Cryptography with RSA

Rafael de Lucena Valle, Universidade Federal de Santa Catarina October 27, 2014

Criptografia por RSA

Algoritmo

Geração de Chaves

1) Escolho dois números primos grandes, P and Q. Fazemos isso utilizando utilizando a geração de números randômicos e o teste de primalidade de Miller Rabin.

2) Calculamos $N = P * Q$

3) Calculamos $M = \Phi($P * Q$) = \Phi(P) * \Phi(Q)$, P e Q primos, assim: $M = (P-1)*(Q-1)$

4) Escolhemos um número E coprimo a M, ou seja o máximo divisor comum igual a 1, utilizamos o algoritmo de Euclides. Para fazer estes calculos com números grandes é necessário utilizar o algoritmo extendido de Euclides, que recebe dois inteiros positivos a, b as e retorna uma tripla (g, x, y), tal que $ax + by = g = gcd(a, b)$.

5) Encontro $D = E^{-1} \bmod M$, isto é equivalente a encontrar D que satisfaça $D*E = 1+X*M$, aonde X é qualquer inteiro. Podemos reescrever a fórmula de modo $D = (1 + X * M)/E$.

Ao final D será a chave privada e E será a chave pública. O número a ser criptografado deve ser menor que P e Q.

Para Criptografar H:
$C = H^E \bmod N$

Para Decriptar:
$H = C^D \bmod N$

Exemplos

$P = 3$
$Q = 11$
$N = 3 * 11 = 33$
$M = (3-1)(11-1) = 20$

Encontramos E, tal que $MDC(E, M) = 1$
para $E = 2$:
=> $MDC(2, 20) = 2$
para $E = 3$:

=> $MDC(3, 20) = 1$ (3 é coprimo a 20)

Encontramos $D = (1 + X * M)/E$.

para $X = 0$:

=> $D = 1/3$

para $X = 1$:

=> $D = 21/3$

=> $D = 7$

Criptografando a mensagem $Z = 2$:

$C = Z^E \bmod N$

$C = 2^3 \bmod 33 = 8$

Decriptando:

$Z = C^D \bmod N$

$2 = 8^7 \bmod 33 = 2$

Código

```
import random
import copy

"""
Decompoe um numero par na forma (2^r) * s
"""
def decomposeBaseTwo(n):
    exponentOfTwo = 0
    while n % 2 == 0:
      n = n/2
      exponentOfTwo += 1

    return exponentOfTwo, n

"""
Exponenciacao binaria, baseado no pseudo-codigo do livro Applied Cryptography do Bruce Schneier.
"""
def modularPow(base, exp, modu):
    res = 1
    base = base % modu
    while exp > 0:
        if (exp % 2) == 1:
            res = (res * base) % modu
        exp = exp / 2
        base = (base * base) % modu
    return res

"""
Verifica as condicoes
```

```
    Se (a^s === 1 (mod n) ou a^2js === -1 (mod n)
    para um j | 0 <= j <= r-1
"""
def fillPrimeConditions(candidateNumber, p, exponent, remainder):
   candidateNumber = modularPow(candidateNumber, remainder, p)

   if candidateNumber == 1 or candidateNumber == p - 1:
      return False

   for _ in range(exponent):
      candidateNumber = modularPow(candidateNumber, 2, p)

      if candidateNumber == p - 1:
         return False

   return True

"""
  O numero randomico a na faixa que inicia em 2 pois, o teste 1^s = 1(mod n)
  Seria uma tentavia inutil
"""
def probablyPrime(p, accuracy=100):
   if p == 2 or p == 3: return True
   if p < 2: return False

   numTries = 0
   exponent, remainder = decomposeBaseTwo(p - 1)

   for _ in range(accuracy):
      candidateNumber = random.randint(2, p - 2)
      if fillPrimeConditions(candidateNumber, p, exponent, remainder):
         return False

   return True


'''returns the Greatest Common Divisor of a and b'''
def euclid(a,b):
    a = abs(a)
    b = abs(b)
    if a < b:
        a, b = b, a

    while b != 0:
        a, b = b, a % b

    return a

'''returns 'True' if the values in the list L are all co-prime
   otherwier, it returns 'False'. '''

def coprime(L):
    for i in range (0, len(L)):
        for j in range (i + 1, len(L)):
            if euclid(L[i], L[j]) != 1:
                return False

    return True

