Доп. дисциплины

Магистры весна 2021

Занятие 6

ВЫЧИСЛЕНИЯ В СИСТЕМЕ RSA

15.04.2021

Алгоритм RSA. *Шесть параметров шифра - его ключи – p,q,n,m,e,d.*

Пусть p и q – не менее, чем 256 битные простые числа:

$$n = p \cdot q, m = \varphi(\mathrm{n}) = (\mathrm{p}-1)\cdot(\mathrm{q}-1), \varphi - \text{функция Эйлера}.$$

Числа n и m - это количество элементов в кольце вычетов и в группе обратимых элементов

$$n = |Z_n|, m = \varphi(\mathrm{n}) = |Z_n^*|, \quad Z_n^* \cong Z_p^* \times Z_q^*.$$

И последние два параметра e и d: $\mathrm{НОД}(\mathrm{e},\mathrm{m}) = 1, \mathrm{ed} = 1(\mathrm{mod}\ \ \mathrm{m})$.

Модель шифра. *Открытый текст – это целое число* $0 < M < n$.

Шифр текст – это целое число $0 < C < n$.

Алгоритм зашифрования $E(\mathrm{M}) = \mathrm{M}^e(\mathrm{mod}\ \ \mathrm{n}) = C$.

Алгоритм расшифрования $\mathrm{D}(\mathrm{C}) = C^d(\mathrm{mod}\ \ \mathrm{n}) = \mathrm{M}^{ed}(\mathrm{mod}\ \ \mathrm{n})$.

Открытый ключ {e, n}. Секретный ключ {d, p, q, m}.

Дискуссионным является вопрос о шифровании открытых сообщений M, которые не взаимно просты с модулем n, т.е. когда M делится на p или на q.

Начнем с совершенно ручного, практически устного примера.

Пример 1. *Пусть p=3, q=5, n=15, m=8, e=3, d=3. Необходимо зашифровать и расшифровать все сообщения, не взаимно простые с числом n: K= [3,5,6,9,10,12].*

Решение. Нам нужно проверить, что

$$\text{для} \quad \forall k \in K \quad k^{ed} = k^{3\cdot 3} = k^9 = k(\mathrm{mod}\ \ \mathrm{n}) = \mathrm{k}(\mathrm{mod}\ \ 15).$$

Поскольку мы очень хорошо понимаем, что нам нужно делать и остается только выполнить рутинные, чисто механические вычисления, то мы имеем полное моральное право привлечь в помощь ЭВМ. А именно язык Julia 1.4.1 официальный сайт https://julialang.org/

```
julia> for k in [3,5,6,9,10,12]
       print(k^9%15,";")
       end
3;5;6;9;10;12.
```

Результат вычислений `3;5;6;9;10;12` совпадает с начальными данными `[3,5,6,9,10,12]`, т.е. расшифрование прошло корректно. Пример закончен.

Теперь реальная слабость алгоритма RSA – большое число ключей расшифрования. Проверим это на простых числах p = 29, q=53.

Пример 2. *Рассмотрим систему RSA*: *p=29, q=53, n=1537, m=1456*

И подберем подходящие ключи зашифрования e и расшифрования d.

Решение. Опять применим язык Julia и подключим пакет Nemo.

```
julia> using Nemo
[ Info: Precompiling Nemo [2edaba10-b0f1-5616-af89-8c11ac63239a]
Welcome to Nemo version 0.17.5
Nemo comes with absolutely no warranty whatsoever
julia> factor(ZZ(28*52))
1 * 13 * 7 * 2^4
```

Честно скажу, пока, не понял, как Nemo упорядочивает разложение на простые множители и зачем вначале пишет 1.

Главное, что 1456=2^4*7*13 и самое маленькое число, с ним взаимно простое, – это e=3. Теперь найдем d:

```
julia> F=ResidueRing(ZZ, 1456)
Integers modulo 1456
julia> F(3)^(-1)
971
```

Итак, d=971,

Поскольку НОД(p-1,q-1)=НОД(28,52)=4, то разных ключей расшифрования должно быть 4.

Согласно теории, решать уравнения нужно по модулю 1456/4=364.

```
julia> F1=ResidueRing(ZZ, 364)
Integers modulo 364
julia> F1(3)^(-1)
243
```

Итак,

$$d_0 = 243 \Rightarrow K = \{243, 243+364, 243+2*364, 243+3*364\} = \{243, 607, 971, 1335\}$$

Проверим, что все эти ключи расшифровывают правильно.

Возьмем наугад какое-нибудь число, например, “самое случайное” 1234,

Зашифруем его на ключе e=3

```
julia> 1234^3%1537
36
```

Наш шифр текст – это C = 36,

А теперь расшифруем его 4-мя разными ключами

```
julia> using Nemo
Welcome to Nemo version 0.17.5
Nemo comes with absolutely no warranty whatsoever
julia> F2=ResidueRing(ZZ, 1537)
Integers modulo 1537
julia> for i in [243,607,971,1335]
       print(F2(36)^i,";")
       end
1234;1234;1234;1234.
```

Пример закончен.

