Aluno: Gabriel Guimarães Almeida de Castro - 150126425 - Segurança Computacional - 10/05/2021

Criptografia RSA 1024bits

O sistema de criptografia RSA é um dos primeiros sistemas de chave pública, e o nome são as iniciais dos sobrenomes dos criadores (Rivest-Shamir-Adleman). A dificuldade para descriptografar algo criptografado por RSA se deve ao tempo computacional necessário para fatorar números primos grandes.

Sendo assim, para criptografar e descriptografar não é necessário muito tempo quando se conhece as chaves. Mas quando não se conhece o tempo necessário pode ser inviável.

Apesar de toda a segurança, devido ao uso de blocos no algorítmo mostrado abaixo, é possível descriptografar mais rapidamente devido a frequência com que certos blocos aparece. Algo que pode aumentar a dificuldade é usar números junto.

Funções básicas

Segue abaixo algumas funções matemáticas básicas como o algorítmo euclidiano para máximo divisor comum, o algorítmo euclidiano estendido e a função totiente de euler.

In [106]:
```
"""
Criptografia RSA 1024bits.
"""
import random

def gcd(a, b):
    # Algoritmo Euclidiano de máximo divisor comum
    while a != 0:
        a, b = b % a, a
    return b

def xgcd(a, b):
    #Algoritmo Euclideano estendido
    x, old_x = 0, 1
    y, old_y = 1, 0

    while (b != 0):
        quotient = a // b
        a, b = b, a - quotient * b
        old_x, x = x, old_x - quotient * x
        old_y, y = y, old_y - quotient * y

    return a, old_x, old_y

def choose_e(totient):
    # Escolhe um número aleatório que, 1 < e < totient, e verifica se é coprimo do totient,
    # que é quando gcd(e, totient) = 1
    while (True):
        e = random.randrange(2, totient)

        if (gcd(e, totient) == 1):
            return e
```

Criar chaves

Para criar uma chave é necessário dois números p e q, sendo eles primos e grandes.

Para encontrar números primos grande de forma mais rápida usamos o teste de Rabin Miller que é um teste probabilístico que indica um provável número primo.

In [99]:
```
import random
def rabinMiller(num):
    s = num - 1
    t = 0

    while s % 2 == 0:
        s = s // 2
        t += 1
    for trials in range(5):
        a = random.randrange(2, num - 1)
        v = pow(a, s, num)
        if v != 1:
            i = 0
            while v != (num - 1):
                if i == t - 1:
                    return False
                else:
                    i = i + 1
                    v = (v ** 2) % num
    return True
def isPrime(num):
    if (num < 2):
        return False
    lowPrimes = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61,
    67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151,
    157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241,
    251, 257, 263, 269, 271, 277, 281, 283, 293, 307, 311, 313,317, 331, 337, 347, 349,
    353, 359, 367, 373, 379, 383, 389, 397, 401, 409, 419, 421, 431, 433, 439, 443, 449,
    457, 461, 463, 467, 479, 487, 491, 499, 503, 509, 521, 523, 541, 547, 557, 563, 569,
    571, 577, 587, 593, 599, 601, 607, 613, 617, 619, 631, 641, 643, 647, 653, 659, 661,
    673, 677, 683, 691, 701, 709, 719, 727, 733, 739, 743, 751, 757, 761, 769, 773, 787,
    797, 809, 811, 821, 823, 827, 829, 839, 853, 857, 859, 863, 877, 881, 883, 887, 907,
    911, 919, 929, 937, 941, 947, 953, 967, 971, 977, 983, 991, 997]

    if num in lowPrimes:
        return True
    for prime in lowPrimes:
        if (num % prime == 0):
            return False
    return rabinMiller(num)

def generate_large_prime(keysize = 8):
    while True:
        num = random.randrange(2**(keysize-1), 2**(keysize))
        if isPrime(num):
            return num
```

Há duas formas de gerar os números primos, pegando um número aleatório que representa a linha do número primo no arquivo primo.txt e outra pelo método probabilístico. A primeira forma é melhor para testes, por ser mais rápida.

In [107]:
```
def choose_keys():

    # Escolhe 2 números entre 100 e 300
    # O número é usado para escolher uma linha dentro do arquivo de primos
    rand1 = random.randint(100, 300)
    rand2 = random.randint(100, 300)

    # Gera os primos
    fo = open('primes.txt', 'r')
    lines = fo.read().splitlines()
    fo.close()
#     prime1 = 1733
#     prime2 = 1847

    prime1 = int(lines[rand1])
    prime2 = int(lines[rand2])
#     prime1 = int(generate_large_prime())
#     prime2 = int(generate_large_prime())

    print('primes', prime1, prime2)


    # calcula n, totient, e
    n = prime1 * prime2
    totient = (prime1 - 1) * (prime2 - 1)
    e = choose_e(totient)

    # calcula d, 1 < d < totient such that ed = 1 (mod totient)
    # e and d are inverses (mod totient)
    gcd, x, y = xgcd(e, totient)

    # certifica-se de que é positivo
    if (x < 0):
        d = x + totient
    else:
        d = x

    # Guarda as chaves
    f_public = open('public_keys.txt', 'w')
    f_public.write(str(n) + '\n')
    f_public.write(str(e) + '\n')
    f_public.close()
    print('Chaves públicas:')
    print('n: ', str(n))
    print('e: ', str(e))

    f_private = open('private_keys.txt', 'w')
    f_private.write(str(n) + '\n')
    f_private.write(str(d) + '\n')
    f_private.close()
    print('Chaves privadas:')
    print('n: ', str(n))
    print('d: ', str(d))
```

