Лабораторная работа №6

Пакеты и приложения

Теоретическая часть

Одним из наиболее значительных прорывов в криптографии двадцатого столетия была разработка шифрования с открытым ключом. Алгоритмы с открытым ключом, или *асимметричные алгоритмы*, базируются на использовании отдельных шифровального (открытого – public) и дешифровального (закрытого – private) ключей.

В алгоритмах с открытым ключом требуется, чтобы закрытый ключ было невозможно вычислить за приемлемое время по открытому ключу. Исходя из этого требования шифровальный ключ может быть доступным кому угодно без какого-либо ущерба безопасности алгоритма шифрования.

В асимметричных криптосистемах отрытый ключ и криптограмма могут быть отправлены по не защищённым каналам. Концепция таких систем основана на применении однонаправленных функций.

В качестве примера однонаправленной функции может служить целочисленное умножение, использованное в алгоритме RSA. Прямая задача – вычисление произведения двух больших натуральных чисел p и q, n = p*q. Это относительно несложная задача для компьютера.

Обратная задача – факторизация или разложение на множители большого натурального числа практически неразрешима при достаточно больших значениях n. Например, если p ≈ q - простые, а их произведение $n \approx 2^{664}$, для разложения этого числа на множители потребуется около 2^{23} операций, что практически невозможно выполнить за приемлемое время на современных ЭВМ.

Алгоритм RSA носит инициалы его изобретателей. Он был предложен тремя исследователями-математиками Рональдом Ривестом (R.Rivest), Ади Шамиром (A.Shamir) и Леонардом Адлеманом (L.Adleman) в 1977-78 годах. Этот алгоритм имеет важное значение, поскольку может быть использован как для шифрования, так и для цифровых подписей.

Стойкость алгоритма RSA определяется сложностью разложения больших чисел на простые множители. Возможно, что существует криптоанализ шифра RSA и без использования операции разложения на множители, но никто до сегодняшнего дня не нашел как это сделать.

Схема алгоритма шифрования данных RSA :

1. Генерируется два простых числа *p* и *q* (по 100 или более десятичных цифр).

2. Вычисляется их произведение $n=pq$.

3. Вычисляется функция Эйлера: $\varphi(n) = (p-1)(q-1)$.

4. Выбирается натуральное число $e < \varphi(n)$ взаимно простое с числом $\varphi(n)$, то есть такое, что НОД$(e, \varphi(n)) = 1$.

5. Число d вычисляется из условия $ed \equiv 1 \ (mod\ \varphi(n))$, то есть d - обратный элемент по модулю $\varphi(n)$. Сравнение равносильно уравнению в целых числах: $ed+\varphi(n)y=1$, где d и y - неизвестные).

6. Два числа (e,n) – публикуются как открытый ключ.

7. Число d хранится в строжайшем секрете – это и есть закрытый ключ, который позволит читать все послания, зашифрованные с помощью пары чисел (e,n).

Алгоритм шифрования сообщения M и его расшифровка:

1.Исходный текст сообщения разбивается на блоки $M_1, M_2,..., M_n$ ($M_i = 0, 1, 2,..., n$)

2. Каждый блок шифруется: $C_i= (M_i)^e \bmod n$.

3.Зашифрованный текст $C_1 C_2 ...C_n$ может передаваться по открытым каналам.

4.Обладатель закрытого ключа d расшифровывает каждый блок: $M_i = C^d_i \bmod n$.

Пример для маленьких чисел (специально вставлен изображением, чтобы не исказилось отображение формул):

1. Пусть p=3, q=11.
2. n=pq=33.
3. φ (n)=2*10=20.
4. e=7 (берем любое взаимно-простое с 20).
5. Из сравнения 7*d ≡ 1 (mod 20), находим d=3.
5. Пусть сообщение для зашифровки: 312
6. Разбиваем исходное сообщение на блоки: $M_1 = 3$, $M_2 = 1$, $M_3 = 2$.
7. Шифруем их:

   $C_1=3^7 \bmod 33 = 2187 \bmod 33 = 9$,

   $C_2=1^7 \bmod 33 = 1$,

   $C_3=2^7 \bmod 33 = 2187 \bmod 33 = 29$.
8. Отправляем последовательно зашифрованное сообщение: 9,1,29.
9. Получатель расшифровывает закрытым ключом d=3:

   $M_1=9^3 \bmod 33 = 729 \bmod 33 = 3$,

   $M_2=1^3 \bmod 33 = 1$,

   $M_3=29^3 \bmod 33 = 24389 \bmod 33 = 2$.
10. Получатель читает исходное сообщение: 312.

Основным недостатком шифра RSA и других алгоритмов с открытым ключом, является их низкая производительность, по сравнению с алгоритмами с секретным ключом. Алгоритм RSA уступает по скорости сопоставимым реализациям алгоритма DES в 100, а то и в 1000 раз.

Хотя шифр RSA еще никому не удалось раскрыть, прогресс в математике может сделать этот шифр устаревшим. При обнаружении эффективного способе разложения больших чисел на множители шифр RSA можно будет легко раскрывать. Хотя, с другой стороны, возможно со временем будет доказано, что менее сложного алгоритма дешифрования RSA вообще не существует.

