Сириус
IT-Колледж

Лекция 2. Числовые алгоритмы

Арифметика по модулю: сложение, умножение, возведение в степень, алгоритм Евклида, расширенный алгоритм Евклида, деление. Проверка чисел на простоту. Генерация случайных простых чисел. Криптография: схемы с закрытым ключом, RSA.

Сириус
IT-Колледж

Арифметика по модулю: сложение, умножение, возведение в степень, алгоритм Евклида, расширенный алгоритм Евклида, деление

Необходимость использования

1. Ограничения на объем типов данных для Python:

   Для `int`: `2**32-1`

   Для `float : 2**64-1`

   Для `long`: неограниченно

2. Время выполнения операций на больших числах
3. Использование встроенных функций ограниченно типами данных и особенностями реализации

Основные идеи

$$
\begin{array}{r}
\color{red}{1} \\
18273 \\
+\ \ 7602 \\
\hline
25875
\end{array}
$$

Основные идеи

Использование массивов, состоящих из чисел “коротких” (играют роль цифр) и число `st`, означающее позиции с которой начинаются отличные от нуля цифры.

Зададим максимальную длину длинного числа `MAXLEN`

Зададим основание “системы счисления” `SYS`

Сложение больших чисел

```
def add(a, st_a, b, st_b):
    res = [0 for i in range(MAXLEN)]
    flag = 0
    mxop = a if st_a > st_b else b
    mnop = a if st_a <= st_b else b
    st_min = min(st_a, st_b)
    st_max = max(st_a, st_b)
    st = st_min

    for i in range(MAXLEN-1, st_max - 1, -1):
        res[i] = mxop[i] + mnop[i] + flag
        flag = res[i] // SYS
        res[i] %= SYS

    for i in range(st_max - 1, st_min - 1, -1):
        res[i] = mnop[i] + flag
        flag = res[i] // SYS
        res[i] %= SYS

    if flag:
        res[st-1] = flag

    return res, st
```

Сириус
IT-Колледж

Умножение больших чисел на малые

```
def mul(a, st, b, offs):
    res = [0 for i in range(MAXLEN)]
    flag = 0

    for i in range(MAXLEN - offs, MAXLEN):
        print(i)
        res[i] = 0

    for i in range(MAXLEN-1, st-1, -1):
        res[i-offs] = a[i] * b + flag
        flag = res[i-offs] // SYS
        res[i-offs] %= SYS

    if flag:
        res[st-offs-1] = flag
    st -= offs

    return res, st
```

Сириус
IT-Колледж

Умножение больших чисел

```
def mul_long(a, st_a, b, st_b):
    res = [0 for i in range(MAXLEN)]
    st = MAXLEN-1
    for i in range(MAXLEN-1, st_a-1, -1):
        temp, st_temp = mul(b, st_b, a[i], MAXLEN-1-i)
        res, st = add(res, st, temp, st_temp)

    return res, st
```

Сириус
IT-Колледж

Возведение в степень

Наивный алгоритм имеет сложность *O(N)*, где *N* - степень.

Помня о том, что при возведении числа в степени в какую-либо степень показатели степеней перемножаются. Отсюда следует, что возможно сократить четные степени. Например:

3^4 - 3 операции умножения

(3^2)^2 - 2 операции умножения

Возведение в степень

```
def power(a, n):
    k = n
    b = 1
    c = a
    while k:
        if not (k % 2):
            k /= 2
            c *= c
        else:
            k -= 1
            b *= c
    return b

#O(log N), где N - степень
```

Сириус
IT-Колледж

Если два целых числа a и b при делении на m дают одинаковые остатки, то они называются сравнимыми (или равноостаточными) по модулю числа m.

$$a \equiv b \pmod{m}$$

Сравнение по модулю

Определение сравнимости чисел a и b по модулю m равносильно любому из следующих утверждений:

- разность чисел a и b делится на m без остатка;
- число a может быть представлено в виде a=b+k * m, где k — некоторое целое число.

