Математические основы криптографии

1. Элементы Теории чисел.

Теория ч. – изучение свойств целых чисел

Множество ц.ч.: $\mathbb{Z}$ ={…, -2,-1,0,1,2,… }

Множество **натуральных** чисел $\mathbb{N}$={1, 2, 3, ...}

Определение 1 Целое число **a** есть **кратное** числа **b, если** **a=b*m** для некоторого целого числа **m**; **(числа ≠ 0)**

Обозначается b|a; н.п. **9|27,** т.к. **27 = 9*3;**

число b называется делителем числа a

Задача. Найти положительные делители чисел **6, 12, 25**

Теорема 1 (алгоритм деления) Для полож ц.ч. a и b существуют единственные целые полож. ч. q и r, где $0 \le r < b$ такие что $a = bq + r$; $a < b$, r – остаток, q – частное

Если $a < b$, то $q=0$.

Н.П. a =4, b=7, тогда q =0, r=4, т.к. 4=7*0+4;

a =15, b=7, тогда q =2, r=1, т.к. 15=7*2+1;

Определение 2 Положит ц.ч. **d** называется **общим делителем** чисел a и b, если d|a и d|b.

Определение 3 Положит ц.ч. **d** называется **наибольшим общим делителем** чисел a и b, **НОД(a,b)**,

если d|a и d|b, и если из c|a и c|b следует **c|d.**

Задача. Найти **НОД(a, b)**

для пар: 54,24; 6,15; 34,13

Теорема 2 Если a = bq + c, то НОД(a, b) = НОД(b, c);

Н.П. 16=6*2+4; a = 16, b=6, q =2, c=4.

НОД(a, b) = НОД(16, 6) = 2; НОД(b, c)= НОД(6, 4)=2

Определение 4 Если НОД(a, b) = 1, то числа a, b называются взаимно простыми.

Простые и взаимно простые числа

Основная теорема арифметики. Всякое натуральное число *N*, кроме 1, можно представить как произведение **простых** сомножителей: $N = p_1 * p_2 * p_3 * ... * p_n$, $n > 1$.

Пример. 1554985071 = 3 × 3 × 4463 × 38713;

39 616 304 = 2 × 13 × 7 × 2 × 23 × 13 × 2 × 13 × 2 × 7 = 2 × 2 × 2 × 2 × 7 × 7 × 13 × 13 × 13 × 23.

Определение 4 Натуральное число p называется *простым*, если $p > 1$ и не имеет положительных делителей, отличных от 1 и p.

Пример. *Простые:* 2, 73, 2521, 2365347734339

Существует около 10^{151} простых чисел длиной от 1 до 512 бит

Определение 5. *Взаимно простые числа* **a, b** не имеют общих множителей, кроме 1, НОД (a, b) =1

Проблема. **При разрядности 1024 и более бит нахождение пары взаимно простых чисел, удовлетворяющих пределенному условию, может занять сотни лет**.

Решето Эратосфена

- Первый алгоритм нахождения простых чисел, не превышающих **n**, был придуман Эратосфеном во 2 в. до н. э. и известен сейчас как «**решето Эратосфена**».

  Его суть в последовательном исключении из списка целых чисел от **1** до **n** чисел (или из сокращенного диапазона, например, от **m** до n, $\mathbf{1<m\leq n}$), кратных 2, 3, 5 и другим простым числам, уже найденным «решетом».

Решето Эратосфена

- Для нахождения всех простых чисел не больше заданного числа n в соответствии с «решетом Эратосфена» нужно выполнить следующие шаги:
- 1. Выписать подряд все целые числа от **2** (либо от **m**) до **n** (2, 3, 4, …, n). Пусть некоторая переменная (положим **s**) изначально равна 2 – первому простому числу.
- 2. Удалить из списка числа от **2s** до **n**, считая шагами по **s** (это будут числа кратные **s**: **2s, 3s, 4s**, …).
- 3. Найти первое из оставшихся чисел в списке, большее чем **s**, и присвоить значению переменной **s** это число. 4.
- Повторять шаги 2 и 3, пока возможно.

