Exercicis Aritmètica Modular i RSA

Albert Ribes Marzá Víctor Alcázar Kosmas Palios

2 d'abril de 2017

Resum

1. Calculeu:

   (a) $3^{28} \bmod 10$

$$3^{2\times 14} \bmod 10 \tag{1}$$

$$(3^2)^{14} \bmod 10 \tag{2}$$

$$9^{14} \bmod 10 \tag{3}$$

$$(-1)^{14} \bmod 10 \tag{4}$$

$$1 \bmod 10 \tag{5}$$

   (b) $3^{200} \bmod 15$

$$3^{200} \bmod 15 \tag{6}$$

$$3^4 \times 50 \bmod 15 \tag{7}$$

$$81^{50} \bmod 15 \tag{8}$$

$$6^{50} \bmod 15 \tag{9}$$

   Com que $6^2 \bmod 15 = 6$, el resultat és:

$$6 \bmod 15 \tag{10}$$

2. **2EXP modular.** Doneu un algorisme de temps polinòmic que amb entrada els enters a, b, c i un nombre primer p computi $a^{b^c} \bmod p$

   Ens podem adonar del següent:
   $a^{b^c} \bmod m = a^{b^{c-1}\times b} \bmod m = (a^b)^{b^{c-1}} \bmod m$
   Per tant podem aplicar el següent algorisme `doubleExp`:

```
int simpleExp(a,b,mod)
if  b == 0  then
   return  1
else if b % 2 == 0 then
   return  simpleExp( (a * a) % mod,b/2,mod)
else
   return  a*simpleExp( (a * a)% mod, b/2, mod) % mod
end if
```

I llavors es por fer servir aquest:

```
double doubleExp(a, b, c, mod)
if  c == 1  then
   return  simpleExp(a, b, mod)
else
   return  doubleExp(simpleExp(a, b, mod), b, c-1, mod)
end if
```

3. **Factorial Modular.** Donats dos enters x i N, calcula $x! \bmod N$.

   (a) Demostreu que un enter y és primer si i només si per tot enter $x < y$ es compleix que $gdc(x!, y) = 1$.

   Hem de demostrar que:

   $$y \text{ primer } \Leftrightarrow \forall x < y : \text{gcd(x!,y)} = 1 \tag{11}$$

   La demostració té dos apartats:

   - $\Rightarrow$
     EL gcd de qualsevol nombre primer sempre és 1 per un nombre primer i qualsevol altre, i es imposible que $x! == y$ ja que y no té factors. Queda demostrat
   - $\Leftarrow$
     Ara demostrarem que:
     y primer $\Leftarrow \forall x < y : \text{gcd(x!,y)} = 1$

     Ho farem amb el contrarrecíproc:
     y compost $\Rightarrow \exists x < y : \text{gcd(x!,y)} \neq 1$
     Com que y és compost, existeixen dos naturals tals que $y = ab$, i es compleix que tots dos són menors que y.
     Llavors ja hem trobat un nombre menor que y que compleix la condició: a
     $d = \text{gcd(a!, ab)}$
     Està clar que a divideix d i que a no és 1, per tant d no pot ser 1. Queda demostrat.

   (b) Considera l'apartat previ per demostrar que si **Factorial Modular** fos computable en temps polinòmic, aleshores el problema de **Factoritzar** també sería computable en temps polinòmic (Recordeu **Factoritzar**: Donat un nombre enter x, calcula els seus factors primers).

4. En un sistema criptogràfic **RSA** amb $p = 7$ i $q = 11$, troba la clau pública (N, c) i la clau privada (N, d) apropiades.

Els passos per trobar les claus RSA amb dos primers p i q són:

- Computar $N = pq$
- Computar $\phi(N) = (p-1)(q-1)$
- Escollir $c \in \mathbb{Z}^*_{\phi(N)}$
- Computar d tal que $cd \equiv 1$ mod ϕ(N)
- La clau pública és $P_B = (c, N)$ i la clau privada és $P_S = (d, N)$

Llavors els passos que fem són:

- $N = 7 \cdot 11 = 77$
- $\phi(N) = (7-1)(11-1) = 6 \cdot 10 = 60$
- Podem escollir entre moltes parelles per a c i d. Algunes de elles son $(7, 43)$, $(11, 11)$, $(13, 37)$, $(17, 53)$, $(19, 19)$, $(59, 59)$, $(23, 47)$, $(27, 47)$, $(7, 1)$. Nosaltres escollim 17 i 55
- Llavors les claus són: $P_B = (17, 77)$ i $S_B = (55, 77)$

5. **Sistema criptogràfic segur?.** Suposem que en lloc d'utilitzar un nombre compost $N = pq$ com es fa en el sistema RSA, utilitzem un nombre primer p. Per encriptar un missatge m mod p farem servir un exponent e, de la mateixa manera que es fa en el sistema RSA. L'encriptament del missatge m mod p seria m^e mod p.

   Demostreu que aquest nou sistema no és segur donant un algorisme eficient per desencriptar. És a dir, doneu un algorisme que, amb entrada p, e, m^c mod p, computi m mod p eficientment. Justifica la correctesa de l'algorisme i analitza el seu temps de computació.

   La resposta comença aqui

   In the given cryptosystem, to decrypt a ciphertext we must use the private key d, in the following way $m = c^d modp$. The problem here is that the private key can be easily computed. We know that the relation between e and d is $ed = 1 mod \phi(p)$. But now $\phi(p) = p - 1$, because p is prime!

   So all we have to do is compute the multiplicative inverse of e modulo p-1. This can be easily done using the Extented Euclidean Algorithm, which takes $O(log(e)2)$ time. Then it is only a matter of exponentiation of c to the power of d modulo p. This in itself takes log(d) steps if done with repeated squaring.

   In total we have $O(log(e)2 + log(d))$ time.