Пример 3.

Еще одна слабость RSA – это выбор близких простых чисел. В этом случае можно подобрать нужное разложение.

Тут, даже в учебных целях очень маленькими числами не обойтись.

Но и 150-значные числа ничему не учат, их длина просто лишает нас всякой надежды на успех.

Ограничимся 5-значными числами. И не устные и до компьютерных очень далеко.

Опять же руками ничего считать не будем. А для поиска простых чисел подключим пакет Primes.

```
julia> using Primes
julia> p=prime(1959)
16987
julia> q=prime(1975)
17137
julia> n=p*q
291106219
julia> m=(p-1)*(q-1)
291072096
julia> factor(m)
2^5 ? 3^3 ? 7 ? 17 ? 19 ? 149
```

Итак

$$m = 291072096 = 2^5*3^3*7*17*19*149.$$

Две новости – хорошая и плохая. Хорошая, что функция factor в пакете Primes ведет себя предсказуемо – простые числа выписываются по порядку.

Плохая та, что, если вы подключите пакет Nemo, где тоже есть функция factor – обе функции и в Nemo и в Primes не будут работать.

И это правильно, если в системе две разные функции с одним и тем же названием, то откуда компьютеру знать о какой из них идет речь.

“Детская болезнь левизны”, как говорил В.И. Ульянов – Ленин, которому на днях, 22.04.2020 исполнилось 150 лет. Это говорит о том, что Julia очень юный язык программирования.

```
julia> using Primes
julia> p=prime(1959)
16987
julia> q=prime(1975)
17137
julia> n=p*q
291106219
julia> m=(p-1)*(q-1)
291072096
julia> factor(m)
2^5 ? 3^3 ? 7 ? 17 ? 19 ? 149
```

Итак, m = `2^5*3^3*7*17*19*149`

Самое маленькое подходящее число шифрование это e=5.

А вот, чтобы найти расшифровывающее d придется перезагрузить Julia, чтобы избавиться от Primes и загрузить ее конкурентку подпрограмму Nemo.

```
julia> using Nemo
Welcome to Nemo version 0.17.5
Nemo comes with absolutely no warranty whatsoever
julia> F=ResidueRing(ZZ, 291072096)
Integers modulo 291072096
julia> F(5)^(-1)
232857677
```

Значит d = `232857677.`

Проверим, что d действительно ключ расшифрования. Имеем

```
julia> 5*232857677 % 291072096 = 1
```

Взлом шифра.

Из неких источников highly likely известно, что числа p и q близки друг к другу и у нас есть примерно 1000 попыток подобрать разложение n = p*q.

а) Начальные данные n = `291106219`

значит

```
julia> n = 291106219
291106219
julia> sqrt(n) #извлекаем квадратный корень
17061.835159208404
```

Самое простое решение - это начиная с числа 17063 (все простые числа нечетные), подниматься вверх, разыскивая наименьший простой множитель, как результат деления на наибольший.

Решение наше абсолютно не изящное, одноразовое. Из серии, что первое пришло в голову.

```
julia> for i in 17063:2:18000
       println(291106219/i)
       end
17060.670397937058
17058.67090536185
17056.671881408565
17054.673325912474
17052.675238708922
17050.677619633338
17048.68046852123
17046.683785208174
17044.687569529833
17042.691821321936
17040.6965404203
17038.701726660813
17036.70737987944
17034.713499912225
17032.720086595284
17030.727139764815
17028.734659257094
17026.742644908463
17024.751096555356
17022.760014034266
17020.769397181783
17018.77924583455
17016.78955982931
17014.800339002864
17012.8115831921
17010.823292233974
17008.835465965527
17006.84810422387
17004.861206846195
17002.874773669762
17000.888804531915
16998.903299270074
16996.918257721725
16994.933679724443
16992.949565115872
16990.96591373373
16988.982725415815
16987.0
```

Но мы нашли решение – это p=16987. Дальше все очевидно:

```
julia> n = 291106219
```

```
291106219
julia> p=16987
16987
julia> q=n/p
17137.0
```

Зная p и q мы легко найдем m и d = e^(-1) (mod m)

```
julia> m = Int((p-1)*(q-1))
291072096
```

Пример закончен.

ЛИТЕРАТУРА

1. Рябко Б.Я, Фионов А.Н. Криптографические методы защиты информации, 2-е изд. [Электронный ресурс]. – М.: Горячая линия-Телеком, 2017. - URL: https://e.lanbook.com/reader/book/111097/
2. Глухов М.М., Круглов И.А., Пичкур А.Б., Черемушкин А.В. Введение в теоретико-числовые методы криптографии. [Электронный ресурс]. - СПб.: Лань, 2011. - URL: https://e.lanbook.com/reader/book/68466
3. ГОСТ 34.12-2018. Информационная технология. Криптографическая защита информации. Блочные шифры.
4. ГОСТ 34.13-2018. Информационная технология. Криптографическая защита информации. Режимы работы блочных шифров.