*Universitetet i Oslo - IN2120 Informasjonssikkerhet – 2022 – midtsemestereksamen*

A silver coin with a person on it

Description automatically generated with low confidence

Historien og Utviklingen av Kryptografi – fra Steintavler til Kvantekryptering

T397

Kryptografi (<Tematittel>?)

Kandidatnummer:   
15193   
15169   
15327

Antall ord: <antall ord>

Innhold

1. Innledning
2. Historisk Kryptografi

Så lenge vi har hatt evnen til å dele informasjon, særlig språklig, har vi hatt nødvendigheten til å skjule slik informasjon. Det er her oppfinnelsen av kryptografi kommer inn. Ordet *kryptografi* kommer opprinnelig fra Hellas og betyr *hemmelig melding*. De tidligste eksemplene på kryptografi baserte seg gjerne på mye mer trivielle algoritmer sammenlignet med det vi har i dag. Disse var gjerne transposisjons- og substitusjonschiffere. I dette delkapittelet skal vi undersøke ulike implementasjoner og kombinasjoner av disse chifferne, hva de ble brukt til og til slutt hvordan man best kan anvende dem, gjerne gjennom kombinasjon, for å lage ganske robuste krypteringsalgoritmer.

// revider introduksjon etter kapittelet er ferdig @filip 😊

1. Substitusjonschiffer

Substitusjonschiffere virker ved å bytte ut hvert enkelt tegn i en melding med en fast erstatning. Hvordan man definerer hva man skal erstatte hvert tegn med kan gjøres på et par forskjellige måter. For eksempel kan du ha en forhåndsbestemt tabell med verdier der du setter en bestemt verdi til hver bokstav. Verdiene i chifferteksten her trenger ikke nødvendigvis å være bokstaver, du kan for eksempel bruke tall, tegnsetting, mellomrom, figurer/tegninger for å gjøre algoritmen mer robust. Men antageligvis den mest kjente metoden er å bruke et forskjøvet alfabet slik som i Cæsar-chiffer. I denne algoritmen tar man den originale meldingen og forskyver hver bokstav X antall plasser rundt i alfabetet. For eksempel hvis man velger +2 plasser blir hver forekomst av A til C, B til D, C til E, osv.

A picture containing graphical user interface

Description automatically generatedDiagram

Description automatically generatedI forhold til historisk bruk er opprinnelsen til Cæsar-chiffer kreditert til dens navn navnebror Julius Cæsar. Algoritmen ble brukt for å beskytte militære hemmeligheter og skal historisk ha blitt brukt med et skift på 3 bokstaver, dvs. at A blir til D ved kryptering og D til A ved dekryptering. Figur 1 viser hvordan alfabetet hadde blitt dekryptert ved Cæsar tradisjonelle bruk av chifferet. Selv om andre former for substitusjonschiffere skal ha vært i bruk før Cæsar-chiffer, er dette den første historiske registrerte, hvilket gjør den spesiell for historien av kryptografi. I virkeligheten er mye av detaljene rundt algoritmen tapt, så mesteparten av det vi vet nå kommer fra senere historikeres beskrivelser, spesielt Suetonius (The cipher of Julius Caesar, 2021).

Figur 2 viser en chiffer disk brukt til å både kryptere og dekryptere slike meldinger der chiffer disken er instilt til å forskyve alfabetet 13 plasser. Disken ble utviklet i 1470 av Leon Battista Alberti, mannen som også lagde det første polyalfabetiske chifferet, Alberti chiffer (<https://en.wikipedia.org/wiki/Cipher_disk>). Dette ble senere utvidet av Blaise de Vigenere til Vigenere chiffer. Med denne chiffer disken kan du rotere den indre ringen etter håndtaket fram til du har riktig skift, så kan du sammenligne indre og ytre ring for å enten oversette klartekst til chiffertekst (kryptering), eller chiffertekst til klartekst (dekryptering). Dette gjøres bokstav for bokstav i en melding og kan dermed ta ganske lang tid, i tillegg til at det kan forekomme feil ettersom det ikke er en automatisk prosess.