Criptografando

A função encrypt criptografa uma mensagem trocando cada caractere pelo valor da tabela ascii. Assim retorna uma string de números. O tamanho do bloco é quantos caracteres tem em um grupo de números.

In [108]:
```
def encrypt(message, file_name = 'public_keys.txt', block_size = 2):

    try:
        fo = open(file_name, 'r')

    # Verifica a existência das chaves públicas
    except FileNotFoundError:
        print('That file is not found.')
    else:
        n = int(fo.readline())
        e = int(fo.readline())
        fo.close()

        encrypted_blocks = []
        ciphertext = -1

        if (len(message) > 0):
            # Primeiro caractere como ascii
            ciphertext = ord(message[0])

        for i in range(1, len(message)):
            # adiciona os caracteres na lista até o tamanho do bloco, depois reseta e continua
            if (i % block_size == 0):
                encrypted_blocks.append(ciphertext)
                ciphertext = 0

            # multiply by 1000 to shift the digits over to the left by 3 places
            # because ASCII codes are a max of 3 digits in decimal
            ciphertext = ciphertext * 1000 + ord(message[i])

        # add the last block to the list
        encrypted_blocks.append(ciphertext)

        # criptografa todos elevando a e
        # e tirando o resto da divisão por n
        for i in range(len(encrypted_blocks)):
            encrypted_blocks[i] = str((encrypted_blocks[i]**e) % n)

        # string de números
        encrypted_message = " ".join(encrypted_blocks)

        return encrypted_message
```

Descriptografa

Nesta parte é feito basicamente o professo inverso de criptografia.

In [109]:
```
def decrypt(blocks, block_size = 2):

    fo = open('private_keys.txt', 'r')
    n = int(fo.readline())
    d = int(fo.readline())
    fo.close()

    # converte numa lista de inteiros
    list_blocks = blocks.split(' ')
    int_blocks = []

    for s in list_blocks:
        int_blocks.append(int(s))

    message = ""

    # divide nos blocos
    for i in range(len(int_blocks)):
        # descriptografa
        int_blocks[i] = (int_blocks[i]**d) % n

        tmp = ""
        # troca os números pelos caracteres da ascii e monta string
        for c in range(block_size):
            tmp = chr(int_blocks[i] % 1000) + tmp
            int_blocks[i] //= 1000
        message += tmp

    return message
```

In [110]:
```
def main():
    choose_again = input('Deseja gerar novas chaves? (s ou n) ')
    if (choose_again == 's'):
        choose_keys()

    instruction = input('Deseja criptografar ou descriptografar? (Enter c ou d): ')
    if (instruction == 'c'):
        message = input('Digite a mensagem:\n')
        option = input('Deseja usar suas chaves públicas? (s ou n) ')

        if (option == 's'):
            print('Criptografando...')
            print(encrypt(message))
        else:
            file_option = input('Digite o nome do arquivo com as chaves públicas: ')
            print('Criptografando...')
            print(encrypt(message, file_option))

    elif (instruction == 'd'):
        message = input('O que gostaria de descriptografar?\n')
        print('Descriptografando...')
        print(decrypt(message))
    else:
        print('Instrução errada!')
```

In [111]:
```
main()
```

```
Deseja gerar novas chaves? (s ou n) s
primes 1361 1567
Chaves públicas:
n:  2132087
e:  1875533
Chaves privadas:
n:  2132087
d:  1043237
Deseja criptografar ou descriptografar? (Enter c ou d): c
Digite a mensagem:
Hello world
Deseja usar suas chaves públicas? (s ou n) s
Criptografando...
11459 1567919 286813 2103404 1363926 872196
```

In [112]:
```
main()
```

```
Deseja gerar novas chaves? (s ou n) n
Deseja criptografar ou descriptografar? (Enter c ou d): d
O que gostaria de descriptografar?
11459 1567919 286813 2103404 1363926 872196
Descriptografando...
Hello world
```

Referências

https://medium.com/@prudywish/how-to-generate-big-prime-numbers-miller-rabin-49e6e6af32fb

https://github.com/ichen2186/rsa-implementation

https://www.geeksforgeeks.org/euclidean-algorithms-basic-and-extended/

In [ ]:
```

```