Сириус IT-Колледж

Операции по модулю

$(A + B) \bmod C = (A \bmod C + B \bmod C) \bmod C$

$(A * B) \bmod C = (A \bmod C * B \bmod C) \bmod C$

$A^B \bmod C = ((A \bmod C)^B) \bmod C$

Пример сложения по модулю

A=14, B=17, C=5
Давайте проверим: (A + B) mod C = (A mod C + B mod C) mod C
ЛЧ = левая часть равенства
ПЧ = правая часть равенства
ЛЧ = (A + B) mod C
ЛЧ = (14 + 17) mod 5
ЛЧ = 31 mod 5
ЛЧ = 1
ПЧ = (A mod C + B mod C) mod C
ПЧ = (14 mod 5 + 17 mod 5) mod 5
ПЧ = (4 + 2) mod 5
ПЧ = 1
ЛЧ = ПЧ = 1

Обратное число по модулю

Числом, обратным к числу A, называется такое число 1/A, что A * 1/A = 1 (то есть, к примеру, для числа 5 обратным будет 1/5).

У каждого вещественного числа, кроме 0, есть обратное.

Умножение на число, обратное A, эквивалентно делению на A (то есть, к примеру, 10/5 — это то же, что 10* 1/5).

Число, обратное A (mod C), обозначается A^-1.

(A * A^-1) ≡ 1 (mod C) , или, что то же самое, (A * A^-1) mod C = 1.

Только у чисел, взаимно простых с C (то есть у тех, у которых нет с C общих простых делителей), есть обратные (mod C)

Нахождение обратного числа по модулю

Вычисляем A * B mod C для всех B от 0 до C-1.

Обратным числом для A mod C будет являться такое B, для которого A * B mod C = 1

Обратите внимание, что B mod C может принимать значения от 0 до C-1, поэтому нет смысла проверять числа, большие чем B.

НОД и НОК

Любое число можно представить в виде одномерного массива, содержащего простые числа, и еще одного одномерного массива для непосредственного представления числа, в каждой ячейке которого содержится степень соответствующего простого числа.

Например, массив простых чисел `[2, 3, 5, 7, 11, 13]`, тогда массив с представление числа `2600` будет выглядеть как `[3, 0, 2, 0, 0, 1]` из которого можно получить `(2 ** 3) * (5 ** 2) * (13 ** 1) = 2600`.

НОД и НОК

Простая реализация умножения: достаточно сложить соответствующие элементы массивов.

Для нахождения НОД достаточно выбрать минимум из степеней.

Для нахождения НОК достаточно выбрать максимум из степеней.

Алгоритм Евклида

Даны два целых положительных числа m и n. Требуется найти их наибольший общий делитель, т. е. наибольшее целое положительное число, которое нацело делит оба числа m и n.

1. [Нахождение остатка.] Разделим m на n, и пусть остаток от деления будет равен r (где $0 \le r < n$).
2. [Сравнение с нулем.] Если $r = 0$, то выполнение алгоритма прекращается; n — искомое значение.
3. [Замещение.] Присвоить $m \leftarrow n$, $n \leftarrow r$ и вернуться к шагу E1.

Расширенный алгоритм Евклида

a*x + b*y = НОД(a,b)

Пусть (x1, y1) - решение для задачи новой пары (b%a, a). Тогда:

$$\begin{cases} x = y_1 - \left\lfloor \frac{b}{a} \right\rfloor \cdot x_1, \\ y = x_1. \end{cases}$$

Проверка чисел на простоту

Простым числом называется число, большее 1, которое делится нацело только на себя и 1 (имеет два натуральных делителя).

Проверка числа на простоту

Необходимо узнать, существуют ли такие натуральные числа x, y (1 < x, y < N), что x*y = N.

Условимся, что x<=y, тогда можно сказать, что x меньше либо равен квадратному корню из числа N.

Проверка числа на простоту

```
def isPrime(n):
    if n == 2:
        return True
    j = int(n ** 0.5) + 1
    for i in range(2, j + 1):
        if n % i == 0:
            return False
    return True
```

Сириус
IT-Колледж

Решето Эратосфена

Решето Эратосфена — алгоритм нахождения всех простых чисел, не превышающих некоторое натуральное число n.

