Python

Diffie-Hellman nøkkelutveksling

Skrevet av: Martin Strand

Kurs: Python
Tema: Tekstbasert, Kryptografi
Fag: Matematikk, Programmering
Klassetrinn: 8.-10. klasse, Videregående skole

Introduksjon

Du har tidligere jobbet med Cæsar-chifferet, en oppgave der du skulle lage enkel kryptering (https://oppgaver.kidsakoder.no/python/hemmelige_koder/hemmelige_koder). Men, den typen kryptering ("symmetrisk kryptografi") har et iboende problem: Før man kan kryptere og sende en melding, må man være enig om hvilken nøkkel man skal bruke. Med andre ord, for å kunne snakke sammen hemmelig, må man først bli enig om en hemmelighet -- og da er man jo like langt?

En løsning på dette problemet er Diffie-Hellman nøkkelutveksling. Ideen kom fra Whitfield Diffie og Martin Hellman i 1976, og er fortsatt i bruk i dag. I denne leksjonen skal vi programmere ideen deres. Til slutt vil du kunne rope til en som sitter på motsatt ende av rommet, og dere blir enige om en felles hemmelighet uten at noen andre i rommet forstår hva det er.

De to viktigste byggeklossene er å ha tilgang til primtall (../primtall/primtall.html), og å trekke tilfeldige tall på en god måte (../tilfeldige_tall/tilfeldige_tall.html). Dersom du ikke husker det vi gjorde i de leksjonene, kan det være lurt å gå tilbake og se over dem på nytt. Spesielt bør du sjekke at du husker hva hva `%`-operatoren gjør.

Steg 1: Regneoperasjoner på en klokke

Før vi kan gå i gang for alvor, må vi introdusere en ny måte å regne på. De fleste kan den fra før, men uten å være klar over det. Tenk deg at klokka er 11.00. Dersom noen spør deg om hva klokka vil være om fire timer, vil den være 15:00, men i dagligtale vil man kanskje bare si at den er "tre". Med andre ord er plutselig 11 + 4 = 3. Det man gjør -- helt ubevisst -- er å alltid trekke fra 12 helt til man kommer til et tall mellom 0 og 12. Vi kan definere regneoperasjoner på akkurat samme måte, og det har Python innebygd. I dette eksempelet kalles 12 *modulusen*.

Åpne IDLE, men uten å lage en ny fil. Der det står `>>>` kan vi skrive inn koda for å se

Åpne IDLE, men uten å lage en ny fil. Der det står `>>>` kan vi skrive inn kode for å se hva den gjør.

Sjekkliste

- [ ] Skriv `(11 + 4) % 12` og trykk Enter. Du skal da få svaret 3. Om du sammenligner med det vi gjorde i primtallsleksjonen, ser du kanskje at 15 delt på 12 gir 3 til rest.

- [ ] Det fungerer også med multiplikasjon. Skriv `(5 * 4) % 12` og trykk Enter. Da skal du få 8, siden $5 \cdot 4 = 20 = 12 + 8$.

- [ ] Prøv deg fram med litt flere eksempler selv. Prøv å forstå hva du skal få før du faktisk ber Python regne det ut for deg.

- [ ] Regn ut `(3 * 4) % 12`. Hva får du?

- [ ] Vi kan også regne ut potenser. I matematikkboka skrives det som 4^3, mens tilsvarende Python-syntaks er `4**3`. Bruk IDLE til å regne ut `4**3 % 12` og `6**2 % 12`.

Steg 2: Finne en nøkkel for Cæsar-chifferet

I denne aktiviteten er det ment at du skal samarbeide med noen andre, så vi skal strukturere koden vår i funksjoner som du kan bruke fra IDLE-konsollen. Etter hvert skal vi utvide koden vår slik at den i prinsippet kan brukes til alle formål.

Sjekkliste

- [ ] Åpne en ny fil i IDLE, og skriv inn den følgende koden

```
p = 23
g = 5

def generate_share(a, g, p):
    return g**a % p

def generate_secret(other_share, a, p):
    return other_share**a % p
```

- [ ] Lagre filen og kjør modulen ved å trykke F5. Du får da opp et Python-skall der du kan skrive kommandoer selv. Finn på et tilfeldig tall mellom 1 og 23, og skriv `generate_share(<ditt tall>, g, p)`. Da får du ut et nytt tall. Dette er det offentlige tallet du skal si høyt til vennen til på motsatt side av rommet. Han eller hun skal gjøre det samme.

- [ ] Skriv `generate_secret(<tallet vennen din ropte>, <ditt tall>, p)`, og trykk Enter. Det tallet du får ut nå er den hemmelige nøkkelen som du og vennen din deler, men som ingen andre kan vite om.

- [ ] Finn fram Cæsar-chifferet, og bruk den hemmelige nøkkelen som `secret`. Send en melding til vennen din, og se om han eller hun får til å dekryptere korrekt.

