Indhold

[Hemmeligholdelse 2](#_Toc501444565)

[SSL - The Secure Socket Layer Protocol 2](#_Toc501444566)

[RSA 3](#_Toc501444567)

[One-Time-Pad 3](#_Toc501444568)

[Autencitet 3](#_Toc501444569)

[OAuth 3](#_Toc501444570)

[Hashing 3](#_Toc501444571)

[JWT 3](#_Toc501444572)

[Hacking 3](#_Toc501444573)

[Sql injection 3](#_Toc501444574)

[Javascript injection 3](#_Toc501444575)

[Cross site scripting 3](#_Toc501444576)

[Hacking I C 3](#_Toc501444577)

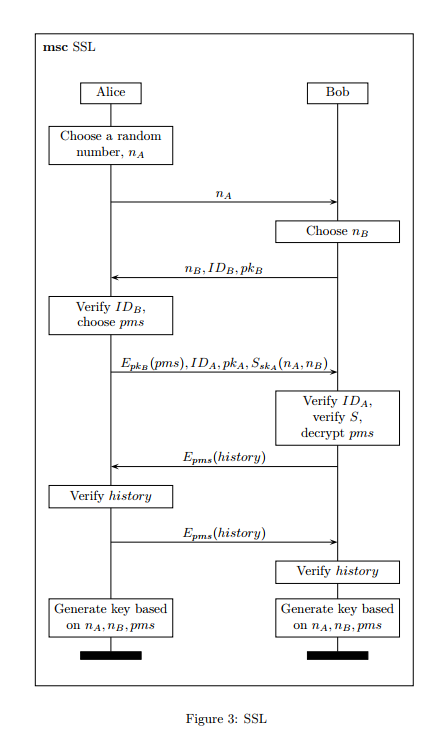
# Hemmeligholdelse

## SSL - The Secure Socket Layer Protocol

**Hvad burger man det til?**

*Secure Socket Layer* er en protokol, som benyttes til at etablere en sikker forbindelse mellem to parter (server og client). Derved sikre hemmeligholdelse (og reelt også autentifikation).

**Hvordan fungerer det?’**

Alice = client

Bob = server

Pk = public key

Sk = Secret key

Pms = Premaster secret (random generated)

Sska(nA,nB) = nounce a og b Signed med Alices sekret key

History = er history af beskeder sendt mellem hinanden, for at sikre at begge parter har det samme antal.(der kunne eventuelt have været en tredjepart omkring, og derfor ville beskederne ikke passe i antal)

ID = sendes med når du sender din Public key.

* SSL er en sammenkobling af RSA og OneTImePad – 3 første liner er RSA – 2 sidste er OTP.
* A sender nounce
* B sender sin nounce, ID, Pk
* A sender pms krypterede med sin public key, sit ID, sin Pk, og nA og nB, der er signed med sin Sk.
* B sender sin history krypteret med pms
* A Sender sin history krypteret med pms
* Til sidst bliver der genreret en key ud fra nounce og pms.

**Hvad er alternativer/fordele/ulemper?**

**Fordele**

* Hvis der er nogen som opsnapper en meddelelse, kan de kun læse den krypterede meddelelse, og ikke den dekrypterede.
* Hemmeligholdelse, verifikation, tillid for brugeren.

**Ulemper**

* Systemet bliver langsommere end hvis man skulle sende ikke krypterede data. Da kryptering altid tager ekstra ressourcer