1. Выписать подряд все целые числа от двух до n (2, 3, 4, …, n).
2. Пусть переменная p изначально равна двум — первому простому числу.
3. Зачеркнуть в списке числа от 2p до n, считая шагами по p (это будут числа, кратные p: 2p, 3p, 4p, …).
4. Найти первое незачёркнутое число в списке, большее чем p, и присвоить значению переменной p это число.
5. Повторять шаги 3 и 4, пока возможно.

Сириус
IT-Колледж

Генерация случайных простых чисел

Генерация случайных простых чисел

1. Строим решето Эратосфена до k, где k это некая константа - например 10^3. Выбираем стартовое простое число s - либо принимаем аргументом, либо из построенного решета. Переходим к шагу 2.
2. Имеем простое число s - если $s>N$, то результат алгоритма это $p:p=s$, иначе мы хотим найти простое число $n:n>s$ - переходим к шагу 3.
3. Выбираем рандомно чётное число $r : s <= r <= 2 * (2 * s + 1)$ и запишем кандидат на простоту $n=s*r+1$. Переходим к шагу 4.
4. Проверяем n на делимость с простыми числами низкого порядка, полученными на шаге 1 - если число делится на одно из них, то оно составное и возвращаемся к выбору нового кандидата n, то есть шагу 2. Иначе число может быть простым, поэтому переходим к шагу 5.

Сириус IT-Колледж

Генерация случайных простых чисел

5. Выбираем рандомно число $a: 1<a<n$ и проверяем для нашего кандидата на простоту n исполнимость следующих двух условий $a^{n-1} \equiv 1 \pmod{n}$ и НОД$(a^r - 1, n) == 1$.

Если оба исполняются, то по критерию Поклингтона число n простое - заменяем $s := n$ и переходим к следующей итерации по поиску большего числа, то есть шагу 2.

Иначе если не исполняется первое условие - $a^{n-1} \equiv 1 \pmod{n}$, то по малой теореме Ферма число n не является простым, поэтому переходим к выбору нового кандидата, то есть шагу 3.

Иначе если не исполняется вторая часть, то анализ немного сложнее - мы нашли делитель $d: 1<d<=n$ для кандидата n, поскольку $\gcd(a^r-1,n)==d$. Предположим, что $d \neq n$, что подразумевает не примитивный делитель, а значит n не простое - переходим к повторному выбору, то есть шагу 3.

Остаётся случай, когда $d==n$, что означает $a^r \equiv 1 \pmod{n}$, а решений этого выражения существует не больше r. Одно из решений это $a==1$, поэтому на интервале $1<a<n$ существует не более $r-1$ решений, следовательно при действительно простом n мы найдём (с вероятностью $1-O(s^{-1})$) такое a, которое бы удовлетворяло критерию Поклингтона, поэтому переходим к шагу 5 для повтора выбора a.