Решето Эратосфена

Пример. Примем **n** =15.

- Шаг 1. Выпишем числа от 2 до 15: 2, 3, 4, 5, 6, 7, 8, 9, 10, 10, 11, 12, 13, 14,15.
- Шаг 2. Удалим из списка числа с учетом **s**=2: 2, 3, 5, 7, 9, 11, 13,15. В этом списке первое число, большее, чем **s**=2, это 3. Текущему **s** присваивается новое значение: s =3.
- Шаг 3. Удалим из списка числа с учетом **s**=3: 2, 3, 5, 7, 11, 13, 15. В этом списке первое число, большее, чем **s**=3, это 5. Текущему **s** присваивается новое значение: **s** =5.
- Шаг 4. Удалим из списка числа с учетом **s**=5: 2, 3, 5, 7, 13. В этом списке первое число, большее, чем **s**=5, это 7. Однако, в этом списке уже нет чисел, кратных текущему значению **s**, т.е. 7.

Таким образом, **числа 2, 3, 5, 7, 13 являются простыми**.

Алгоритм Евклида

Даны два числа – a и b; $a > 0$, $b > 0$, считаем, что $a > b$.
Находим ряд равенств:

$$
\begin{array}{ll}
a = b\,q_1 + r_1, & 0 \le r_1 < b, \\
b = r_1\,q_2 + r_2, & 0 \le r_2 < r_1, \\
r_1 = r_2\,q_3 + r_3, & 0 \le r_3 < r_2, \\
r_2 = r_3\,q_4 + r_4 & 0 \le r_4 < r_3, \\
\cdots\cdots\cdots & \cdots\cdots \\
r_{n-3} = r_{n-2}\,q_{n-1} + r_{n-1}, & 0 \le r_{n-1} < r_{n-2}, \\
r_{n-2} = r_{n-1}\,q_n + r_n, & 0 \le r_n < r_{n-1}, \\
r_{n-1} = r_n\,q_{n+1}, & r_{n+1} = 0,
\end{array}
$$

заканчивается, когда получаем некоторое $r_{n+1} = 0$.
Тогда r_n – наибольший общий делитель чисел a и b.

<u>Задача</u>. Пусть $a = 525$, $b = 231$. Найти НОД.

$525 = 231 \cdot 2 + 63;$

$231 = 63 \cdot 3 + 42;$

$63 = 42 \cdot 1 + 21;$

$42 = 21 \cdot 2.$

Получаем последний положительный остаток $r_3 = 21$.
Таким образом, НОД (525, 231) = 21.

<u>Задача</u>. Пусть $a = 1234$, $b = 54$. Найти НОД.

