Python

Hash-funksjoner

Skrevet av: Martin Strand

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

Introduksjon

Tidligere har vi sett hvordan vi kan sikre at ingen kan lese meldingene vi sender fram og tilbake. Kryptering har også et annet viktig formål, nemlig at ingen skal kunne *endre* meldingen mens den er på vei til deg. For å forstå hvorfor det er nyttig, tenk deg at du skal betale 200 kr til en venn. Men, når beskjeden din når fram til nettbanken, har den blitt endret til at du vil betale 2 000 kr til læreren din i stedet. Dagens leksjon handler om et verktøy vi har for å hjelpe oss med dette, *hash-funksjoner*.

Steg 1: Strekkoder

Vi starter med et eksempel som brukers hver gang vi er i butikken, nemlig strekkoden EAN-13. Finn en bok i nærheten og se på baksiden. Der er det som oftest en strekkode med med 13 sifre under. Det siste sifferet sin jobb er å sjekke om det har skjedd en feil med de 12 andre. Dersom det har skjedd en feil må den som sitter i kassa scanne varen på nytt, men det er langt bedre enn at det blir slått inn en pakke fløte når det egentlig skulle vært melk. Vi skal nå bygge et program som gjør det samme som kassasystemet på nærbutikken din.

Regelen er enkel nok: Start med 12 sifre, og del dem opp slik at du jobber med ett og ett. Legg sammen det andre, fjerde, osv. sifferet og multipliser summen med 3. Så legger du sammen første, tredje, femte, osv. siffer og legger til det du har fra før. Se på tallet du nå har fått, og finn ut hvor mye du må legge til for å komme opp til nærmeste tier oppover. Det er sjekksifferet ditt.

Husk at når vi jobber med en liste i Python, så er *den første* på plass 0, så når det står `digits[0]`, så er det det første tallet, og `digits[1]` er det andre tallet -- og så videre.

Eksempel

Jeg har valgt en tilfeldig bok fra bokhylla mi. EAN-13-koden er 9780062731029.

La oss sjekke at den er gyldig.

Først regner vi ut 3 · (7 + 0 + 6 + 7 + 1 + 2) = 69 og 9 + 8 + 0 + 2 + 3 + 0 = 22. 69 + 22 = 91. Den nærmeste tieren oppover er 100, så da er sjekksifferet 9 -- akkurat som det skal være.

Sjekkliste

- [ ] Åpne IDLE og lag en ny fil. Lim inn den følgende koden:

```
def verify_barcode(digits):

    # Regn ut i henhold til formelen
    summert = 3 * (digits[1] + digits[3] + digits[5] + digits[7]
               + digits[9] + digits[11])
    print(summert)
    summert = summert + (digits[0] + digits[2] + digits[4] + digit
s[6]
                   + digits[8] + digits[10])

    # Finn differansen mellom 10 og det siste sifferet i summen
    # Sjekksifferet kan ikke være 10, så den siste % 10 gjør at de
t
    # blir 0 i stedet.
    checksum = (10 - (summert % 10)) % 10

    if checksum == digits[12]:
        return True
    else:
        return False
```

- [ ] Lagre og kjør koden med F5. I konsollen, test koden din med følgende kode: `verify_barcode([9,7,8,0,0,6,2,7,3,1,0,2,9])`.

- [ ] Test koden din med andre EAN-koder. Se hva som skjer dersom du skriver et galt siffer, eller bytter om rekkefølgen på to.

- [ ] Det er litt slitsomt å skrive inn EAN-koden siffer for siffer. Legg til den følgende koden i fila di:

```
def interpret_barcode(barcode):

  # Vi sjekker først at vi faktisk har fått nøyaktig 13 sifre
  if (len(barcode) != 13) or not (barcode.isdigit()):
      return None

  # Oversett teksten til en liste av sifre
  digits = [int(d) for d in barcode]

  return digits
```

Lagre filen. Du kan nå teste den ved å skrive
`verify_barcode(interpret_barcode("9780062731029"))`

- [ ] Sjekk disse strekkodene: 9788205260429 og 978098146739. Hvis noen av sjekksifrene var gale, er du i stand til å finne riktig sjekksiffer?