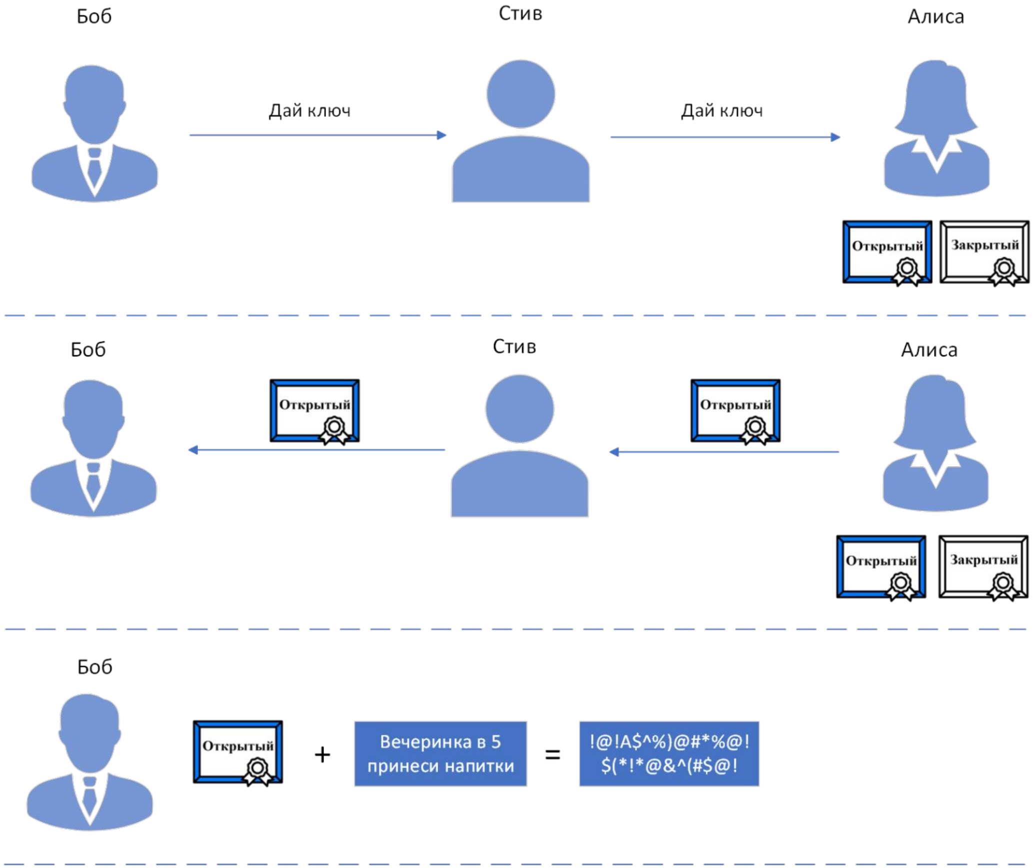
Криптография: схемы с закрытым ключом, RSA.