## RSA

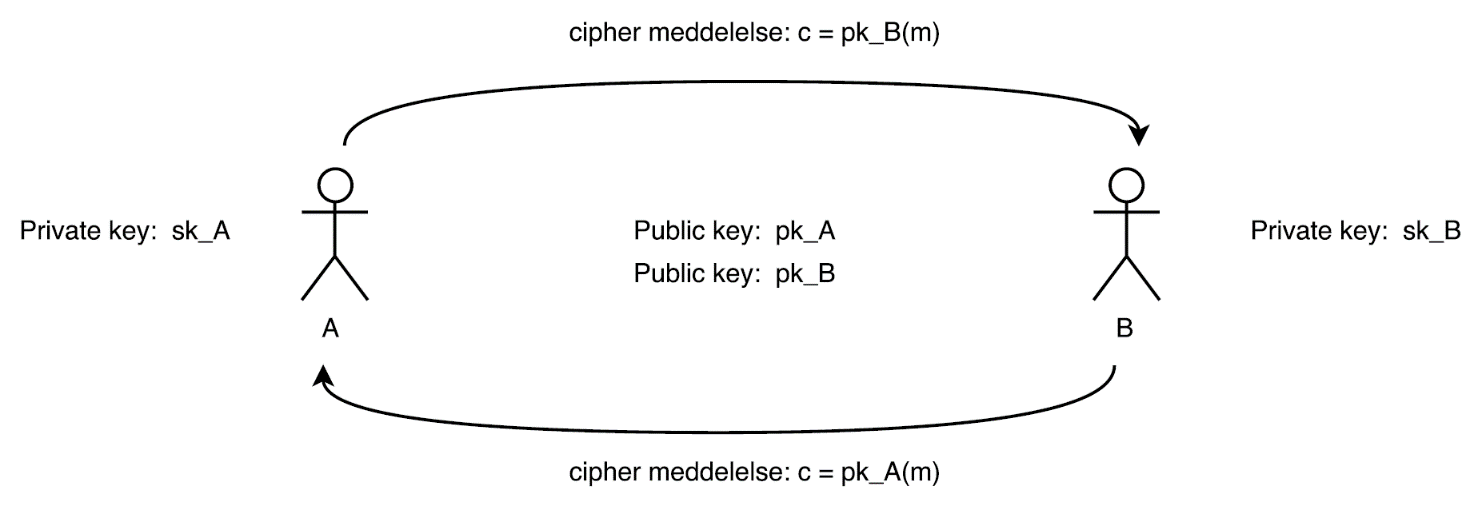
**Hvad bliver det brugt til?**

Benyttes til, at sikre at det kun er en bestemt person, som kan læse vores beskeder, ved at bruge et asymmetrisk nøgle par.

**Hvordan fungerer det?**

De to parter, som ønsker at kommunikere, har hver et sæt nøgler; *public-* og *private key*. Begges *public keys* er frit tilgængelige, mens begges *private keys* kun er tilgængelige for dem selv.

Når part A ønsker at kommunikere en meddelelse til part B, kryptere part A meddelelsen med part Bs *public key*, og sender dem. Nu er det kun part Bs *private key*, som kan dekryptere meddelelsen.



**Hvad er alternativer/ fordele og ulemper?**

Alternativ:

* One-time pad

Fordele:

* Nøglerne kan genbruges flere gange
* Man behøver kun at hemmeligholde sin egen nøgle

Ulemper:

* Relativt langsomt (meget matematik)

## One-Time-Pad

**Hvad bliver det brugt til?**

Benyttes til, at sikre at det kun er en bestemt person, som kan læse vores beskeder, ved at bruge et en symmetrisk nøgle, som skal deles imellem parterne og som er lige så lang eller længere, end beskeden som skal kommunikeres.

**Hvordan fungerer det?**

Part A som gerne vil sende en krypteret meddelelse til part B, tager det tilsvarende antal tegn fra det ens *one-time pad*. Hvert tegns placering i alfabetet udgøre en værdi. Værdierne fra meddelelsen og værdierne fra *one-time pad* nøglen, lægges sammen til en ny værdi. Er den samlede værdi større end antallet af bogstaver i alfabetet (26), tages værdien % 26.

Dette giver en resultatværdi, hvor man finder det tilsvarende bogstav i alfabetet. Dette giver så den krypteret meddelelse.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| H | E | J | M | E | D | D | I | G |
| 8 | 5 | 10 | 13 | 5 | 4 | 4 | 9 | 7 |
| J | I | Y | N | K | C | D | E | P |
| 10 | 9 | 25 | 14 | 11 | 3 | 4 | 5 | 16 |
| 18 | 14 | 9 | 1 | 16 | 7 | 8 | 14 | 23 |
| R | N | I | A | P | G | H | N | W |

**Hvad er alternativer/ fordele og ulemper?**

Alternativ:

* RSA

Fordele:

* Relativt hurtigt at anvende
* Sikker og kan ikke knækkes

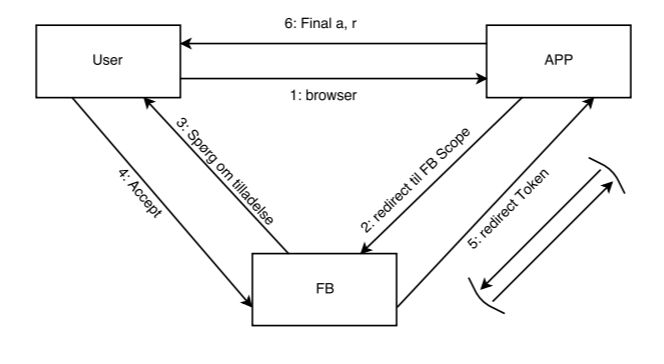
Ulemper:

* Begge parter SKAL have samme key
* Nøglen kan kun bruges én gang

# Autencitet

## OAuth

**Hvad burger man det til?**

Oauth er en måde at sikre autentifikation. Det bliver brugt til at opnå adgang til en brugers information eller oplysninger fra en tredjeparts side, fx facebook eller sportify. Derudover bliver det brugt til at undgå at man selv skal opbevarer login data, eller i det hele taget at have sit eget login system.

**Hvordan fungerer det?**

1. En client kontakter appen
2. Appen redirecter til FB med scope
3. FB sender login side til bruger
4. Bruger acceptere
5. FB laver callback med acces og refresh token til app
6. App sender acces token videre til bruger

De to ekstra er hvis der bliver sendt en token til appen fra FB først, hvorefter appen så bruger denne token til at requeste acces og refresh token

**Fordele/Ulemper/alternativer**

Fordele:

* Det er simpelt for brugeren
* Mulighed for at tilpasse rettigheder kontra apikeys

Ulemper:

* Man er afhængig af tredjepart, så hvis siden lukker ned, virker det ikke længere

Alternativer:

* JWT

## Hashing

https://crackstation.net/hashing-security.htm#normalhashing

**Hvad burger man det til?**

Verificering og sammenligning af fx passwords.

Man bruger hashing til at ændre en mængde data, om til en streng af karakterer med en bestemt længde (som ikke kan blive ændret tilbage). Det har den fordel at hvis inputet bliver ændret bare en lille smule, bliver resultatet af strengen ændret fuldstændig.

I forbindelse med bruger password bruges det til at når en bruger gemmer sit password, bliver det hashed og lagt i databasen. Når så en bruger skal logge ind igen, bliver inputet hashed og sammenlignet med den hash værdi der ligger i databasen. Stemmer de overens bliver brugeren logget ind.

**Hvordan fungerer det?**

Det fungerer på den måde at man tager noget fra et stort domæne og gør det til et lille domæne. Fx alle ord i verden og gør til et 32 karakter lang streng med bogstaver og tal fra a- z og 0 -9. Man kan ydermere bruge salt i sin hashing, dette vil sige man kan tilføje en ekstra streng til sin hashing, som gør at den bliver sværere at knække. Salt kan være statisk eller dynamisk, ved dynamisk vil det være nødvendigt at opbevarer den dynamiske salt i databasen.

**Fordele/Ulemper/Alternativer**

Fordele:

* Hvis inputet ændres det mindste ændres hashingen totalt.

Ulemper:

* Flere password kan have samme hashcode.
* Der er nogle hashings der ikke er sikre længere, det er vigtigt man bruger de nyeste og hele tiden holder sig opdateret her omkring.

Alternativer:

## JWT

<https://jwt.io/introduction/>

https://medium.com/vandium-software/5-easy-steps-to-understanding-json-web-tokens-jwt-1164c0adfcec

**Hvad burger man det til?**

Benyttes til at sikre autentifikation, altså vi kan verificere at hvem vi kommunikere med, og at det er den samme person for hver forespørgsel.