$1234 = 54 \cdot 22 + 46;$

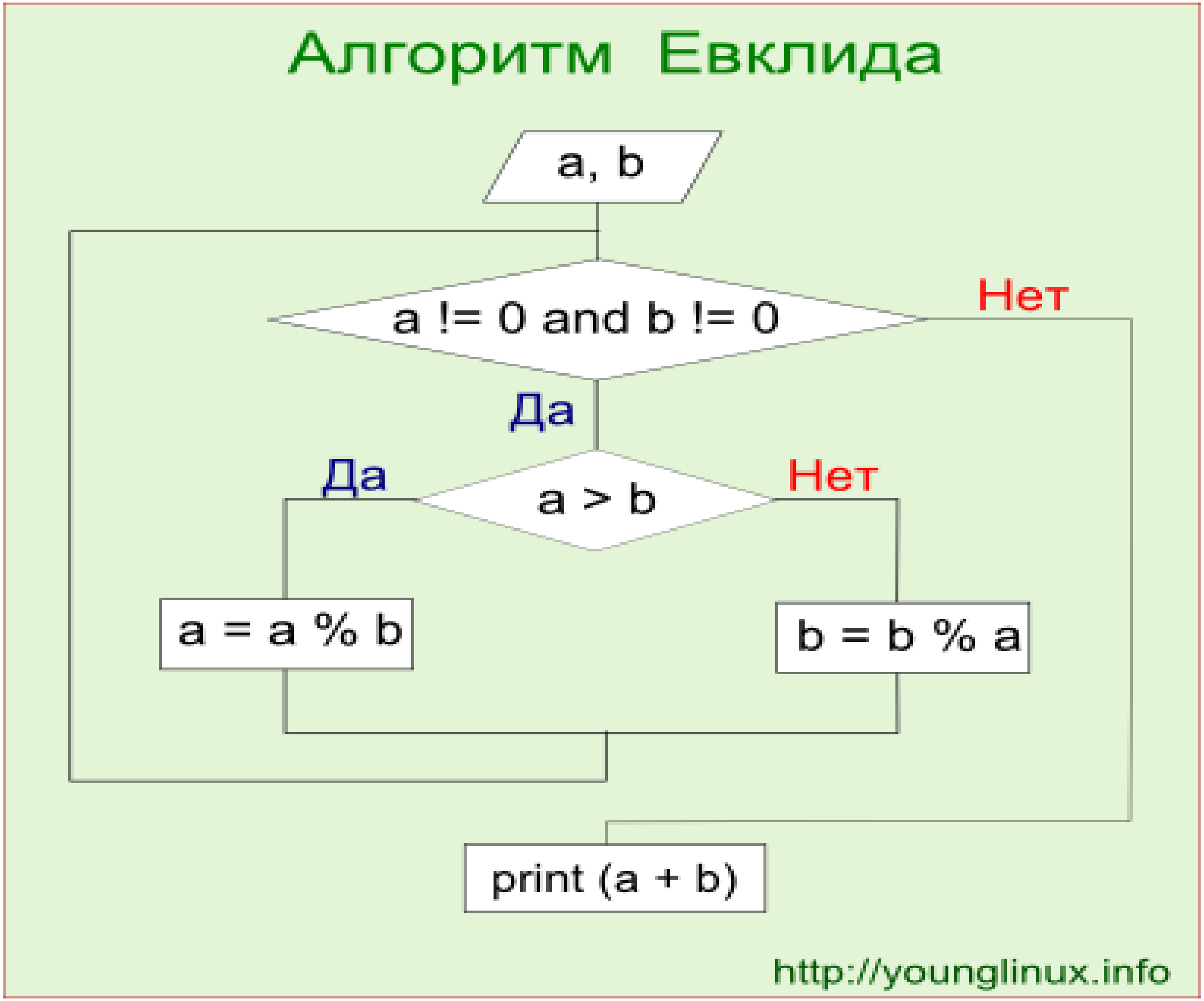
$54 = 46 \cdot 1 + 8;$

$46 = 8 \cdot 5 + 6;$

$8 = 6 \cdot 1 + 2;$

$6 = 2 \cdot 3.$

Последний ненулевой положит остаток равен 2,
поэтому НОД (1234, 54) = 2


Алгоритм Евклида
a, b
a != 0 and b != 0
Нет
Да
a > b
Да
Нет
a = a % b
b = b % a
print (a + b)
http://younglinux.info

Количество натуральных чисел, меньших некоторого числа n и взаимно простых с ним можно подсчитать на основе известной **функции Эйлера** (по имени швейцарского математика Леонарда Эйлера, 1707-1783), иногда называемой «**фи-функцией**», обозначается **φ(n)**.

Пример. Для числа 24 (n =24) существует 8 взаимно простых с ним чисел (1, 5, 7, 11, 13, 17, 19, 23), поэтому φ(24) = 8.

Если n – простое число, то φ(n) = n-1.