Logo, company name

Description automatically generatedChiffer diskens rotasjon på 13 plasser er i likhet med ROT13 som er en type substitusjonschiffer der man velger å forskyve alfabetet med 13 plasser i motsetning til Cæsar-chiffers tradisjonelle 3 plasser. Siden det er 26 bokstaver i det latinske alfabetet, vil du ved å velge en rotasjon på 13 bokstaver ha egenskapen av å kunne bruke samme algoritme både for kryptering og dekryptering. Det vil si at hvis du for eksempel krypterer «A» og får «N» vil du kunne bruke samme algortime for å dekryptere «N» tilbake til «A». Dette er fordi når du skifter en bokstav 13 plasser 2 ganger ender man opp med den samme bokstaven igjen. ROT13 algoritmen for både kryptering og dekryptering kan fremstilles med følgende formel:

(x+13) mod 26x+13 mod 26

Her er x indeksen til bokstaven i alfabetet (A = 1, B = 2, C = 3, osv.) som vi plusser med 13 for å skifte den 13 til høyre. Til slutt tar vi modulo 26 av dette for å rulle tilbake til starten av alfabetet dersom addisjonen tar oss utenfor alfabetet (x + 13 > 26). Vi kan på en måte se på addisjonen som krypteringen og modulo operasjonen som dekrypteringen. Med det sagt var ROT13 chifferet utviklet av programmerere på 1980-tallet så det er antageligvis bare en tilfeldighet at chiffer disken fra 1400-tallet er innstilt til en rotasjon på 13 plasser ettersom brukere av ROT13 antageligvis ikke hadde bruk for slik primitiv teknologi.

1. Transposisjonschiffer

Transposisjonschiffere er konseptuelt veldig simpel i likhet med substitusjonschiffer. Disse lages ved å endre på rekkefølgen av karakterer i en melding, noe som også kan gjøres på forskjellige måter. For eksempel kan du manuelt bytte bokstavene i tilfeldig rekkefølge sånn at chifferteksten «atj gikel rme» fremstille meldingen «jeg liker mat». En mer strukturell type transposisjonschiffer er *kolonnemessig transposisjon,* en operasjon brukt i ADFGVX-chiffer, mer om dette om litt. Men siden hverken transposisjons- eller substitusjonschiffrene er særlig robuste algoritmer alene, skal vi nå, gjennom ADFGVX-chiffer, se hvordan du kombinere flere simple operasjoner til å lage en mer avansert algoritme.

ADFGVX-chiffer er et fraksjonerende transposisjonschiffer utviklet av Fritz Nebel for bruk i den tykse hæren i første verdenskrig(<https://en.wikipedia.org/wiki/ADFGVX_cipher>). Algoritmen utvider på ideen om transposisjon med å introdusere fraksjonering og ved å kombinere transposisjon med substitusjon. Dette går ut på at man legger ut alfabetet i et rutenett, et såkalt polybius kvadrat. I ADFGX, en tidligere implementasjon av algoritmen, var størrelsen på rutenettet 5x5. Dette ga oss totalt 25 mulige krypterbare tegn, der man kunne slå sammen 2 bokstaver, i og j i original implementasjon, i en rute for å dekke hele det latinske alfabetet. Men i denne algoritmen inneholdte chifferteksten alltid alle tall ukryptert, noe som var en åpenbar svakhet. Derfor ble V’en i ADFGVX lagt til, noe som gir oss et 6x6 rutenett. Dette gir oss i stedet 36 mulige krypterbare tegn, nok til hele alfabetet (26) og alle 10 siffer. Dette rutenettet skal vi så fylle opp med hele alfabetet og sifferene våre, men rekkefølgen på dette alfabetet skal holdes hemmelig. Det er i lagingen av dette hemmelige blandede alfabetet vi gjør substitusjonsdelen av algoritmen. Så la oss lage dette polybius kvadratet.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **A** | **D** | **F** | **G** | **V** | **X** |
| **A** | a | 3 | p | x | 9 | m |
| **D** | 7 | e | w | g | d | 5 |
| **F** | f | 6 | u | 4 | v | l |
| **G** | 8 | i | h | c | 2 | y |
| **V** | o | t | 0 | j | b | n |
| **X** | q | r | s | k | 1 | z |

