Mănăilă Denis-Daniel, Gr.4.1, An III, CTI-RO, AC-UPT

RSA key generator – MIPS

1.Descrierea problemei

RSA [1] este un algoritm de criptare a datelor cu cheie publică.

Acesta presupune:

a. Alegerea a doua numere prime și înmulțirea acestora.

fie p,q prime a.î

$$n = p * q$$

b. Determinarea lui ϕ (n) (numit și indicatorul lui Euler)

- acesta arata totalul numerelor care sunt prime între ele cu n și mai mici decât n

- s-a demonstrat că această valoare în cazul numerelor prime între ele este:

$$\phi(n) = m = (p - 1) * (q - 1)$$

c. Alegem un număr “e” astfel încât “e” și “m” să fie prime între ele și e < m

d. Calculăm numărul “d” care satisface relația:

$$d * e \% m = 1$$

În acest moment suntem pregătiți pentru a cripta mesaje. (n,e) sunt chei publice , iar d este cheie privată. (NU se distribuie)

Pentru a cripta un mesaj folosim relația :

$$C = M^e \% n \text{ , C = mesaj codat, M = mesaj}$$

Pentru a decripta un mesaj folosim relația:

$$D = C^d \% n, \text{ D = mesaj decodat, C = mesaj criptat}$$

2.Implementare

Lucrarea cuprinde 3 programe de assembly: un generator de chei RSA, un criptor de mesaje și un decriptor.

RSA: partea de „main”

```
.text
        jal sel_p
        jal sel_q
        jal prnt_e
        jal calc_n
        jal calc_phi

        #afisarea si calculul lui d
        li $v0,4
        la $a0,d_msg
        syscall
        lw $t3,e
        move $a1,$t3
        lw $t3,phi
        move $a2,$t3
        jal mod_inv
        sw $v0,d
        move $t3,$v0
        move $a0,$t3
        li $v0,1
        syscall
        li $v0,4
        la $a0,lf
        syscall

        li $v0,10 #Halt
        syscall
```

Rutinele sel_p și sel_q:

```
sel_p:  li $v0,4
        la $a0,p_msg
        syscall
        la $s1,p_set            # mi-am declarat un array cu prime_nr elemente
        lw $t1,prime_nr
        move $a1,$t1
        li $v0,42               # syscall 42 alege random un numar cu val maxima $a1 (prime_nr)
        syscall
        move $s2,$a0
        mul $s3,$s2,4           # stiu ca un int este 4B deci adresele sunt din 4 in 4
        add $s1,$s1,$s3         # la adresa de start adaug 4 * numarul ales random
        lw $t0,($s1)            # dereferentiez valoarea adresei si dau de numarul dorit
        sw $t0,p
        move $a0,$t0
        li $v0,1
        syscall
        li $v0,4
        la $a0,lf
        syscall
        jr $ra
```

Rutina alege și afișează un numar aleator dintr-un array hard-codat.

Sel_q face același lucru pentru numărul q.

Mănăilă Denis-Daniel, Gr.4.1, An III, CTI-RO, AC-UPT

Rutina prnt_e: afișez numărul „e”

care este hard-codat. (e = 17)

```
prnt_e: li $v0,4
        la $a0,e_msg
        syscall
        lw $a0,e
        li $v0,1
        syscall
        li $v0,4
        la $a0,lf
        syscall
        jr $ra
```

Rutinele pentru calculele și afișările lui n si phi:

```
calc_n: li $v0,4                # calculez n
        la $a0,n_msg
        syscall
        lw $t0,p
        lw $t1,q
        mul $t2,$t1,$t0         # n = p * q
        sw $t2,n
        li $v0,1
        move $a0,$t2
        syscall
        li $v0,4
        la $a0,lf
        syscall
        jr $ra
```

```
calc_phi: li $v0,4              # calculez phi
          la $a0,phi_msg
          syscall
          lw $t0,p
          lw $t1,q
          sub $t0,$t0,1
          sub $t1,$t1,1
          mul $t2,$t1,$t0       # phi = ( p - 1 ) * ( q - 1 )
          sw  $t2,phi
          move $a0,$t2
          li $v0,1
          syscall
          li $v0,4
          la $a0,lf
          syscall
          jr $ra
```