Другие примеры:

φ(1) =1; φ(5) = 4; φ(9) = 6;
φ(2) = 1; φ(6) = 2; φ(10) = 4;
φ(3) = 2; φ(7) = 6; φ(11) = 10;
φ(4) = 2; φ(8) = 4; φ(12) = 4.

Любое положительное целое число р может быть выражено с помощью положительных целых чисел, не превосходящих и взаимно простых с каждым делителем числа р.

Пример. Число р= 6 = 2 *3 имеет четыре делителя: 1, 2, 3 и 6:

$\varphi(1) + \varphi(2) + \varphi(3) + \varphi(6) = 1 + 1 + 2 + 2 = 6$.

Если $n = p * q$, то $\varphi(n) = (p-1)*(q-1)$.

Если числа **p** и **q** – взаимно простые, то

$\varphi(p*q) = \varphi(p) * \varphi(q)$.

Пример. Пусть p = 8 и q = 15. Тогда **φ** (8) = 4, поскольку только 1, 3, 5 и 7 – положительные целые числа, которые меньше 8 и взаимно простые с 8.
Также **φ** (15) = 8, поскольку только 1, 2, 4, 7, 8, 11, 13 и 14 – положительные целые числа, которые меньше 15 и взаимно простые с 15.

Следовательно,

φ(120) = φ (8) * φ (15) = 32.

Если *p* и *q* – очень большие простые числа и известен результат их перемножения (число *n*), то обратная задача – найти *p* и *q* по известному *n* (<u>задача факторизации</u>) даже для современных вычислительных средств представляется практически неразрешимой.
<u>Эта особенность используется в некоторых алгоритмах асимметричной криптографии</u>.

Большие числа

Число	Физический эквивалент
1 из 2^7	Вероятность погибнуть в автокатастрофе
2^{30}, лет	Время до превращения Солнца в сверхновую звезду
2^{34}, лет	Возраст Вселенной
2^{170}	Число атомов планеты

2. Модулярная арифметика

Понятие «модулярная арифметика» ввел немецкий ученый Гаусс. В этой арифметике мы интересуемся остатком от деления числа **a** на число **n**.

Если таким остатком является число **b**, то можно записать:

$a \equiv b \pmod{n}$ или **$a \equiv b \bmod n$**.

МА – арифметика вычетов

числа 23 и 11 равны по модулю 12:

$23 = 11 \bmod 12$

$a \equiv b \bmod n$, если **$a=b+kn$** при целом **k ()**;

Опред.1.В операции **$a \equiv b \bmod n$** **b** называют ***вычетом*** по модулю **n**;

a mod n обозначает ***вычет*** от **a**;

Опред.2.Множество целых чисел от **0** до **n-1** образует ***полную систему вычетов*** по модулю **n**

Правила модулярной арифметики

- $(a + b) \bmod n = ((a \bmod n) + (b \bmod n)) \bmod n$
- $(a - b) \bmod n = ((a \bmod n) - (b \bmod n)) \bmod n$
- $(a * b) \bmod n = ((a \bmod n) * (b \bmod n)) \bmod n$
- $(a * (b+c)) \bmod n = (((a * b) \bmod n) + ((a * c) \bmod n)) \bmod n$
- $a+0=a$, $\bar{a}+a=1$, $a+1=\bar{a}$

Пример. Вычислить

$\mathbf{a}^8 \bmod \mathbf{n}$

Простое решение: $((a^2 \bmod n)^2 \bmod n)^2) \bmod n$

Правила модулярной арифметики

Пример. $\mathbf{a^x}$ **mod n** ; **x** не является степенью **2**:

x=25: a^{25} mod n = (a* a^{24}) mod n = ((((a^2 * a)^2)^2)^2 * *a) mod n = (((((((a^2 mod n)* a) mod n)^2 mod n)^2 mod n)^2 mod n) * a) mod n

Метод – *аддитивная цепочка*

Обратные числа в модулярной арифметике

Традиционно: обратное от **4** равно**1/4**, т.к **4*1/4=1**

В модулярной арифметике:

$a * x \equiv 1 \bmod n$ эквивалентно поиску таких значений **x** и **к**, что **$a * x = n*к +1$**

Общая задача: найти такое **x,** что

$1=(a*x) \bmod n$ (1)

$a^{-1} \equiv x \bmod n$ (2)

Уравнение (2) имеет **единственное** решение, если **a** и **n** – взаимно простые числа, в противном случае (2) решений не имеет.