Steg 2: Sterkere koder

Sjekksifferet i EAN-13 er godt nok så lenge det handler om å fange opp de feilene som gjøres ved et uhell, men det er ikke spesielt vanskelig å finne to strekkoder som har samme sjekksiffer -- for eksempel holder det at første og tredje siffer bytter plass. For en god hash-funksjon må det (blant annet) være vanskelig å finne to input som gir samme output.

Vi skal ikke prøve å programmere våre egne hash-funksjoner, for de er heldigvis innebygd i Python. Alt vi skal gjøre er å lage et lite skall rundt de innebygde funksjonene, slik at de blir enklere å bruke for oss.

Hash-funksjonen vi skal bruke nå er god nok til å brukes i så godt som alle situasjoner der man vil ha sterk sikkerhet.

Sjekkliste

- [ ] Lim inn den følgende koden i IDLE

```
from hashlib import sha256

 def hash(data):
     h = sha256()
     h.update(data.encode())
     return h.hexdigest()
```

- [ ] Lagre filen og kjør den. I konsollen kan du teste hash-funksjonen ved å skrive `hash("978006273102")`. Se hva som skjer dersom du bare endrer tallet bittelitt.

Steg 3: Sikring av Cæsar-chifferet

Tidligere har du programmert Cæsar-chifferet. Men, en av de mange svakhetene der er at hvis du sender meldingen med et bud, så kan budet bare legge inn en helt ny melding. Antageligvis blir det helt uforståelig når det dekrypteres, men det er ingen måte mottakeren kan avgjøre om det er du som har gjort en feil, eller om det er budet som har endret på meldingen. Det kan vi nå gjøre noe med.

I koden til Cæsar-chifferet er det en variabel som heter `secrets`. Den skal vi bruke her også. I koden bruker vi forkortelsen *mac*, den står for *message authentication code*.

- [ ] Lag en ny fil i IDLE
- [ ] Legg til de følgende funksjonene i fila di

```
secret = 17
message = "yvååc kcfåu"

def mac(key, message):
    return (message, hash(str(key) + message))

def verify_mac(key, message, mac):
    if mac == hash(str(key) + message):
        return "Alt ok"
    else:
        return "Noen har tuklet med meldingen!"
```

Lagre og kjør.

- [ ] Prøv `mac(secret, message)`. Det du skal sende til mottakeren nå er ikke bare `yvååc kcfåu`, men `('yvååc kcfåu',`
- `'b5a1c943bf39423f55a6fa4e43a7642feddd371c355ec4560b2f360a2634d244')`.

- [ ] Mottakeren (eller du selv) skriver følgende: `verify_mac(secret, 'yvåac kcfåu', 'b5a1c943bf39423f55a6fa4e43a7642feddd371c355ec4560b2f360a2634d244')`. Det skal da komme en beskjed om at alt er i OK. Prøv å endre på enten meldingen
- eller hash-verdien. Hva får du da beskjed om?
- [ ] Sett sammen denne koden med det du har gjort i Cæsar-chifferet og lag en ny funksjon `encrypt_and_mac` som både krypterer og legger til en MAC som over. (Hint: I Python kan du returnere flere verdier ved å skrive `return (value1, value2)`.)

Så lenge nøkkelen er hemmelig er det ingen måte å endre meldingen på uten at det blir fanget opp. Det vi har gjort nå minner litt om det som gjøres i program som brukes til viktige oppgaver. Forskjellen er så klart at de bruker en mye større nøkkel, og i tillegg at nøkkelen inkluderes på en litt smartere måte. Vi skal ikke prøve å forstå hvorfor her, det kommer først på universitetet.