```

```
def extendedEuclid(a, b):
    x,y, u,v = 0,1, 1,0
    while a != 0:
        q,r = b//a,b%a; m,n = x-u*q,y-v*q
        b,a, x,y, u,v = a,r, u,v, m,n
    return b, x, y

'''returns the multiplicative inverse of a in modulo m as a positve value between zero and m-1'''
def multiplicativeInverse(a, m):
    if coprime([a, m]) == False:
        return 0
    else:
        linearcombination = extendedEuclid(a, m)
        return linearcombination[1] % m

def randomWithNDigits(n):
    range_start = 10**(n-1)
    range_end = (10**n)-1
    return random.randint(range_start, range_end)

def generateRandomPrime(digits, precision):
    random_number = randomWithNDigits(digits)
    while (probablyPrime(random_number, precision) == False):
        random_number = randomWithNDigits(digits)
    return random_number

''' Try to find a large pseudo primes and generate public and private keys for RSA encryption.'''
def generateKeys(a,k):
    p = generateRandomPrime(a, k)
    while True:
        q = generateRandomPrime(a, k)
        if q != p:
            break

    n = p * q
    m = (p-1) * (q-1)
    while True:
        e = random.randint(1, m)
        if coprime([e, m]):
            break
    d = multiplicativeInverse(e, m)
    return (n, e, d)

'''Converts a string to a list of integers based on ASCII values, printable characters range is 0x20 - 0x7E.'''
def string2numList(strn):
    returnList = []
    for chars in strn:
        returnList.append(ord(chars))
    return returnList

'''Converts a list of integers to a string based on ASCII values'''
def numList2string(L):
    returnList = []
    returnString = ''
    for nums in L:
        returnString += chr(nums)
    return returnString

'''Take a list of integers(each between 0 and 127), and combines them into block size n using base 256. If len(L) % n != 0, use some random junk to fill L to make it '''
def numList2blocks(L,n):
    returnList = []
    toProcess = copy.copy(L)
    if len(toProcess) % n != 0:
        for i in range (0, n - len(toProcess) % n):
            toProcess.append(random.randint(32, 126))
    for i in range(0, len(toProcess), n):
        block = 0
        for j in range(0, n):
            block += toProcess[i + j] << (8 * (n - j - 1))
        returnList.append(block)
    return returnList

'''inverse function of numList2blocks.'''
def blocks2numList(blocks,n):
    toProcess = copy.copy(blocks)
    returnList = []
    for numBlock in toProcess:
        inner = []
        for i in range(0, n):
            inner.append(numBlock % 256)
            numBlock >>= 8
        inner.reverse()
        returnList.extend(inner)
    return returnList

'''given a string message, public keys and blockSize, encrypt using RSA algorithms.'''
def encrypt(message, modN, e, blockSize):
    cipher = []
    numList = string2numList(message)
    numBlocks = numList2blocks(numList, blockSize)
    for blocks in numBlocks:
        cipher.append(modularPow(blocks, e, modN))
    return cipher

'''reverse function of encrypt'''
def decrypt(secret, modN, d, blockSize):
    numBlocks = []
    numList = []
    for blocks in secret:
        numBlocks.append(modularPow(blocks, d, modN))
    numList = blocks2numList(numBlocks, blockSize)
    message = numList2string(numList)
    return message

if __name__=='__main__':
    digits = int(raw_input("Give the size of random number in digits: "))
    precision = int(raw_input("Which precision to test primality? "))
    message = raw_input("Which message to be used? ")
    (n, e, d) = generateKeys(digits, precision)
    print ('n = {0}'.format(n))
    print ('Public Key e = {0}'.format(e))
    print ('Private Key d = {0}'.format(d))
    cipher = encrypt(message, n, e, len(message))
    print('The Cipher is = {0}'.format(cipher[0]))
    newMessage = decrypt(cipher, n, d, len(message))
    print 'The Decrypt message is: ', newMessage
```

Execução

Para executar o script basta abrir um terminal e utilizar um shell tipo o bash, o script é interativo.

Listing 1: Executando o script.

```
python rsa.py
Give the size of random number in digits: 30
Which precision to test primality? 50
Which message to be used? Testando RSA
n = 1515264926296438943142200350022183748919441655974998849969 11
Public Key e = 108451208136200938592600328144194051182428384664500311106857
Private Key d = 321854211995001520366765470823834554800228056499614024985 69
The Cipher is = 117137340459652070495657539285125162635569908944187313923826
The Decrypt message is:  Testando RSA
```