Пример. При **a**=5 и **n**=14 **x**=3 : **$5^{-1} \equiv 3 \bmod 14$** , т.к. **$(5*3) \bmod 14 = 1$**

Обратные числа в модулярной арифметике

Если **НОД (a,n) =1**, то **$a^{-1}a = 1$ *mod* n**, a^{-1} - число, обратное a по модулю n

Справедливо также, если **$x^{-1} = y$ *mod* n** , то **$y^{-1} = x$ *mod* n**

Если **yx = 1 *mod* n** и **НОД (x,n) =1, НОД (y,n) =1,**

то справедливо

$y^{-1} = x$ *mod* n

$x^{-1} = y$ *mod* n (*)

Уравнения (*) можно записать в ином виде:

$$xy + kn = 1 \qquad (**)$$

k – целое число (результат деления **xy/n**)

Вспомним **Алгоритм Евклида**

Даны два числа – a и b; $a > 0$, $b > 0$, считаем, что $a > b$.
Находим ряд равенств:

$$a = b\,q_1 + r_1, \qquad 0 \le r_1 < b,$$
$$b = r_1\,q_2 + r_2, \qquad 0 \le r_2 < r_1,$$
$$r_1 = r_2\,q_3 + r_3, \qquad 0 \le r_3 < r_2,$$
$$r_2 = r_3\,q_4 + r_4 \qquad 0 \le r_4 < r_3,$$
$$\cdot\ \cdot\ \cdot\ \cdot\ \cdot\ \cdot\ \cdot\ \cdot\ \cdot\ \cdot\ \cdot\ \cdot \qquad \cdot\ \cdot\ \cdot\ \cdot\ \cdot\ \cdot\ \cdot\ \cdot$$
$$r_{n-3} = r_{n-2}\,q_{n-1} + r_{n-1}, \qquad 0 \le r_{n-1} < r_{n-2}\,,$$
$$r_{n-2} = r_{n-1}\,q_n + r_n, \qquad 0 \le r_n < r_{n-1},$$
$$r_{n-1} = r_n\,q_{n+1}, \qquad r_{n+1} = 0,$$

заканчивается, когда получаем некоторое $r_{n+1} = 0$.
Тогда r_n – наибольший общий делитель чисел a и b.

Пример.

Пусть a = 525, b = 231. Найти НОД.

Применим алгоритм Евклида:

525 = 231 * 2 + 63;
231 = 63 * 3 + 42;
63 = 42 * 1 + **21**;
42 = 21 * 2.

Получаем последний положительный остаток r_3= 21.
Таким образом, НОД (525, 231) = 21.

Пример. Решаем уравнение **7x= 1 mod 20** или **x^{-1}= 7 mod 20**

Находим **НОД (7,20):**

20=7*2+6

7=6*1+1 $\longrightarrow$ **обратная подстановка: 1 = 7-6*1 = 7-(20-7*2) =**

= 7-20 +7*2 = 7*3 + 20*(-1) = yx + kn =1,

y=7, x=3, k=-1;

Таким образом, число 3 является обратным числу 7 по модулю 20

Пример. Решаем уравнение **7y = 1 mod 40** или **y^{-1}= 7 mod 40**

Находим **НОД (7,40):**

40=7*5+5

7=5*1+2

5=2*2+1 $\longrightarrow$ **обратная подстановка: 1=5-2*2=5-2(7-5*1)=**

=5*3+7(-2)=(40-7*5)3+7(-2) = 40*3+7(-17) = kn + xy=1 mod n

7(-17) = 7y, так как -17 mod 40 = 23, то y=23, т.е.