Nå som vi har dette kvadratet kan vi kryptere meldingen vi skal sende. Da vil for eksempel meldingen «jeg liker mat» oversettes til koordinatene til hver bokstav i tabellen vår. Her vil den første bokstaven være raden og den andre bokstaven være kolonnen i polybius kvadratet vårt. Da ender vi opp med chifferteksten «VG DD DG FX GD XG DD XD AX AA VD»

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| J | E | G | L | I | K | E | R | M | A | T |
| VG | DD | DG | FX | GD | XG | DD | XD | AX | AA | VD |

Så skal vi innføre en såkalt *kolonnemessig transposisjon.* Det vil si at vi skriver denne chifferteksten inn i et nytt rutenett med et nøkkelord eller frase som overskrift, la oss bruke «hallo». Overskriften har en lengde på 5 bokstaver så rutenettet får en lengde på 5 som vi fyller ut med koordinatene i våres originale polybius kvadrat, vi wrapper over til neste rad for hver femte koordinat. Vi skriver opp dette nye rutenettet**.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **H** | **A** | **L** | **L** | **O** |
| V | G | D | D | D |
| G | F | X | G | D |
| X | G | D | D | X |
| D | A | X | A | A |
| V | D |  |  |  |

Neste steg er å alfabetisk sortere overskriften og dens tilhørende kolonne. Vi holder denne operasjonen stabil. Det vil si at dersom det er to kolonner med lik bokstav i overskriften beholder de deres forholdsvise rekkefølge før denne operasjonen. Altså i vårt tilfelle forblir L kolonnen med tilhørende DG og XD verdiene før L kolonnen med FX og AX verdiene. I denne prosessen er det også vanlig å fylle inn tomme verdier på nederste rad med tilfeldige verdier, sånn at man ikke skal kunne se hvilken rekkefølge kolonnene har blitt transposisjonert til. La oss legge inn dette også.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **A** | **H** | **L** | **L** | **O** |
| G | V | D | D | D |
| F | G | X | G | D |
| G | X | D | D | X |
| A | D | X | A | A |
| D | V | F | X | D |

Vi er nå ferdig med den kolonnemessige transposisjonen og har det vi trenger for å sende meldingen vår kryptert. For å sende meldingen vår trekker vi verdiene ut av rutenettet og skriver det på en linje. Da ender vi til slutt opp med chifferteksten «GFGAD VGXDV DXDXF DGDAX DDXAD».

For å dekryptere denne meldingen kan du bruke nøkkelordet (hallo) for å ganske enkelt rekonstruere rutenettet. For å gjøre dette legger du chifferteksten inn i et rutenett der hver kolonne separeres av mellomrommene i chifferteksten. Så kan vi sortere nøkkelordet vårt og legge det til som overskrift. Da ender vi opp med det siste rutenettet tegnet ovenfor. Derfra kan vi ta kolonnene tilbake til sin egentlige plass, det vil si der de lå før den kolonnemessige transposisjonen. Da får vi det første rutenettet tegnet ovenfor. Til slutt kan vi lese av koordinatene fra det hemmelige polybius kvadratet for å endelig dekryptere meldingen.

Hver for seg er transposisjons- og substitusjonschiffer ganske enkelt å dekryptere, men som vi ser gjennom ADFGVX-chiffer kan du få noe ganske komplisert hvis du kombinerer disse metodene. Vi kan nå se at vi begynner å nærme oss kryptografiske løsninger som er svært vanskelige å dekryptere for utenforstående med mindre de har tilgang til de(n) hemmelige nøkkelen(e). I motsetning til klassisk substitusjon- og transposisjonschiffer, der hvem som helst kan dekryptere chifferteksten så lenge de vet hvordan den er kryptert, ser vi at det nå begynner å danne seg en ny flaskehals rundt hemmelige nøkler, den nye store tingen innen kryptografi, i det vi nærmer oss moderne kryptografi.