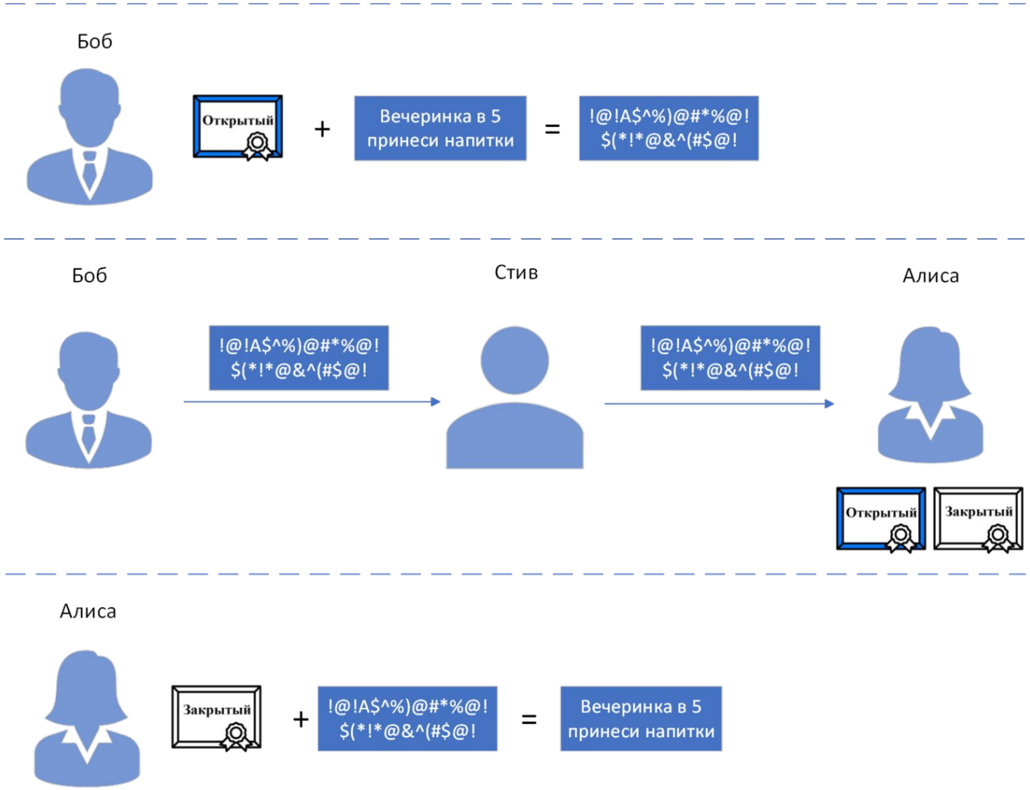
RSA

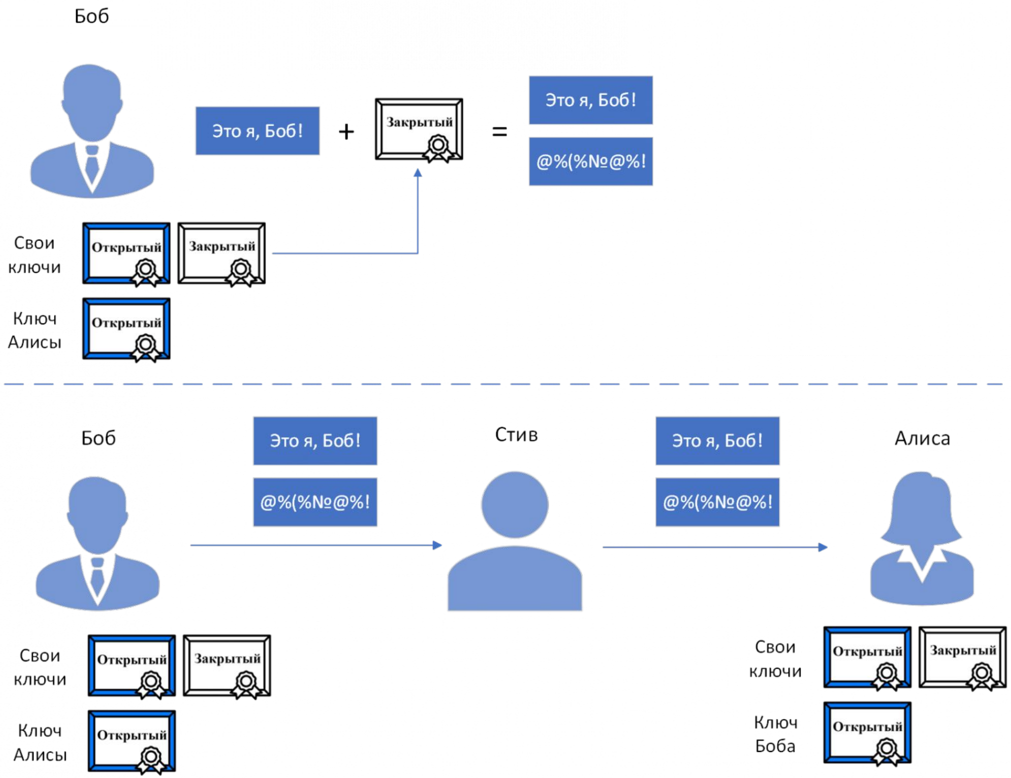
RSA (аббревиатура от фамилий Rivest, Shamir и Adleman) — криптографический алгоритм с открытым ключом, основывающийся на вычислительной сложности задачи факторизации больших полупростых чисел.