Dersom du jobber alene, kan du teste dette ved å kjøre `generate_share` og `generate_secret` to ganger. Da må du passe på å bruke riktig tall på riktig plass.

Steg 3: Bedre tilfeldighet

I forrige steg ble du bedt om å skrive inn et tilfeldig tall. Mennesker er ikke flinke til å finne på tilfeldige tall, så det er på tide å dra inn noe av det vi har lært fra før. Se over Tilfeldige tall (../tilfeldige_tall/tilfeldige_tall.html) på nytt dersom du har glemt hva vi gjorde der.

Sjekkliste

- [ ] Endre koden din slik at den nå ser slik ut.

```
from secrets import randbelow

p = 23
g = 5

def generate_share(g, p):
    a = randbelow(p)
    return (a, g**a % p)

def generate_secret(other_share, a, p):
    return other_share**a % p
```

Les koden, og forsøk å forstå hva den gjør nå. Kjør koden når du er klar.

Steg 4: Angrep mot koden vår

Dersom du er av den nysgjerrige typen, har du kanskje hørt hva de andre i rommet har sagt til hverandre, og kanskje har du lyst til å få tak i deres hemmelige nøkkel. Vi kan gjøre det ved å angripe `generate_share` -funksjonen vi har laget over. Siden `p` er så liten, kan man lage en rutine som prøver alle tall mindre enn `p` , og sjekker om man kan få den samme verdien som ble ropt ut.

Sjekkliste

- [ ] Implementer funksjonen vi beskrev over.

```
def find_secret(share, g, p)
    # din kode her
    return a
```

- [ ] Du har nå funnet noens hemmelige tall. Den personen hadde en partner. Bruk tallet som partneren sa høyt som `other_share` i `generate_secret` , og finn deres delte hemmelighet ved hjelp av det hemmelige tallet du fant.

Steg 5: Bruke større parametre

I koden over har vi hardkoded `p = 23`, og vi hadde ikke noe problem med å knekke systemet, man kunne til og med gjort det for hånd med bittelitt tålmodighet. Nå skal vi se hvordan vi kan bruke det vi har lært om å finne primtall til å gjøre dette litt bedre.

Sjekkliste

- [ ] Finn fram koden din fra Primtall og effektivitet (../primtall/primtall.html), og legg den til å fila vi jobber med. Den skal nå se slik ut

```
from math import sqrt, ceil from secrets
import randbelow

def is_prime(n):
    if not (isinstance(n, int) and n > 1):
        return False
    small_primes = [2, 3, 5, 7, 11, 13, 17, 19]
    if n in small_primes:
        return True
    for prime in small_primes:
        if n % prime == 0:
            return False
    for i in range(20, ceil(sqrt(n)) + 1, 2):
        if is_prime(i):
            if n % i == 0:
                return False
    return True

p = 23
g = 5

def generate_share(g, p):
    a = randbelow(p)
    return (a, g**a % p)

def generate_secret(other_share, a, p):
    return other_share**a % p
```

- [ ] Vi trenger en funksjon som lar oss finne et primtall som er større enn en grense vi bestemmer. Den enkle måten å gjøre det på er å starte fra minimum, telle opp én og én, og se om det nye tallet er et primtall. Legg til det følgende i koden din:

```
def find_prime(start):
    n = start
    while not is_prime(n):
        n = n + 1
    return n
```

Legg merke til at `start` ikke trenger å være et primtall, det er ganske enkelt omtrent så stort som du vil at primtallet ditt skal være.

- [ ] Slett linjen `p = 23`. Kjør programmet, og skriv inn `p = find_prime(<velg et tall her>)`. Primtallet `p` er ikke hemmelig, og du og partneren din må bruke samme `p` for at koden skal virke.

- [ ] **Utfordring:** Tallet `g = 5` er fortsatt hardkodet. Egentlig må `g` velges etter primtallet, og det må være en såkalt *primitiv rot modulo p* (eng: *primtive root modulo p*). Bruk Google og Wikipedia for å finne en algoritme for å beregne slike. Pass på at du forstår hva koden din gjør.

Litt fra den virkelige verdenen

Hver gang du går inn på en nettside med en hengelås i adressefeltet ditt, så har nettleseren din brukt de samme algoritmene som over for å bli enig med nettsiden om hemmelige nøkler for å holde informasjonen din sikker.

Advarsel

Koden vi har skrevet nå er en fin introduksjon til disse konseptene. Du bør likevel aldri bruke den til noe som er ment å være sikkert på ordentlig. Med mindre man har lang utdanning eller erfaring innen programmering av sikker kode, må man alltid anta at det er utallige sikkerhetsproblemer med koden. Det er det også her, i tillegg til at koden ikke er praktisk nok til å kunne brukes til så store tall som man gjør i ekte anvendelser.