1. Enigma

Antageligvis den mest velkjente krypteringsalgoritmen som finnes, var enigma et gigantisk steg fremover i historien av kryptografi. Den viste oss hvor avansert krypteringsalgoritmer kunne være og hvor vanskelig de kunne bli å knekke. I tillegg, gjennom Alan Turing og hans medarbeideres arbeid i å knekke enigma, bidro den til utviklingen av datamaskinen og alt som kom med den, antageligvis en av de viktigste utviklingene i historien. I dette delkapittelet skal vi undersøke hvordan enigma maskinen virket, hvordan den til slutt ble knekket, og til slutt hvilke bidrag denne hadde for fremtiden.

A picture containing electronics

Description automatically generatedA picture containing electronics

Description automatically generatedEnigma maskinen var utviklet av Arthur Scherbius på 1920-tallet. Maskinen ble notorisk brukt av nazi Tyskland under 2. verdenskrig Ettersom maskinen først var utviklet langt før utbruddet av 2. verdenskrig inkluderte dens originale markedsmålgruppe både ikke-militære og utenlandske organisasjoner, som vi kan se på variantene av enigma deriblant «Swedisch» og «Swiss K» ). <https://www.cryptomuseum.com/crypto/enigma/hist.htm>).

1. Moderne kryptografi

I moderne kryptografi har mange store ting blitt oppnådd. Fra gamle dager av romerne hvor Caesar Cipher, et substitusjons-chiffer, ble oppfunnet som skifter karakter med tre steder hvor en A blir til en D, B blir E og så videre. For sin tid var det enkelt og effektivt, men langt fra det vi har i dag. Selv med ADFGVX-chifferet som er en mye mer avansert algoritme enn noen implementering av et substitusjons- eller transposisjons-chiffer, er den fortsatt ikke sammenlignbar med krypteringsmetodene vi har i dagens moderne kryptografi. Der kryptografi begynner å ta form er etter 1. verdenskrig. Rett før slutten av 1. verdenskrig ville en elektromekanisk maskin bli oppfunnet hvor en nøkkel er innebygd i en roterende skive. Dette ville være forløperen til Enigma-maskinen. En tysk ingeniør, Arthur Scherbius ville deretter finne opp Enigma-maskinen som ikke bare ville bruke en rotorskive, men flere. Dette vil da bli brukt av det tyske militæret på grunn av dets effektivitet med å kryptere informasjon. Imidlertid er alle disse metodene nå foreldet med begynnelsen av informasjonstiden hvor vi har datamaskiner raskere enn noen gang kunne drømme om sammen med komplekse algoritmer for å dekryptere all informasjon kryptert med gamle metoder. Med utviklingen innen maskinvare har vi også gjort fremskritt innen kryptografi, og slik utvikling skal vi se med Diffie-Hellman nøkkelutveksling og RSA.

1. Diffie-Hellman nøkkelutveksling

Diffie-Hellman nøkkelutveksling er et av de tidligste eksemplene på offentlig nøkkelutveksling. Den ble laget gjennom et felles samarbeid mellom Ralph Merke og navnet ble laget mellom Whitfield Diffie og Martin Hellman. Denne metoden for digital kryptering lar to parter som ikke har noen forkunnskaper om hverandre, lage en dekrypterings nøkkel sammen. For å oppnå dette må imidlertid to partier – Alice og Bob – utveksle informasjon gjennom en usikker kommunikasjonskanal for å kunne utføre DH-nøkkelutvekslingen.

