RFMA310-1 20H Diskret matematikk Hjemmeeksamen2020

8018

8 Desember 2020

1 Introduction

Denne besvarelsen starter med Euler og sykliske grupper, deretter forklares teorien bak ElGamal med Diffie Hellmann nøkkelutveksling. Innholdet er basert på forelesninger, heftet i abstrakt algebra[1], oppgaveteksten og den anbefalte Wikepediasiden[2].

2 euler

Gitt mengden:

$$A = \{a_1, a_2, ..., a_{\phi(n)}\} = \{1 \leq a < n \mid gcd(a, n) = 1\}.$$

Da er mengden

$$B = \{aa_n, aa_2, ..., aa_{\phi(n)}\}$$

kongruent med A.

$$aa_i \equiv b_i (mod\, n),\;\; b_1 \in \{a_1, a_2, ..., a_{\phi(n)}\}$$

Hver gang man multipliserer et element $a_0 \in A$ med et annet element $a_1 \in A$, vil man treffe et element som er i A. fordi:

$$gcd(a, n) = gcd(b, n) = 1 \Leftrightarrow gcd(ab, n) = 1$$

Det er selvforklarende at ab kan primtalfaktoriseres tilbake til a og b, som er relative primtall med n. Derfor må også ab være relativ primtall med n, og er med i mengden: $gcd(ab, n) = 1$.

Euler teorem sier at A skal være kongruent med B. Derfor må multiplikasjonen $a \cdot A$, a∈ A, lage en permutert syklisk gruppe B, som er bijektiv med A og har de samme elementene.

Motsigelsesbevis: dersom A & B ikke er kongruent; må elementene $a_j, a_i \in A$ treffe det samme elementet a_k når $a \in A$ multipliseres med A, ellers ekskluderes

ikke noen elementer, og kongruensen er gjeldene.
$aa_i \equiv aa_j \, mod \, n$, kan ikke være sant, hvis i $\neq$ i, er dette opplagt.

Hvis man har A $\equiv$ B og deler med $\{a_1, a_2, ..., a_{\phi(n)}\}$ på begge sider, for man følgende:

$$a^{\phi(n)} \equiv 1 \, mod \, n$$

Dersom man multipliserer to elementer a_1 og a_2 $\in$ A, og resulatet blir større en n, vil det etter moduloperasjonen fortsatt treffe et element c $\in$ A.

VIKTIG: Forholdet mellom A og B er **isomorfisk**. A og B er injektive, og består av eksakt samme elementer bare permutert. Den mengdeteoretiske funksjonen ϕ fra heftet i abstrakt algebra, er i dette tilfelle fra A til B.

$$\phi : \, A \, \rightarrow \, B$$

3 sykliske grupper

En gruppe G er syklisk hvis det fins et element $g \in G$, slik at hvert element $a \in G$ kan genereres på måten:

$$a = g^n = g_0 \otimes g_1 \otimes ... \otimes g_{n-1}$$

Alle sykliske grupper er abelske, derfor gjelder følgende regler:

$$g^{x^y} = g^{y^c} = g^{x \cdot y} = k$$

Pga. den kommutative loven, er regnerekkefølgen likegyldig ved utregningen av K. Det sistnevnte er svært viktig.

Den sykliske gruppen A, der alle elementene tilfredsstiller kravet:

$$A = \{a_1, a_2, ..., a_{\phi(n)}\} = \{1 \leq a < n \mid gcd(a, n) = 1\}$$

kan bruke alle elementer som generator. Et positivt heltall som opphøyer g, vil altid tilordne et element i G. ElGamal bygger på den sykliske gruppen

$$G = (Z/p)^X, \, p \in Spec(Z)$$

Da vil elementene i G være relativ primtall med p, dette er selvforklarende. Gruppen følger de matematiske reglene jeg har beskrevet hittil.

4 Key generation

For å forklare algoritmen, brukes det klassiske datasikkerhetseksempelet der Alice og Bob skal kommunisere, mens Eve prøver å få tilgang til den krypterte samtalen. Alice lager en syklisk gruppe G, den inneholder q elementer og en generator. Deretter regner Alice ut $h = g^x$, x $\in$ G. H blir en del av public key, mens x er Alice sin private nøkkel. Alice sender ut en public key til Bob med(G, q, h, g).

5 encryption bob

Bob skal sende en melding til Alice med public key. Først må bob bestemme seg for en privat nøkkel y $\in$ G. Deretter regner Bob ut en delt hemmelighet $s = h^y$ (fra tidligere $g^x = h, h^y = g^{x \cdot y}$). Den delte hemmeligheten kan Bob og Alice regne seg frem til fordi; sykliske grupper er abelske:

$$g^{x^y} = g^{y^x} = g^{x \cdot y}$$

Bob sender avsted:

1. $c_1 = g^y$ som alice trenger for å regne ut den delte hemmeligheten s
2. $c_2 = m \cdot s$ Det er meldingen m, som krypteres den delte hemmeligheten s.