Задание

Изучить возможности классов BigInt в Scala и класса BigInteger в Java. Используя эти классы, реализовать на Scala алгоритм RSA и вспомогательные функции для чисел любой размерности и оформить их в отдельный пакет:

- def isPrime (n: BigInt) - возвращает true, если число n простое и false, если составное
- def primes(n: BigInt, m: BigInt) - возвращает список (List) всех простых чисел от n до m включительно
- def randomPrime(bits: Int) - возвращает случайное простое число BigInt с длиной в битах: bits
- def encrypt(s: String, e: BigInt, n: BigInt) - возвращает зашифрованную строку s открытым ключом (e, n); результатом должен быть массив приведенный к строке: 123456,234567,345678,...
- def decrypt(s: String, d: BigInt, n: BigInt) - расшифровывает s секретным ключом (d, n); результатом должна быть исходная строка до ее шифрования

Написать с использованием данного пакета утилиту (по выбору: либо для командной строки, либо GUI с использованием Spring), позволяющую решать следующие задачи:

- генерировать диапазон простых чисел и позволять их выбирать из списка
- генерировать ключи RSA, основанные на простых числах любой размерности
- разделять открытый и секретный ключи RSA (хранить их раздельно)
- вводить (или вставлять из буфера) сообщение для шифрования
- шифровать сообщение открытым ключом
- отображать зашифрованное сообщение в виде текста или сохранять зашифрованное сообщение в файле для возможности его передачи

- прочитать зашифрованное сообщение из файла при его получении
- расшифровывать полученное сообщение секретным ключом

Дополнительное задание

Было перехвачено зашифрованное сообщение (см. свой вариант). В нем содержится два слова на русском языке. Также было установлено, что для шифрования использовался алгоритм RSA с открытым ключом (e, n) (см. свой вариант).

Также стало известно о наличии уязвимости в генераторе случайных простых чисел, используемом в момент создания криптосистемы. А именно, из-за ошибки в ПО генератора при задании параметров шифра были выбраны не слишком далекие друг от друга простые числа. Используя эту информацию, необходимо взломать криптографическую систему RSA, найти секретный ключ и расшифровать сообщение.

Вариант 1.

1482983448278432,1510598768350176,1531578985264449,1538623954900000,144903380198915
7,1524
559844999168,33554432,1462538217461399,1415708784197632,1524559844999168,1632591617
145743

(e, n) = (5, 6775897556066698567489175947518959875823270419332852746 16037)

Вариант 2.

1256216039,1238833224,1249243533,1256216039,1245766976,1231925248,32768,1238833224,12
808240 56,1291467969,1305751357,1280824056,1245766976

(e, n) = (3, 4739524150219688443214936538680274824139580400573028737 75749)

Вариант 3.

1531578985264449,1538623954900000,1449033801989157,1503656690178125,141570878419763
2,3355
4432,1462538217461399,1415708784197632,1482983448278432,1415708784197632,1538623954
900000

(e, n) = (5, 8803426790884604023148748777140782721399990806553236593 87641)

Вариант 4.

1517566463014207,1524559844999168,1510598768350176,1524559844999168,160320279790549
9,1428
964391886624,33554432,1482983448278432,1415708784197632,1462538217461399,1415708784
197632, 1482983448278432

(e, n) = (5, 715364797360727635638121396118392462599509766023373171934289)

Вариант 5.

1736164314502319024768,1781594144997124173696,1637563138325435542097,1804688569158
695124
992,1626909883459371532288,34359738368,1828039120816690000000,168077632890748088985
3,1816 331681783800622529,1626909883459371532288,1736164314502319024768

(e, n) = (7, 1384229008335999500787886773947516718775458375276829134498757)

Вариант 6.

1428964391886624,1449033801989157,1503656690178125,1449033801989157,152455984499916
8,1415
708784197632,33554432,1524559844999168,1415708784197632,1531578985264449,1531578985
264449, 1428964391886624,1449033801989157,1538623954900000

(e, n) = (5, 498179155408394962343042390059729729395602144208302883104531)

Вариант 7.

1291467969,1266723368,1231925248,1270238787,1231925248,32768,1280824056,1291467969,12
950290 00,1287913472,1280824056,1238833224

(e, n) = (3, 435761429006910881388493416772930350969992473078380051952483)

Вариант 8.

1702746956835476633159,1680776328907480889853,1758753502863159967744,1747427621807
167572
627,1986225654744065245487,34359738368,1804688569158695124992,183981118214633722273
1,1887 551606348838984375,1680776328907480889853,1724963292590968821961

(e, n) = (7, 459116378368265802904875563897970859301854490348136006233843)

Вариант 9.

1422324234276593,1545694825126451,1524559844999168,1415708784197632,150365669017812
5,3355 4432,1435629326171875,1510598768350176,1524559844999168,1415708784197632

(e, n) = (5, 516807652414358333957738743877510301181735407196024452343963)

Вариант 10.

1828039120816690000000,1804688569158695124992,1626909883459371532288,1648276130895
815505
024,1626909883459371532288,34359738368,1758753502863159967744,168077632890748088985
3,1887 551606348838984375,1828039120816690000000,1626909883459371532288

(e, n) = (7, 597928275110736450263008525687048655941889967814372211394299)

Вариант 11.

1758753502863159967744,1781594144997124173696,1816331681783800622529,1736164314502
319024
768,1648276130895815505024,1626909883459371532288,34359738368,175875350286315996774
4,1626 909883459371532288,1804688569158695124992,1816331681783800622529

(e, n) = (7, 745151659314139025940989123214176769383982415763432111965783)

Замечания

- Из зашифрованной строки перед ее использованием нужно удалить все символы переноса строки, добавленные pdf конвертором.
- Для нахождения квадратного корня из BigInt рекомендуется использовать итерационную формулу Герона.