Begge partier må bli enige om et veldig stort primtall,

NN

. De må også bli enige om et andretall,

GG

der

GG

er mindre enn

NN

og må være primitivt i forhold til

NN

. I tallteori er et tall

GG

en primitiv rot modulo

NN

hvis hvert tall coprime til

NN

er kongruent med potensen

GG

. Et elementært eksempel vil være tallet 3 er en primitiv rot modulo 7 siden:

31   =   30×3   ≡   1×3   =   3   ≡   331   =   30×3   ≡   1×3   =   3   ≡   3

32   =   31×3   ≡   3×3   =   9   ≡   232   =   31×3   ≡   3×3   =   9   ≡   2

33   =   32×3   ≡   2×3   =   6   ≡   633   =   32×3   ≡   2×3   =   6   ≡   6

34   =   33×3   ≡   6×3   =   18   ≡   434   =   33×3   ≡   6×3   =   18   ≡   4

35   =   34×3   ≡   4×3   =   12   ≡   535   =   34×3   ≡   4×3   =   12   ≡   5

36   =   35×3   ≡   5×3   =   15   ≡   136   =   35×3   ≡   5×3   =   15   ≡   1

37   =   36×3   ≡   1×3   =   3     ≡   337   =   36×3   ≡   1×3   =   3     ≡   3

(Wikipedia, 2022)

Alle verdier i den siste kolonnen er alle rester som ikke er null modulo 7, noe som antyder at 3 er en primitiv rot modulo 7. Dette kan observeres fra det faktum at sekvensen

gkgk

alltid gjentar modulo

nn

etter en verdi av

kk

, siden modulo

nn

produserer bare et begrenset antall verdier. Hvis

gg

er en primatrot modulo

nn

, og

nn

er primtall, begynner repetisjonen ved

n−1n−1

. (Wikipedia, 2022)

Med introduksjonen av primitiv rot modulo ute av veien, velger partiene – Alice og Bob –

NN

og

GG

og utveksler disse verdiene til hverandre. De vil deretter følge denne algoritmen for å lykkes med å lage sin egen private og offentlige nøkkel.

Text

Description automatically generatedText, letter

Description automatically generated

(Wikipedia, 2022)

De offentlige nøklene,

AA

og

BB

, som navnet antyder, er åpent til alle sammen, mens den private nøkkelen,

aa

og

bb

, ikke er åpent til noen bortsett fra henholdsvis Alice og Bob. La oss ta et eksempel for å vise prosessen med algoritmen. Normalt er

NN

og

GG

veldig store tall, men i dette eksemplet skal vi ha

GG

til å være 3 og

NN

til å være 7 for å gjøre det enklere med våre forkunnskaper fra tabellen.

Text

Description automatically generatedText

Description automatically generated

De eneste nøklene som holdes hemmelige er de private nøklene. Alle de andre verdiene utveksles gjennom en usikker kommunikasjonskanal. Styrken til Diffie-Hellman nøkkelutveksling er avhengig av at

Ba mod N =Ab mod N

tar ekstremt lang tid å beregne med dagens moderne maskiner. Til tross for at du har kunnskap om alle verdier bortsett fra de private nøklene, er det fortsatt ekstremt vanskelig å beregne selv ved hjelp av komplekse algoritmer.

Diffie-Hellman-nøkkelutvekslingen er veldig sikker mot avlytting hvis begge parter velger riktig verdi for

GG

. Men Diffie-Hellman-utvekslingen kan fortsatt bli utnyttet for et man-in-the-middle angrep, da den ikke gir autentisering til Alice og Bob. En angriper kan etablere to distinkte nøkkelutvekslinger, en med Alice og den andre med Bob, og etterligne begge partier som lar den aktive angriperen dekryptere og kryptere informasjonen som utveksles gjennom en usikker kommunikasjonskanal. Derfor er Diffie-Hellman implementert med en algoritme som DSA eller RSA for å autentisere en eller begge partiene som anses som den sikreste implementeringen blant mange varianter av Diffie-Hellman nøkkelutveksling.