Partea de calcul a lui “d” și rutina mod_inv (modulo_inverse) [2]

```
#afisarea si calculul lui d
li $v0,4
la $a0,d_msg
syscall
lw $t3,e
move $a1,$t3
lw $t3,phi
move $a2,$t3
jal mod_inv
sw $v0,d
move $t3,$v0
move $a0,$t3
li $v0,1
syscall
li $v0,4
la $a0,lf
syscall

li $v0,10 #Halt
syscall
```

```
# calculez d astfel in incat ((d*e) mod n) = 1
# algoritmul este preluat de aici
# https://www.geeksforgeeks.org/multiplicative-inverse-under-modulo-m/

mod_inv: move $s1,$a1 #a
         move $s2,$a2 #m
         li $s3,0     #y
         li $s4,1     #x
while_1: div $s1,$s2
         mflo $s5     #q
         move $s6,$s2 #t
         mfhi $s2
         move $s1,$s6
         move $s6,$s3
         mul $s7,$s5,$s3
         sub $s3,$s4,$s7
         move $s4,$s6
         bgt $s1,1,while_1
end:     bgez $s4,fin
         add $s4,$s4,$a2
fin:     move $v0,$s4
         jr $ra
```

Datele declarate:

```
.data
p:       .word  0
q:       .word  0
phi:     .word  0
e:       .word 17
d:       .word 0
n:       .word 0
lf:      .asciiz "\n"
n_msg:   .asciiz "n:"
phi_msg: .asciiz "phi:"
e_msg: .asciiz "e:"
p_msg: .asciiz "p:"
q_msg: .asciiz "q:"
d_msg: .asciiz "d:"
prime_nr: .word 14
p_set : .word 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113
q_set : .word 431, 433, 439, 443, 449, 457, 461, 463, 467, 479, 487, 491, 499, 503
```

De asemenea am creat un criptor și un decriptor de cuvinte.

Criptor:

```
.text
        li $v0,4
        la $a0,e_msg
        syscall

        li $v0,5
        syscall
        sw $v0,e

        li $v0,4
        la $a0,n_msg
        syscall

        li $v0,5
        syscall
        sw $v0,n

        li $v0,4
        la $a0,txt_msg
        syscall

        li $v0,8
        li $a1,19
        la $a0,txt
        syscall

        li $v0,4
        la $a0,lf
        syscall

        la $t5,txt
        lb $t2,($t5)
while:  move $a1,$t2
        lw $a2,e
        lw $a3,n
        jal mod_c

        move $a0,$v0
        li $v0,1
        syscall

        la $a0,sp
        li $v0,4
        syscall

        add $t5,$t5,1
        lb $t2,($t5)
        bne $t2,10,while

        la $a0,lf
        li $v0,4
        syscall

        li $v0,10
        syscall

mod_c:  move $s1,$a1 #x
        move $s2,$a2 #y
        move $s3,$a3 #p
        li $s4,1     #res
        li $t1,2
        divu $s1,$s3
        mfhi $s1
while_2: divu $s2,$t1
         mfhi $s5
         beq $s5,0,even
         mul $s7,$s4,$s1
         divu $s7,$s3
         mfhi $s4
even:   divu $s2,$t1
        mflo $s2
        mul $s6,$s1,$s1
        divu $s6,$s3
        mfhi $s1
        bgtz $s2,while_2
        move $v0,$s4
        jr $ra
```

Programul se folosește de algoritmul [3] care este bazat pe următoarea proprietate:

$$(a * b) \bmod n = ((a \bmod n) * (b \bmod n)) \bmod n$$

Fapt important când ridici la putere chiar și numere mici. De exemplu, caracterul “A” în ASCII:

$$65 \wedge 17 \bmod n$$

Criptorul citește un șir de caractere și criptează fiecare caracter din string pe baza valorilor “n” și “e” citite.

