Дисциплина «Алгоритмы решения прикладных задач»
Рабочая тетрадь 7.
Шифрование с открытым ключом. Расширенный алгоритм Евклида. Алгоритм шифрования RSA

Теоретический материал

1) Расширенный алгоритм Евклида

Расширенный алгоритм Евклида — это расширение алгоритма Евклида, которое вычисляет, кроме наибольшего общего делителя (НОД) целых чисел **a** и **b**, ещё и коэффициенты соотношения Безу, то есть целые **x** и **y**, такие что

$$ax + by = \text{НОД}(a, b).$$

Пусть **d** - наибольший общий делитель чисел **a** и **b**. Тогда выражение **ax+by** всегда кратно **d**. Оказывается, что можно подобрать такие числа **x** и **y**, что **ax+by=d**. Эту задачу решает расширенный алгоритм Евклида. Рассмотрим его рекурсивную реализацию. Пусть функция gcdex получает на вход числа **a** и **b** и возвращает кортеж из трех чисел **d**, **x**, **y**, где **d** - наибольший общий делитель **a** и **b**, а **x** и **y** - такие целые числа, что **ax+by=d**.
Условием окончания рекурсии является **b=0**. В этом случае **d=a, x=1, y=0**. Если же **b≠0**, то вызовем функцию рекурсивно для чисел **b** и **a%b** и получим ответ для исходных чисел.

2) Нахождение обратного элемента по модулю.

Задача вычисления обратного элемента: по данным числам x, n, найти такое число x^{-1}, что $x \cdot x^{-1} \equiv 1 \pmod{n}$. Сперва запускается расширенный алгоритм Евклида для (n, x). Он выдаёт тройку $(1, i, j)$, где $in + jx \equiv 1 \pmod{n}$. Отсюда следует $jx \equiv 1 \pmod{n}$, и потому $j = x^{-1}$.

Пример 2. *Пример: вычислить* 3^{-1} *по модулю* 35. *Запускается расширенный алгоритм Евклида для* $(35, 3)$, *он выдаёт* $(1, -1, 12)$. *Ответ:* 12; *и верно,* $3 \cdot 12 \equiv 1 \pmod{35}$.

3) Возведение в степень по модулю

Чтобы вычислить x^y % N, нужно перемножить те степени x, которые соответствуют ненулевым позициям в двоичной записи y. Например,

$25_{10} = 11001_2$

$$x^{25} = x^{2^4} \cdot x^{2^3} \cdot x^{2^0} = x^{16} \cdot x^8 \cdot x^1$$

```cpp
#include <iostream>
using namespace std;
int modexp(int x, int y, int N)
{
  if (y == 0) return 1;
  int z = modexp(x, y / 2, N);
  if (y % 2 == 0)
    return (z*z) % N;
  else
    return (x*z*z) % N;
}
int main()
{
  int x, y, N;
  cout << "x= "; cin >> x;
  cout << "y= "; cin >> y;
  cout << "N= "; cin >> N;
  cout << modexp(x, y, N);
  cin.get(); cin.get();
  return 0;
}
```

4 .Алгоритм RSA

4) Создание открытого и секретного ключа

RSA-ключи генерируются следующим образом:

1) выбираются два различных случайных простых числа p и q заданного размера (например, 1024 бита каждое);

2) вычисляется их произведение $n = p \cdot q$, которое называется модулем;

3) вычисляется значение функции Эйлера от числа n:

$\varphi(n) = (p-1) \cdot (q-1)$;

4) выбирается целое число

$1 < e < \varphi(n)$, взаимно простое со значением функции $\varphi(n)$;

число e называется открытой экспонентой (англ. public exponent);

обычно в качестве e берут простые числа, содержащие небольшое количество единичных бит в двоичной записи, например,

простые из чисел Ферма: 17, 257 или 65537, так как в этом случае время, необходимое для шифрования с использованием

быстрого возведения в степень, будет меньше;

слишком малые значения e, например 3, потенциально могут ослабить безопасность схемы RSA.

5) вычисляется число

d, мультипликативно обратное к числу e по модулю $\varphi(n)$, то есть число, удовлетворяющее сравнению:

$d \cdot e \equiv 1 \pmod{\varphi(n)}$

(d называется секретной экспонентой; обычно оно вычисляется при помощи расширенного алгоритма Евклида);

6) пара (e,n) публикуется в качестве открытого ключа RSA (англ. RSA public key);

7) пара (d,n) играет роль закрытого ключа RSA (англ. RSA private key) и держится в секрете.

5) Шифрование

- Взять *открытый ключ (e, n)*
- Взять *открытый текст* m
- Зашифровать сообщение с использованием открытого ключа
- $c = E(m) = m^e \mod n$

6) Расшифрование

- Взять *шифрованный текст c*
- Взять *закрытый ключ (d, n)*
- Применить закрытый ключ для расшифрования сообщения^

$m = D(c) = c^d \mod n$

Задание 1

Напишите программу для реализации расширенного алгоритма Евклида

Решение:

```go
func nodExt(a, b int) (int, int, int) {
  if b == 0 {
    return a, 1, 0
  }
  d, x1, y1 := nodExt(b, a%b)
  x := y1
  y := x1 - (a/b)*y1
  return d, x, y
}
```