1. RSA

Rivest–Shamir–Adleman, eller ofte kjent som RSA, implementeres ofte sammen med Diffie-Hellman nøkkelutveksling for å gi autentisering mellom Alice og Bob og for å forhindre et mann-i-midten-angrep. I motsetning til Diffie-Hellman, kan RSA ikke bare brukes til å utveksle digitale signaturer, men kan også kryptere informasjon. Kryptering av meldinger med RSA brukes imidlertid ikke ofte da det er for ineffektivt og brukes først og fremst til å kryptere session nøkkelen for meldingsintegritet eller digital signatur. Styrken til RSA er identisk med Diffie-Hellman i den forstand at matematiske operasjoner er raske å beregne, men svært vanskelige å reversere. Begge protokollene muliggjør kryptering av informasjon, men det er vanskelig å reversere informasjonen.

Implementeringen av RSA er sterkt avhengig av modulær aritmetikk, Eulers teorem og totientfunksjon. For å opprette en RSA offentlig eller privat nøkkel, må denne algoritmen følges.



(Brilliant.org, 2022)

Den offentlige nøkkelen er tallparet av

nn

og [Formel]. Til tross for at begge tallene utveksles på en usikker kommunikasjonskanal, vil det fortsatt ta ekstremt lang tid å bestemme [Formel] med moderne datamaskiner selv ved hjelp av komplekse algoritmer.

Med kunnskapen vi har om å lage offentlige og private nøkler med RSA, skal vi nå gå gjennom hvordan RSA krypterer. For å kryptere en melding, [Formel], med den offentlige nøkkelen for å lage chifferteksten, [Formel], må denne algoritmen følges.



Som med Diffie-Hellman tidligere, skal vi demonstrere et eksempel med små tall. Tallene [Formel] og [Formel] er generelt svært store tall for ytterligere å styrke sikkerheten til protokollen på grunn av at dens styrke er avhengig av datamaskiner tar kort tid å beregne [Formel] i forhold til å finne de to primtallene som er faktor til [Formel] som tar år eller noen ganger til og med lengre enn levetiden til vår egen planet.



Vi har nå vår offentlige nøkkel [Formel]  og vår private nøkkel [Formel]. Hvis vi skulle utveksle en melding, [Formel], bruker vi en vanlig konverteringsprosess ASCII-alfabetet. Men hvis vi skulle lage en melding som er større enn [Formel], ville meldingen bli delt opp i biter. La oss ta ordet **MATTE** som et eksempel, siden alle alfabeter har sin egen unike verdi, må vi dele verdiene og bytte dem individuelt. Siden alle alfabeter er større enn n må vi først dele M til 7 og 7, A til 6 og 5 og så videre.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **A** | **B** | **C** | **D** | **E** | **F** | **G** | **H** | **I** | **J** | **K** | **L** | **M** |
| 65 | 66 | 67 | 68 | 69 | 70 | 71 | 72 | 73 | 74 | 75 | 76 | 77 |
| **N** | **O** | **P** | **Q** | **R** | **S** | **T** | **U** | **V** | **W** | **X** | **Y** | **Z** |
| 78 | 79 | 80 | 81 | 82 | 83 | 84 | 85 | 86 | 87 | 88 | 89 | 90 |



Med RSA nå implementert, er autentisering nå mulig. Begge partier – Alice og Bob – vil signere en melding ved å bruke sin private RSA-nøkkel og utveksle den til hverandre. Med dette kan de både bekrefte identiteten til mottakeren og avsenderen ved å sjekke de signerte meldingene med deres offentlige nøkkel med den andres digitale sertifikat. Med begge partier autentisert er det teknisk mulig å fortsette å kryptere meldinger og trygt utveksle dem på en usikret kommunikasjonskanal med RSA, men det vil være for ineffektivt. For å unngå denne ineffektiviteten brukes RSA ofte sammen med andre sikkerhetsprotokoller som Diffie-Hellman nøkkelutveksling for å etablere en delt symmetrisk nøkkel. Dette brukes deretter i en symmetrisk nøkkelalgoritme for å kryptere informasjonen som Alice og Bob alltid hadde til hensikt å utveksle trygt og trygt seg imellom.

