rsa-calc

February 13, 2022

1 Praktische implementatie RSA met python

Voor RSA private en public key encryption zijn een aantal zaken benodigd:

- Twee priemgetallen. We kiezen hier in dit voorbeeld voor $'p = 17$ *en* $'q = 23$
- Een variabele 'n'. Deze is gelijk aan $n{=}p \times q$
- Een variabele 'z'. Deze is gelijk aan *z=(p - 1)(q - 1)*

We lopen door de code heen met uitleg tussen de verschillende blokken.

Er worden verschillende modules gebruikt:

- **gcd**: Een module om relatieve priemgetallen te kunnen vaststellen
- **os**: gebruiken we hier om een clear screen te kunnen uitvoeren.
- **random**: Uit deze module gebruiken we randrange om een random waarde uit een reeks te kunnen kiezen.
- **colorama**: Hiermee kunnen we verschillende kleuren gebruiken in de output.

[1]:
```
from math import gcd as gcd
from os import system, name
from random import randrange
from colorama import Fore, Style
```

[2]:
```
#Prime keys are given:
p = 17
q = 23
string_value = ''

#calculate neccessary variables for RSA:
n = p * q
z = (p-1)*(q-1)
```

Nu definiëren we twee functies die een public key en een private key waarde zoeken. De public key waarde gebruiken we hier voor encryptie (e). Dit is een waarde die een relatief priemgetal moet zijn van z en moet kleiner zijn dan n. Een relatief priemgetal deelt met z geen gemeenschappelijke factoren anders dan de deelfactor 1.

In Python gebruiken we hiervoor de module **gcd**. Wat die doet is de ggd (grootste gemeenschappelijke deler) berekenen van de waarden e en z. Enkel wanneer de ggd de waarde 1 heeft zijn de twee getallen copriem. De manier waarop dat werkt is dat er telkents een modulo berekening wordt

toegepast op de 2 waarden. Als de modulo op 0 is uitgekomen en de waarde dan 1 is weten we zeker dat de getallen copriem zijn.

De berekening gaat in python als volgt: $a,\ b = b,\ a\ mod\ b$

Naast de module **gcd** gebruiken we **randrange**. Met randrange kun je een random nummer kiezen in een bepaalde range. Omdat $e < n$ kiezen we randrange(n).

De tweede functie kiest de private key waarde die we gebruiken voor decryptie (d). De regels voor hiervoor zijn dat $(e \times d)$-1 deelbaar moet zijn door z zonder restwaarde. Oftewel $(e \times d)$ -1) mod $z = 0$. Daarnaast moet $e \times d > z$.

We gebruiken ook hier weer **randrange** maar dan met een minimale waarde namelijk z/e afgerond op een heel getal. Als maximum waarde in de range gebruiken we in dit voorbeeld 500. Het script kiest in een while loop telkens een waarde en berekent daarover de formule zoals hierboven omschreven.

[3]:
```
def calcPublicKey(n):

    #randrange kan een random nummer uit een range halen. Bijvoorbeeld
    #randrange(10)
    #haalt een waarde tussen 0 en 9.
    e = randrange(n)

    #De while zoekt net zolang naar een integer waarde die een copriem is
    #van z. Als dat zo is dan stopt de loop.
    while gcd(e,z) != 1:
        e = randrange(n)

    print('public key is {},{}'.format(n,e))
    return(e)
```

[4]:
```
def calcPrivateKey(e,z):
    ''' Berekenen van de private key.
    d moet een waarde zijn waarbij geldt dat (e*d) - 1 deelbaar moet zijn door
    z en waarbij er geen restwaarde is. Oftewel de waarde e*d -1 mod z is 0
    Om d te bepalen moet e*d in ieder geval al groter zijn dan z. De minimale
    waarde van d is z/e afgerond op een heel getal. '''

    d  = randrange(round(z/e),500)
    while (((e*d)-1)%z != 0):
        d  = randrange(round(z/e),500)

    print('private key is {},{}'.format(n,d))
    return(d)
```

Hieronder volgt de main van het programma met een clear screen voor linux en windows OS

[5]:
```
if __name__ == "__main__":
    #clear the screen. Different commands depending on OS.
```

```
    if name == 'posix':
        _ = system('clear')
    else:
        _ = system('cls')
```

Vanuit main worden calls gedaan op de functies van hierboven om de public en private keys te maken. We gebruiken ook nog de module **colorama** om met verschillende kleuren te kunnen werken. Als laatste vraagt input om een string om te kunnen encrypten en decrypten. Alle letters in de string worden omgezet naar ascii en 1 voor 1 geplaatst in een list *ascii_values*.

[6]:
```
    #calculate the public and private keys
    e = calcPublicKey(n)
    d = calcPrivateKey(e,z)
    text = input(Fore.GREEN + "Voer een woord in om te encrypten: ")
    ascii_values = []
    for character in text:
        ascii_values.append(ord(character))
```

```
public key is 391,321
private key is 391,193
Voer een woord in om te encrypten: Dit is de tekst
```

Volgende stap is om de ingegeven string daadwerkelijk te encrypten met de public key (e) en te decrypten met de private key (d) om aan te tonen dat het werkt. Encryptie gebeurt als volgt: *((asciiwaarde^e) mod n)*. Decryptie gaat als volgt: *((cipherwaarde^d) mod n)*.

[7]:
```
    #Encryption
    cipher_values=[]
    for ascii in ascii_values:
        cipher_values.append(pow(ascii, e, n))
    print(f'{Style.RESET_ALL}encrypt met public key {n},{e} geeft: {Fore.
↪RED}{cipher_values}{Style.RESET_ALL}')
    #decryption
    decrypted_values = []
    for cipher in cipher_values:
        decrypted_values.append(pow(cipher, d, n))
```

```
encrypt met public key 391,321 geeft: [68, 54, 116, 219, 54, 115, 219,
202, 288, 219, 116, 288, 5, 115, 116]
```

Als laatste herstellen we de string door een loop te maken door de list met originele ascii karakters en deze weer om te zetten naar characters. Deze drukken we af op het scherm.

[8]:
```
    #write to character
    for character in decrypted_values:
        string_value = '{}{}'.format(string_value, chr(character))
```

```
    print(f'decrypt met private key {n},{d} geeft: {Fore.
↪GREEN}{string_value}{Style.RESET_ALL}')
```

```
decrypt met private key 391,193 geeft: Dit is de tekst
```