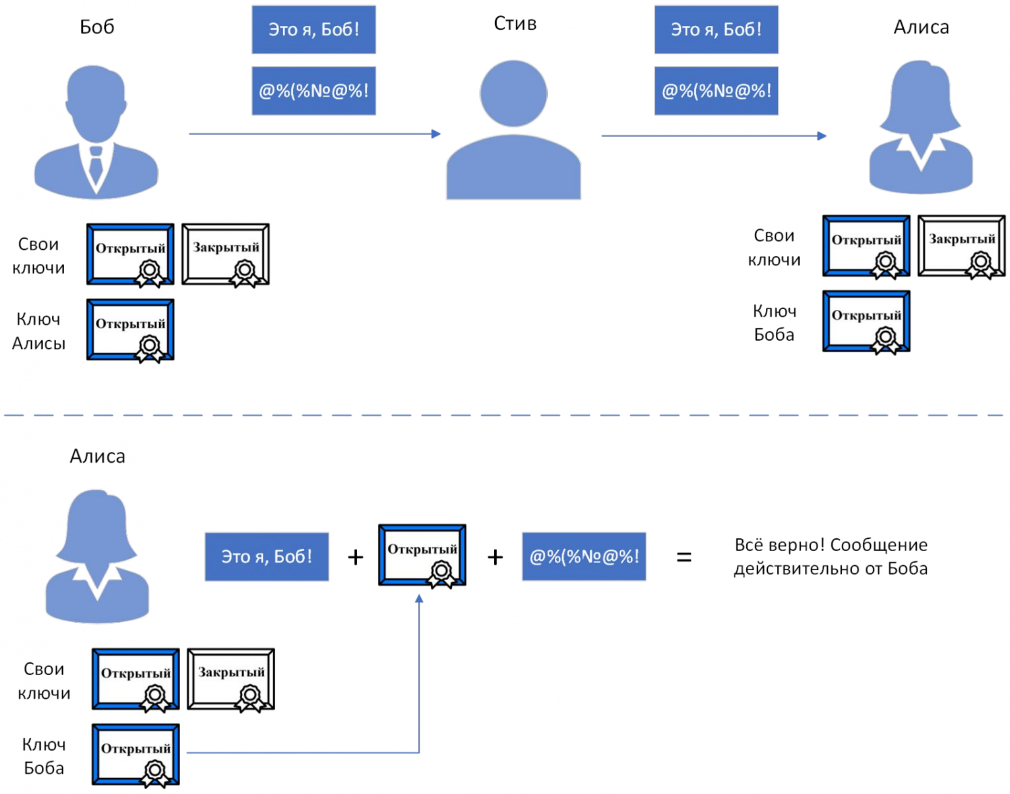
Например, 592939 * 592967 = 351593260013. Но как имея только число 351593260013 узнать числа 592939 и 592967? Это называется «сложность задачи факторизации произведения двух больших простых чисел», т.е. в одну сторону просто, а в обратную невероятно сложно.

Используется при обмене данными и в качестве цифровой подписи. Является базовой частью протокола HTTPS.


Боб
Стив
Алиса
Дай ключ
Дай ключ
Открытый
Закрытый
Боб
Стив
Алиса
Открытый
Открытый
Открытый
Закрытый
Боб
Открытый
+
Вечеринка в 5 принеси напитки
=
!@!A$^%)@#*%@!
$(*!*@&^(#$@!


Боб
Открытый
+
Вечеринка в 5 принеси напитки
=
!@!A$^%)@#*%@! $(*!*@&^(#$@!
Боб
!@!A$^%)@#*%@! $(*!*@&^(#$@!
Стив
!@!A$^%)@#*%@! $(*!*@&^(#$@!
Алиса
Открытый
Закрытый
Алиса
Закрытый
+
!@!A$^%)@#*%@! $(*!*@&^(#$@!
=
Вечеринка в 5 принеси напитки


Боб
Это я, Боб!
+
Закрытый
=
Это я, Боб!
@%(%№@%!
Свои ключи
Открытый
Закрытый
Ключ Алисы
Открытый
Боб
Это я, Боб!
@%(%№@%!
Стив
Это я, Боб!
@%(%№@%!
Алиса
Свои ключи
Открытый
Закрытый
Ключ Алисы
Открытый
Свои ключи
Открытый
Закрытый
Ключ Боба
Открытый


Боб
Это я, Боб!
@%(%№@%!
Стив
Это я, Боб!
@%(%№@%!
Алиса
Свои ключи
Открытый
Закрытый
Ключ Алисы
Открытый
Свои ключи
Открытый
Закрытый
Ключ Боба
Открытый
Алиса
Это я, Боб!
+
Открытый
+
@%(%№@%!
=
Всё верно! Сообщение действительно от Боба
Свои ключи
Открытый
Закрытый
Ключ Боба
Открытый