På grunn av robustheten til RSA, brukes den overalt hvor kryptering er nødvendig. Det brukes også for å sikre at nettsteder er legitime for å filtrere bort bedrageren siden bare det virkelige nettstedet har den private nøkkelen. Det eliminerer derfor effektivt et man-in-the-middle angrep der informasjonen din ville bli fanget opp og etterlignet en eller begge partier. Mens RSA anses som sikker, er styrken avhengig av nøkkelstørrelsen, som er antall biter som er is [Formel]. Med moderne algoritmer og datamaskiner kan 512-biters RSA lett bli brutt forcet der den private nøkkelen trekkes ut på bare et par timer. I dag skapes det 4090-bit RSA ettersom det er standarden, da hvor som helst lavere anses som risikabelt. Hvis en sårbarhet skulle bli funnet i RSA ville det ha en katastrofal konsekvens ikke bare i kryptografi, men overalt på grunn av at kryptering er så viktig i bank, netthandel og så mange andre. En mann ved navn Peter Shor utviklet en algoritme for kvantedatamaskiner som er i stand til å faktorisere tall raskere enn noen maskin vi har i dag. Kvantedatamaskiner har imidlertid fortsatt en lang vei å gå.

1. Kvantekryptering

(Mulig at kvantekryptering burde være siste delkapittel)

Utviklingen av datamaskiner igjennom årene har ført til at datamaskiner blir raskere og smartere hele tiden. Dette gjør det mye enklere å dekryptere meldinger, noe som fører oss til trusselen potensielle kvantedatamaskiner kan utgjøre. I motsetning til dagens moderne datamaskiner som bruker enere og nuller, kalt bits, er kvantedatamaskiner basert på kvantemekanikk og bruker kvantebits, kalt qubits. Vanlige bits kan ha to forskjellige tilstander, 1 og 0 som representerer av og på, imens qubits derimot har i tillegg en tredje tilstand som heter *superposisjon*. Superposisjon refererer til et fenomen innenfor kvantemekanikk hvor et kvantesystem kan være i flere tilstander eller steder samtidig, som innenfor kvantedatamaskiner baserer seg på partiklene i superposisjon([https://iotpractitioner.com/quantum-computing-series-part-4-superposition-in-quantum-mechanics/](https://iotpractitioner.com/quantum-computing-series-part-4-superposition-in-quantum-mechanics/" \t "_blank) ). Disse partiklene representerer qubits, som kan holde verdien 0, 1 eller begge samtidig.  Det som gjør kvantedatamaskiner så utrolig mektig er at vi kan benytte qubits i superposisjon for å representere et eksponentielt antall tilstander, som ved hjelp av kvantealgoritmer vil kunne beregne mange mulige svar for ett gitt input. Nå som vi forstår at kvantedatamaskiner kan utgjøre en stor trussel for informasjonssikkerhet, mer spesifikt kryptografi, over hvor mektige de kan bli, skal vi se på ulike måter vi kan muligens forsvare oss mot denne forestående trusselen.

Den mest utbredte og kjente metoden innenfor kvantekryptografi er det vi kaller *kvantenøkkelutveksling*, på engelsk heter det *Quantum Key Distribution*, og blir ofte forkortet til *QKD*.