Ответ:

```
Введите два числа: 121 11
НОД(121, 11) = 11
Коэффициенты: x = 0, y = 1
Введите два числа: 505 25
НОД(505, 25) = 5
Коэффициенты: x = 1, y = -20
Введите два числа: 169 49
НОД(169, 49) = 1
Коэффициенты: x = -20, y = 69
Введите два числа: 284 16
НОД(284, 16) = 4
Коэффициенты: x = -1, y = 18
Введите два числа: 12415424314 908512
НОД(12415424314, 908512) = 2
Коэффициенты: x = 16549, y = -226153157
```

Задание 2

Напишите программу для шифрования сообщения алгоритмом RSA (с генерацией открытого и секретного ключей)

Решение:

```
func nodExt(a, b *big.Int) (gcd, x, y *big.Int) {
  if b.Cmp(big.NewInt(0)) == 0 {
    return a, big.NewInt(1), big.NewInt(0)
  }
  gcd, x1, y1 := nodExt(b, new(big.Int).Mod(a, b))
  x = y1
  y = new(big.Int).Sub(x1, new(big.Int).Mul(new(big.Int).Div(a, b), y1))
  return gcd, x, y
}

func modInverse(e, phi *big.Int) (*big.Int, error) {
  gcd, x, _ := nodExt(e, phi)
  if gcd.Cmp(big.NewInt(1)) != 0 {
    return nil, errors.New("обратного элемента не существует")
  }
  return new(big.Int).Mod(x, phi), nil
}

// генерация простого числа заданного размера
func generatePrime(bits int) (*big.Int, error) {
  prime, err := rand.Prime(rand.Reader, bits)
  if err != nil {
    return nil, err
  }
  return prime, nil
}

// генерация пары ключей
func generateRSAKeys(bits int) (*big.Int, *big.Int, *big.Int, *big.Int, error) {
  p, err := generatePrime(bits / 2)
  if err != nil {
    return nil, nil, nil, nil, err
  }
  q, err := generatePrime(bits / 2)
  if err != nil {
    return nil, nil, nil, nil, err
  }
  n := new(big.Int).Mul(p, q)
  phi := new(big.Int).Mul(new(big.Int).Sub(p, big.NewInt(1)), new(big.Int).Sub(q,
big.NewInt(1)))
  e := big.NewInt(65537)
  d, err := modInverse(e, phi)
  if err != nil {
    return nil, nil, nil, nil, err
  }
  return n, e, d, phi, nil
}

// шифрование с использованием открытого ключа
func encryptRSA(message, e, n *big.Int) *big.Int {
  ciphertext := new(big.Int).Exp(message, e, n)
  return ciphertext
}

// расшифровка с использованием закрытого ключа
func decryptRSA(ciphertext, d, n *big.Int) *big.Int {
  plaintext := new(big.Int).Exp(ciphertext, d, n)
  return plaintext
}
```

Ответ:

```
Введите текст для шифрования: привет
Исходное сообщение: привет
Зашифрованное сообщение: 2364862124034733790982293361784175083928719200232191333329931018407991419364195936040928951471895291166954306041333740207686940051383992543931140708210160798958734219402230114356448915992879426831790952650488552005102856680552107401773041286565284830391411487832654810988134850521647285022944977112952295154702155770902668844322293678262463540764131765691934316130814845296979086847234602857730604496144510931937293845027241764009211435604078901728368934325560347206120282450944481341597173148326688965382136788472923525603213377445198253609049933118556398627141829950400692554717649105163088298236878629012378610154 7
Расшифрованное сообщение: привет
Введите текст для шифрования: привет
Исходное сообщение: привет
Зашифрованное сообщение: 1391014059171977943703184818549350781376079874584707048108629920234831143622462360156236222017319419476736314132930704903870128838852969419161756645465490947840722859919191994424291866669228677462107106145028138239476228483723544718786704566288043587776505611481991242694592950102688215023509643109356680627362490008534836930559043883592630020363457910307126807704352393440254116712344646527804202994257810880933902007397716313221645812694174374079902080318510799020297840162761390535827525939435098716739384952394712302600853739455122382146327010784380543872800853108438516183454468544834672246264185989693828475140 9
Расшифрованное сообщение: привет
Введите текст для шифрования: 123
Исходное сообщение: 123
Зашифрованное сообщение: 2374768898875471246823123269135512925917599474985603158376330459680240666129210611603587759097337224476628368128028722513112993666289549536149800878974169415094941771356661816726573674359278607969755759844780818911902460661123969958144828614731542200534891407429059770181198270805711909834842565015806622277664555527844685959066863521927768960009149308084174786083286378542409415879140208785210675028853716081184770653269421635330953424939484098377390004745394600709207439286423002325628094740796862885422765854404833686884579439431059577629374044704612941532592782054923524916041696239009510080046154116422845338036
Расшифрованное сообщение: 123
Введите текст для шифрования: что?
Исходное сообщение: что?
Зашифрованное сообщение: 1024442861768708620745721441238169468248497014056283505613976669940869924841664000616888266870374004779536991440393134609754512884954352099967453863813405109516449922335154102802585901839606358125353207348835183408991522207028238592806968335303248397121858944816504913751329192795178624249442334941450169005704634411043368202026430999053637953915382521294547969519752115234103135720772498876840592413868827931049754395593054933185553450476748004109193906998049008185955662850060547744309242376808700475500374495161451080814396253161249650296894037089598679588489840861418557634545641354972420584674744105156327358635 5
Расшифрованное сообщение: что?
```