Процедура создания ключей

1. Выбираем два случайных простых числа p и q
2. Вычисляем их произведение: N = p * q
3. Вычисляем функцию Эйлера: φ(N) = (p-1) * (q-1)
4. Выбираем число e (обычно простое, но необязательно), которое меньше φ(N) и является взаимно простым с φ(N) (не имеющих общих делителей друг с другом, кроме 1).
5. Ищем число d, обратное числу e по модулю φ(N) .Т.е. остаток от деления (d*e) и φ(N) должен быть равен 1. Найти его можно через расширенный алгоритм Евклида.

e и n – открытый ключ

d и n – закрытый ключ

Пример генерации ключей

Пусть p = 19, q = 41

$$N = p \cdot q = 779$$

$$\varphi(N) = (p - 1) \cdot (q - 1) = 720$$

$$e = 691$$

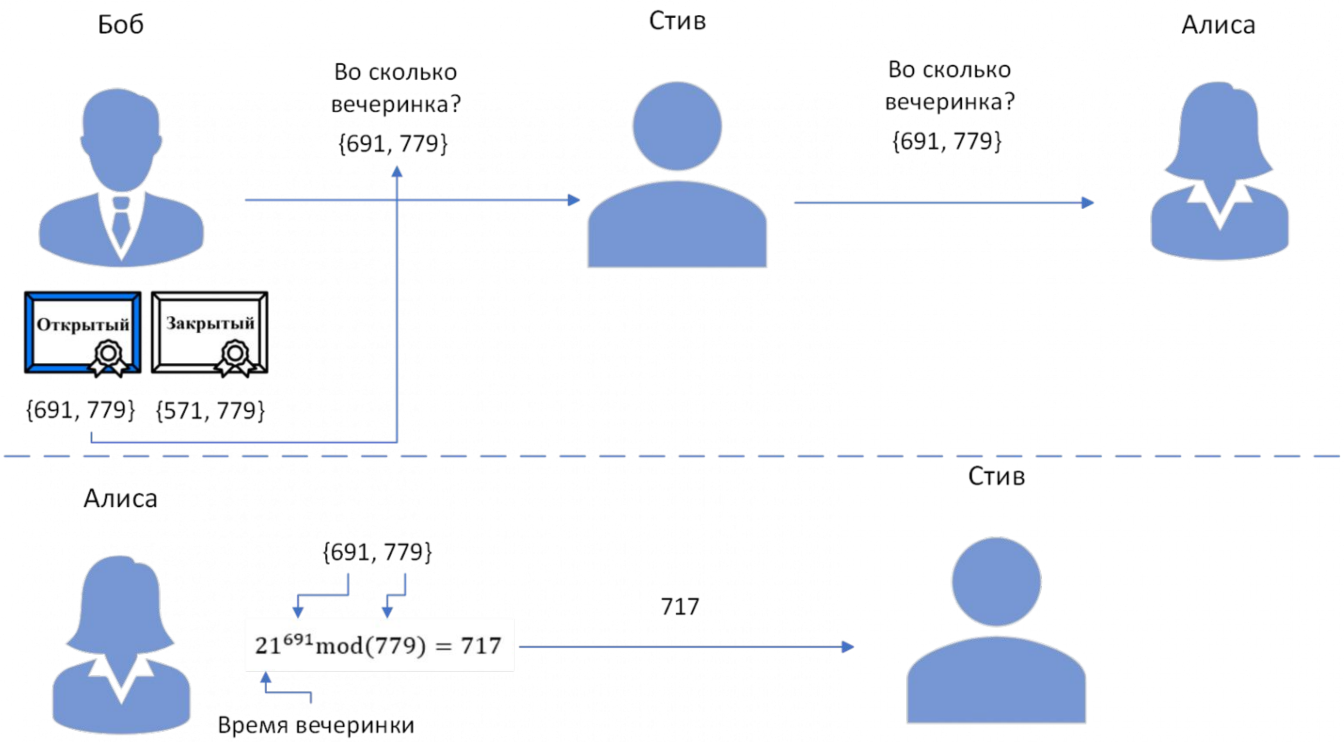
$$d = 571$$

Получается:

{691, 779} – открытый ключ

{571, 779} – закрытый ключ

Процесс шифрования



Процесс шифрования