Å dekryptere meldingen tar for lang tid på grunn av det diskre logaritme problemet. Eve må finne y, gitt at hun vet c_1 og g. Dette tar lang tid om den sykliske gruppen og g er valgt med omhu.

6 decryption

Alice mottar c_1 og c_2. Her er det slik at c_1^x blir til den delte hemmeligheten, fordi: $g^{x^y} = g^{y^c} = g^{x \cdot y}$, som nevnt tidligere. Begge ender opp med den delte hemmeligheten s. Alice dekrypterer meldingen med en invers av den delte nøkkelen $m = c_2 \cdot s^{-1}$. Den multiplikative invers til s kan regnes ut med extended eucledian algoritme. En annen metode benytter følgende sammenheng fra langranges teorem:

$$s \cdot c_1^{q-x} = g^{xy} \cdot g^{(q-x)y} = (g^q)^y = e^y = e$$

når vi ganger s med c_1^{q-x} får vi enhetselementet, det betyr at c_1^{q-x} er den multiplikative invers til s.

$$m = m \cdot s \cdot s^{-1}$$

7 Kommentarer på koden

For at Encrypt og Decrypt funksjonen skal fungere, må meldingen være i en liste med char, deretter multipliseres hvert enkelt element med hele keyen. Valget av datastruktur og koden for å legge char inn i listen, har jeg lånt fra geeksforgeeks[3].

Som nevnt i oppgaveteksten kan en vilkårlig syklisk gruppe brukes, men det brukes sjeldent. Det er ikke lagt inn noe funksjon i programmet som finner primtall. Figur 1 viser chipertext, og den dekrypterte meldingen.

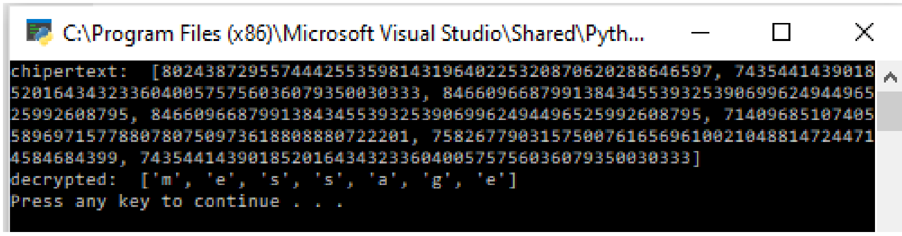

Figure 1: Bilde av det som printes av koden

8 Kilder

[1]M. D. Larsson. (2020). Abstraktalgebra [pdf]. Available: https://usn.instructure.com/courses/22198/files?
[2]"ElGamal encryption." Wikipedia. Web Adress: https://en.wikipedia.org/wiki/ElGamal_encryption (accessed Des. 5,2020).
[3]"ElGamal Encryption Algorithm." Geeksforgeeks. Web Adress: https://www.geeksforgeeks.org/elgamal-encryption-algorithm/ (accessed Des. 5,2020).


```python
from math import pow
import random

def gcd(a, b):
        while b > 0:
                r = a % b
                a = b
                b = r
        return a


def element_in_q(q):
        element = random.randint(pow(10,20), q)
        while gcd( q, element) != 1:
                element = random.randint(pow(10,20), q)
        return element

def power(_base_, exponent , modulo):
        x = 1                                # variablen returverdien lagres i
        base = _base_
        while exponent > 0:
                if exponent % 2 == 0:        # Ved partall ganges x med base
                        x = (x*base) % modulo
                base = (base*base)% modulo   #basen ganges med seg selv for
                exponent = int(exponent / 2) #aa effektivisere koden
                                             # Maa redusere eksopnenten
```

```
                                                # logaritmisk pga effektiviseringen
        return x % modulo

def encrypt(plainText, q, h, g):

        c2 = []                                  #er letter å jobbe med liste
        for i in range(0 , len(plainText)):
                c2.append(plainText[i])          #legger meldingen inn i en list med char

                #Implementasjonen med list kommer fra:
                #https://www.geeksforgeeks.org/elgamal-encryption-algorithm/
                #se seksjon 6


        y = element_in_q(q)
        s = power(h,y,q)
        c1 = power(g, y, q)

        for i in range(0, len(plainText)):
                c2[i] =  ord(c2[i]) * s          #kryptering


        return c2, c1 # lag p

def decrypt(cipherText, c1 ,x , q):
        decryptedText = []
        h = power(c1, x, q)


        for i in range(0, len(cipherText)):  #ref: seksjon 6
                decryptedText.append(chr(int(cipherText[i]/h)))

        return decryptedText




plainText = "message"
q = random.randint(pow(10,10), pow(10, 50))
g = element_in_q(q)
#eulers theorem, A = {a0...an} er kongruent med B = a*{a0...an}

x = element_in_q(q)     #X er privat nøkkel
h = power(g, x, q)      # h er public key

cipherText, c1 = encrypt(plainText, q, h, g)
```

```python
clearText = decrypt(cipherText, c1, x,q)

print("chipertext: ", cipherText)
print("decrypted: ", clearText)
```