23 является обратным числу 7 по модулю 40

Обратные числа по модулю можно вычислить на основе расширенного алгоритма Евклида (листинг см. ниже на C ++)

```
#define isEven(x) ( (x & 0x01) == 0)
# define isOdd(x) (x & 0x01)
# define swap(x,y) (x^= y, y^= x, x^= y)
void ExtBinEuclid(int *u, int *v, int *ul, int *u2, int *u3) {
// warning: u and v will be rearranged if u <v
int k, tl, t2, t3;
if (*u < *v) swap(*u,*v);
for (k = 0; isEven(*u) && isEven(*v); ++k) {
        *u >>= 1; *v >>= 1 ;
}
*ul = 1; *u2 = 0; *u3 = *u; tl = *v; t2 = *u - 1; t3 = *v;
do {
do {
if (isEven(*u3)) {
if (isOdd(*ul) || isOdd(*u2)) {
*ul += *v; *u2 += *u;
}
 *ul >>= 1; *u2 >>= 1; *u3 >>= 1;
}
if (isEven(t3) || *u3 < t3) {
swap(*ul,tl); swap(*u2, t2); swap(*u3, t3);
}
} while (isEven(*u3));
while (*ul < tl || *u2 < t2) {
        *ul += *v; *u2 += *u;
}
*ul -= tl; *u2 -= t2; *u3 -= t3;
} while (t3 > 0);
```

```
while (*ul >= *v && *u2 >= *u) {
*ul -= *v; *u2 -= *u;
                                        }
              *ul <<= k; *u2 <<= k; *u3 <<= k;
}
main(int argc, char  **argv) {
int a, b, gcd;
if (argc < 3) {
cerr << “Using: xeuclid u v" << endl;
return -1;
              }
int u = atoi(argv[l]);
int v = atoi(argv[2J);
 if (u <= 0 || v <= 0 ) {
cerr << " Arguments should be positive! " << endl;
 return -2;
}
// warning: u and v will be rearranged if u <v ExtBinEuclid(&u, &v, &a, &b, &gcd);
cout << a <<" * " << u << " + (-"
<< b << ") * << " << v << " = " << gcd << endl;
                        if (gcd == 1)
cout << " Inverse Value " << v << " mod " << u << " equal to
                        "
                                        << u - b << endl;
return 0;
}
```


Source: Bruce Schneier, Applied Cryptography: Protocols, Algorithms, and Source Code in C, 2nd Edition , 1996

Функция Эйлера

Леонард Эйлер (1707-1783) – швейцарский математик

Опр.4. **Приведенной системой вычетов** по модулю **n** называют подмножество полной системы вычетов, члены которого взаимно просты с **n.**

Пример 3. **Приведенной системой вычетов** по модулю **12** будет подмножество (1, 5, 7, 11).

Если **n** – простое число, в ПСВ по модулю **n** входят числа от **1** до **n-1.**

Ф. Эйлера определяет число элементов в ПСВ по модулю n, т.е. кол-во целых положительных чисел, меньших n и взаимно простых с n; n>1 .

Ф. Эйлера обозначается **φ(n).**

Функция Эйлера

Если – **n** - простое число, то **$\varphi(n) = n-1$.**

Если **$n=p*q$**, где **p** и **q** – простые числа, то

$\varphi(n) = (p-1)*(q-1)$ (3)

Малая теорема Ферма. Если **n** – простое число и **a** не кратно **n**, то справедливо

$a^{(n-1)} \equiv 1 \bmod n$ ($a^{(n-1)} \bmod n \equiv 1$) или **$a^n \equiv a \pmod n$** (4)

Пример. Если ***a* = 2** и ***n* = 7**, то $2^7 = 128$, и $128 - 2 = 7 \times 18$.