(Litt spekulasjon/refleksjon, burde spesifisere entropi og at dette dreier seg mer om «true random numbers» og generators for dem) – kommentar, fjern senere

Kvantekryptografi baserer seg på den tilsynelatende tilfeldigheten til kvantemekanikk. Denne tilfeldigheten kan være ekte 100% tilfeldighet, men hva om det ikke er det? Hva skjer med kryptografien i det tilfellet? Kan dette være en potensiell svakhet for kvantekryptografi i fremtiden, noe vi burde vurdere på forhånd slik at vi er forberedt? Vanskelig å si, men ideen bak kvantekryptografien sin tilfeldige natur kan virke litt urimelig ved første øyekast. Dette er fordi fysiske objekter, innenfor vår kunnskap, må følge fysikkens lover, ellers gir det ikke mening og ting vil bare skje uten en grunn eller årsak. Dersom vi zoomer inn til kvanteverden, virker det som om prosessene er tilfeldige, men hvis ting er tilfeldige på mikro nivå, vil ikke de samme tingene være tilfeldige på makro nivå når vi zoomer ut igjen? Disse større objektene, som mennesker, biler, planeter, osv. er jo tross alt bygd opp av disse mindre tingene i kvanteverden. Kan hende kvanteverden ikke er like tilfeldig som vi tror, at vi bare ikke klarer å måle mønstre på en såpass liten skala at det gir mening og dermed virker det uforutsigbart, altså tilfeldig.

En grunn for at kvanteverden virker så kaotisk at det blir tilfeldig, kan skyldes at vår teknologi ikke er bra nok til å kunne nøyaktig måle og utforske kvanteverden riktig ennå. Dermed ser det ut til at det kan være litt for tidlig for oss å konkludere om kvanteverden faktisk er tilfeldig, pseudo-tilfeldig eller deterministisk. Dette ble litt filosofisk, men poenget med dette er at dersom vi konkluderer basert på uvitenhet om tilfeldigheten bak en slik kryptografisk algoritme, kan dette føre til en svakhet innenfor kvantekryptografi. Eller på en annen side kan det hende at det fremdeles forblir så utrolig kompleks uansett hvilken forutsigbarhet algoritmen har at en kvantedatamaskin fortsatt ikke kan knekke det. I så fall vil det ikke være en svakhet. En lignende problemstilling blir da at hvis algoritmen for krypteringen er tilfeldig, hvordan vil man da kunne dekryptere det med en nøkkel gitt at chifferteksten er fullstendig tilfeldig og har dermed ikke noe mønster som kan spores tilbake til den originale meldingen? Hvis chifferteksten faktisk ikke er fullstendig tilfeldig og har da et mønster, så må vi kunne anta at det er mulig å knekke denne antatte "uknekkelige" algoritmen med en mektig nok kvantedatamaskin.

1. Konklusjon

Referanser

Brilliant.org, 2022. *RSA Encryption.* [Internett]    
Available at: https://brilliant.org/wiki/rsa-encryption/   
[Funnet 31 10 2022].

Etternavn1, F. & Etternavn2, F., 2022. Et publisert tidskrift. *Tidskrift 1*, Oktober, pp. 1-12.

Jøsang, A., 2021. *Informasjonssikkerhet - teori og praksis.* s.l.:Universitetsforlaget.

Wikipedia, 2022. *Diffie-Hellman Key Exchange.* [Internett]    
Available at: https://en.wikipedia.org/wiki/Diffie%E2%80%93Hellman\_key\_exchange   
[Funnet 23 Oktober 2022].

Wikipedia, 2022. *Primitive root modulo n.* [Internett]    
Available at: https://en.wikipedia.org/wiki/Primitive\_root\_modulo\_n   
[Funnet 23 Oktober 2022].

<https://www.romeandart.eu/en/art-cipher-jfulius-caesar.html> - brukt 2022-10-26

<https://commons.wikimedia.org/wiki/File:Caesar3.svg> – brukt 2022-10-26

<https://commons.wikimedia.org/wiki/File:CipherDisk2000.jpg> – brukt 2022-10-26

(<https://en.wikipedia.org/wiki/ROT13>

<https://en.wikipedia.org/wiki/Caesar_cipher>

<https://en.wikipedia.org/wiki/Substitution_cipher>

<https://en.wikipedia.org/wiki/Cipher_disk>

<https://en.wikipedia.org/wiki/ADFGVX_cipher> )

<https://www.cryptomuseum.com/crypto/enigma/hist.htm>