Decriptor:

```
.text
        li $v0,4
        la $a0,d_msg
        syscall

        li $v0,5
        syscall
        sw $v0,d

        li $v0,4
        la $a0,n_msg
        syscall

        li $v0,5
        syscall
        sw $v0,n

        li $v0,4
        la $a0,lf
        syscall

        li $v0,5
        syscall
        sw $v0,nr

        la $t5,txt
while:  lw $a1,nr
        lw $a2,d
        lw $a3,n
        jal mod_c
        sb $v0,($t5)

        li $v0,5
        syscall
        sw $v0,nr
        add $t5,$t5,1
        bnez $v0,while

        li $t2,0
        sb $t2,($t5)

        la $a0,txt
        li $v0,4
        syscall

        li $v0,10
        syscall
mod_c:  move $s1,$a1 #x
        move $s2,$a2 #y
        move $s3,$a3 #p
        li $s4,1     #res
        li $t1,2
        divu $s1,$s3
        mfhi $s1
while_2: divu $s2,$t1
         mfhi $s5
         beq $s5,0,even
         mul $s7,$s4,$s1
         divu $s7,$s3
         mfhi $s4
even:   divu $s2,$t1
        mflo $s2
        mul $s6,$s1,$s1
        divu $s6,$s3
        mfhi $s1
        bgtz $s2,while_2
        move $v0,$s4
        jr $ra
```

Decriptorul se folosește de același algoritm [3] ca și criptorul pentru a calcula modulo-urile.

Acesta citește valorile n și d, apoi, numerele criptate până la citirea valorii “0”, după care afișează la output șirul decriptat.

3.Exemplu de funcționare

Începem prin a folosi generatorul de chei:

```
p:61
q:431
e:17
n:26291
phi:25800
d:4553

-- program is finished running --
```

După care folosim criptor-ul pentru a ascunde un mesaj:

```
e = 17
n = 26291
text:BOMB

10528 1307 12700 10528

-- program is finished running --
```

În ultimul rând folosim decriptor-ul pentru a vedea ce am transmis:

```
d = 4553
n = 26291

10528
1307
12700
10528
0
BOMB
-- program is finished running --
```

4.Concluzii:

4.1 Blocking points:

- Calculul lui “d”
  - Inițial nu am realizat cât de repede crește valoarea “n” așa că m-am gândit să fac un loop pentru d pornind de la 1 până la n
  - Proastă idee pentru că am plecat să fac o cafea, iar când m-am întors algoritmul încă mergea.
  - Așa că după o căutare rapidă pe internet am dat de algoritmul extins al lui Euclid, care este o combinație între CMMDC și identitatea lui Bézout

- Calculul modulelor
  - Din nou am încercat să fac calculul prin forță brută
  - Proastă idee pentru că chiar și la criptare făceam overflow (10^{17} mod n) , nu mai vorbim de datele criptate unde puteam ajunge la (10250^{4500} mod n)
  - Astfel că din nou după o altă căutare am dat de proprietatea exemplificată anterior în documentație

4.2 Comentarii legate de implementare:

- Acesta este un “demo” pentru RSA deoarece lucrează cu numerele propriu-zise pe 32 de biți.
- În practică se folosesc numere prime de 1024 de biți pentru a genera un cod RSA pe 2048 de biți.
- Cum nu există tip de date de 1024 de biți toți algoritmii trebuie adaptați pentru așa numiții “algoritmi de calcul cu numere mari”
- Prima problemă este generarea numerelor
  - Generezi un număr aleator de 1024 de biți
  - Aceste numere sunt salvate ca string-uri (ex. char a[1025]) unde fiecare caracter reprezintă o cifră a numărului real
  - Verificarea că este prim (Ciurul lui Eratostene)
- A doua problemă : Cum nu există tip de date de 1024 de biți toți algoritmii aritmetici simpli trebuie adaptați pentru așa numiții “algoritmi de calcul cu numere mari”

4.3 Ce am învățat?

- Niciodată nu am realizat cât de bine sunt protejate datele noastre
- Cât de mult ne ușurează munca un limbaj de programare de nivel înalt
- Bazele programării la într-un limbaj low-level

5. Bibliografie

[1] Chuck Easttom , "Modern Cryptography: Applied Mathematics for Encryption and Information Security"

[2] https://www.geeksforgeeks.org/multiplicative-inverse-under-modulo-m/

[3] https://www.geeksforgeeks.org/modular-exponentiation-power-in-modular-arithmetic/

[4] https://en.wikipedia.org/wiki/Extended_Euclidean_algorithm

[5] https://en.wikipedia.org/wiki/RSA_(cryptosystem)