или

Обобщение Эйлера МТФ: Если НОД (**a**, **n**) =1, то

$a^{\varphi(n)} \bmod n = 1$ (5)

Нетрудно вычислить **a^{-1}**: (разделив обе части (5) на **a**)

$a^{-1} = a^{\varphi(n)-1} \bmod n$ (6)

Пример 4. Найти число, обратное 5 по модулю 7 (**n**).

Ответ: $\varphi(n)=7-1=6$; $5^{6-1} \bmod 7 = 3$.

3. Проблема дискретного логарифма

При известных **a, x** и **n** легко вычисляется

$y=a^x \bmod n$.

Обратная задача: при известных **y**, **a** и **n** найти **x**.

Это задача вычисления дискретного логарифма:

$a^x \equiv y \bmod n$ (см. соотношение (2)).

Пример 5. Если **$3^x \equiv 15 \bmod 17$**, то **$x=6$.**

Решения существуют не для всех дискретных значений.

Пример 6. Уравнение **$3^x \equiv 7 \bmod 13$** решения не имеет.

При 1024-битовых и более значениях решение задачи может занять десятки и сотни лет

Определение. **Первообразный корень** (**primary** (**residual**) **root**) по модулю **n** является таким числом, что его степени дают все возможные по модулю **n** вычеты, которые взаимно просты с **n**.

Пример. Следующие остатки по модулю 5 от $\mathbf{2}^i$: 2, 4, 3, 1 (они дают <u>все возможные остатки</u>). Число 2 является первообразным корнем по модулю 5.

Пример. Следующие остатки по модулю **7** от $\mathbf{2}^i$: 2, 4, 1, 2, ... (они <u>не дают всех возможных остатков</u>). Число 2 не является первообразным корнем по модулю 7.

Пример. Следующие остатки по модулю **17** от **3**^{i}:

3, 9, 10, 13, 5, 15, 11, 16, 14, 8, 7, 4, 12, 2, 6, 1.

Число 3 является первообразным корнем по модулю 17.

В уравнении

$$y = a^x \bmod n$$

мы используем модуль **n** простого числа, например 17, и находим первообразным корень от 17 в этом случае 3. Он имеет важное свойство: при использовании разных степеней ($a^i = a^x$) решение будет равномерно распределяться от 0 до 16.

Число **3** - это генератор.

Важнейшее свойство ***y = a^x mod n*** - ***однонаправленность***

В современных криптосистемах используется **n** порядка 2048 бит. Ниже приводится пример такого числа.

1615850303565550365035743834433497598022205133485774201606517271376232
7569433945446598600705761456731844358980460949009747059779575245460547
5440761932241415603154386836504980458750988751948260533980288191920337
8413839610932130987808091904716923808523529082292601815252144378794577
0532904303776199561965192760957166694834171210342487393282284747428088
0176631610290389028296655130963542301570751292964320885583629718018592
3092867879917557615082295220184880661664361561356284235541010486257855
0863465661734839271290328348967522998634176499319107762583194718667771
8010677166148023226592393024760740967779268055297981172 47

Понятие хэш-функции

Опр. ***Однонаправленная функция*** предполагает простоту ее вычисления (вычисления **f(x)** по известному аргументу **x**) и сложность обратного вычисления (вычисления **x** по известному **f(x)**)

Опр. ***Хэш-ф***. – математическая или иная ф., которая принимает на входе строку символов переменной (произвольной) длины и преобразует ее в выходную строку фиксированной (обычно – меньшей) длины, называемой **значением х.-функции** или ее **сверткой**

Однонаправленная х.-ф. – основа многих протоколов

Свойства хэш-функции

- Формальная запись: **h=H(M)**
- Зная **M**, легко вычислить **h**
- Зная **h**, трудно определить **M**, для которого **H(M)= h**
- Зная **M**, трудно определить **M’** (**M≠ M’**), для которого **H(M)= H(M’) –** коллизия 1-го рода
- Трудно найти два случайных сообщения (**M** и **M’**), для которых **H(M)= H(M’) -** коллизия 2-го рода