число (наибольший общий делитель), конец
4. Заменить пару чисел <a, b> парой <b, r>, перейти к пункту 2

При вычислении наибольшего общего делителя с помощью алгоритма Евклида будет выполнено не более 5 \* p операций деления с остатком, где p есть количество цифр в десятичной записи меньшего из чисел a и b. На практике алгоритм работает очень быстро.

**Решение уравнения a \* x + b \* y = 1**

В 5-м пункте алгоритма RSA предполагается нахождение такого числа e, чтобы e \* d = 1 (mod m). Для этого нужно использовать модифицированный алгоритм Евклида, который работает только если числа d и m взаимно просты. Вычисление числа e сводится к решению уравнения m \* x + d \* e = 1 в натуральных числах. Число x не существенно.

**Алгоритм решения уравнения a \* x + b \* y = 1**

1. Необходимо определить матрицу E = D:\01_Computer Sciences\C#\info from www\0 -Crypto algorithms\RSA\RSA  алгоритм шифрования, ключи RSA. Описание и реализация алгоритма RSA._files\matrix1.png
2. Вычислить r - остаток от деления a на b: a = b \* q + r
3. Если r = 0, то второй столбец матрицы дает решение: D:\01_Computer Sciences\C#\info from www\0 -Crypto algorithms\RSA\RSA  алгоритм шифрования, ключи RSA. Описание и реализация алгоритма RSA._files\matrix2.png, конец
4. E = E \* D:\01_Computer Sciences\C#\info from www\0 -Crypto algorithms\RSA\RSA  алгоритм шифрования, ключи RSA. Описание и реализация алгоритма RSA._files\matrix3.png
5. Заменить пару чисел <a, b>, парой <b, r>, перейти к пункту 2

В данном алгоритме все вычисления можно производить по модулю большего из чисел a и b. Отрицательное число -q заменяется положительным, полученным путем вычитания числа q из числа, взятого в качестве модуля. Например, если из чисел a и b большим является число b, то все вычисления можно производить по модулю числа b, при этом -q будет представлено как b - q. Скорость работы алгоритма и количество производимых им операций примерно равно соответствующим параметрам алгоритма Евклида, описанного выше.

**Большие числа и работа с ними**

На данный момент времени рекомендуется в качестве чисел e и d брать числа, длиной не менее 768 бит. Чтобы подобрать ключ такой длины потребуется $1000000 и примерно год времени. Ключ в 1024 бит является достаточно надежным для обычных целей шифрования. Для повышенной безопасности рекомендуется брать ключи размером 2048 бит. Т.о. числа p и q должны иметь разрядность вдвое ниже чисел e, d, m и n (p и q рекомендуется брать примерно одного порядка, но не слишком близко друг к другу).

Действия со столь большими числами требуют специальных алгоритмов и структур данных, которые мне бы хотелось здесь рассмотреть. Основные вопросы этой области: как хранить эти числа, как их складывать, умножать, делить, брать остаток от деления, возводить в большую степень по модулю целого числа.

**Хранение больших чисел, алгебраическое сложение, умножение**

По опыту написания RSA знаю, что числа лучше всего хранить в массиве 2-х байтовых переменных. Об отрицательных числах можно забыть: использоваться не будут, т.к. их всегда можно заменить на обратные по модулю (см. решение уравнения a \* x + b \* y = 1). Переменные в 2 байта удобны при умножении, если вы пишете на языке высокого уровня: результат будет 4-х байтовым и потом его можно разделить на две части для дальнейшей обработки.

Умножение производится с помощью обычного школьного алгоритма умножения "в столбик". Есть также и более эффективные алгоритмы умножения, но у них существует понятие "обменной точки" - необходимой длины числа для того, чтобы алгоритм начал выигрывать по скорости в сравнении с простым алгоритмом. Опять же по опыту знаю: обычный алгоритм работает вполне нормально. Конечно, для ускорения работы лучше использовать более быстрый алгоритм, но и более сложный в реализации.

Сложение и вычитание также производятся "в столбик". Здесь альтернативы нет - быстрее сложить или вычесть вряд ли получится, хотя, кто знает...

Алгоритм деления и взятия остатка мне пришлось реализовывать через двоичную систему счисления (в отличие от умножения, которое производилось над 2-х байтовыми числами). Данная операция получается немного медленнее, но алгоритм из того же ряда: школьный "уголок". Здесь рекомендую самостоятельно поискать лучшее решение для увеличения производительности.

**Алгоритм быстрого возведения в степень**

В алгоритме RSA очень много возведений в степень по модулю натурального числа. Конечно же, не нужно производить триллиарды умножений, а затем брать остаток от деления числа из миллиардов цифр. Остаток от деления берется после каждого умножения. Таким образом, при перемножении двух чисел, состоящих из k бит потребуется 2 \* k-битное число, которое затем делится на модуль и получается остаток, опять же состоящий из k бит (если модульное число состоит из k бит).

Сложность этого алгоритма может быть оценена как O(ln m), где m - модуль, по которому производится умножение. Запись O(ln m) означает, что для реализации алгоритма потребуется порядка ln m операций. Например, если число имеет разрядность 1024 бит (при этом длина m не менее 1024 бит), то умножение по модулю необходимо будет провести порядка ln m = ln 21024 = 710 раз, что относительно немного.

Алгоритм вычисления ad (mod m):

1. Число d представить в двоичной системе счисления: d = d0 \* 2r + ... + dr - 1 \* 2 + dr, где di - цифры в двоичном представлении, равные 0 или 1, d0 = 1
2. Положить a0 = a, затем для i = 1, ... , r вычислить ai = (ai - 1) 2 \* adi (mod m)
3. ar есть искомое число ad (mod m)