**Hvordan fungerer det?**

Når en klient kontakter en server, for at logge ind, returneres der en JWT (Hvis login lykkedes). Denne JWT gemmes ved clienten lokalt eller i en cookie. Herefter sender clienten altid sin JWT med når der laves en forespørgsel til serveren. På denne måde kan server se at det er den samme client.

Hvad består den af?

Den bliver lavet og sendt i json objekter.

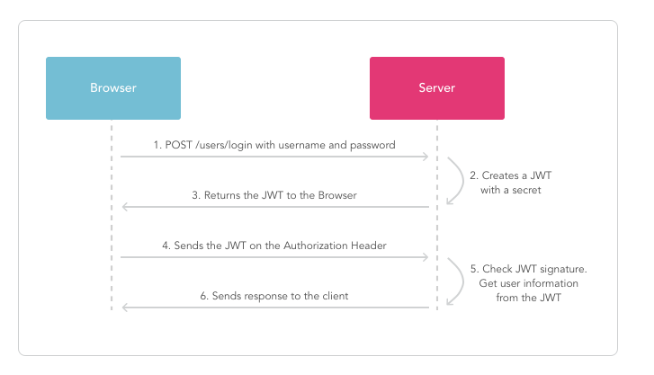
JWT er delt op i tre dele, delt af punktummer.

* Header
* Payload
  + Registret claims
  + Public claims
  + Private claims
* Signature

Header: Består af to dele, en type = JWT og en hashing = HMAC SHA256 or RSA

Payload: Indeholder information omkring typisk brugeren(public), eller fx omkring hvornår din token udløber(regisret).

Signature: Bliver brugt til at bekræfte at senderen af JWT er hvem han siger og til at bekræfte at beskeden ikke er blevet ændret på. Den indeholder Header og payload krypteret med en sekret key, derved kan det verificeres at der ikke er blevet ændret på dataen.



**Fordele/Ulemper/alternativer**

* Fordele:
  + kompakt 🡪 få antal bits,
  + simple web tokes, kan ikke bruge RSA keys, men det kan JWT, SWT kan du bruge symmetriske nøgler
  + Bruger Json objekter, som mange sprog kan håndtere.
* Alternativer:
  + At bruge session (vi har ikke haft om det)

# Hacking

### Cross site scripting (XSS)

**Hvad er det?**

Er når udefrakommende og utilsigtet kode, bliver behandlet på en side, via et eller anden form for input. Kan f.eks. benyttes til, at påvirke cookies eller sessions data.

Dette kan f.eks. gøres ved at bruge *JavaScript Injection*.

Denne type angreb kan både benyttes til brud på autentifikation og hemmeligholdelse.

**Hvordan fungerer det?**

*Reflected*

Når der f.eks. sendes en GET forespørgsel med en e-mail, hvoraf e-mail adressen efterfølgende behandles direkte på siden. Her kan man i URLen udskifte e-mail adressen ud med noget ondsindet JavaScript.

*Stored*

Når der f.eks. sendes en POST med en e-mail, hvoraf e-mail adressen efterfølgende gemmes i en database. Her kan man udskifte e-mail adressen ud med noget ondsindet JavaScript. Når e-mailen senere hentes ud igen, vil det ondsindet kode køres.

**Hvad kan man gøre for at undgå det?**

For at imødekomme denne type angreb, skal man *sanitise* eller kontrollere alt data. Både det data der tages som input, men også det data der behandles som output.

### SQL Injection

**Hvad er det?**

Når udefrakommende kode benyttes til, at opnå adgang til en SQL-database.

Dette kan resultere i en række forskellige muligheder; at den uvedkommende får mulighed for at tilføje, ændre eller slette data fra vores database.

Den uvedkommende kan f.eks. stjæle vigtig følsom information eller opnå administrations rettigheder, og derved brydes både autentifikation og hemmeligholdelse.

**Hvordan fungerer det?**

Når en ondsindet person indsætter SQL-kode i f.eks. et input felt, som man forventer har kontakt til en SQL-database.

**Hvad kan man gøre for at undgå det?**

Begræns database brugerens. Overvej om det er nødvendigt at brugeren kan slette eller opretter tabeller, eller om brugeren kun skal kunne læse data.

Altid *sanitise* eller kontrollere data.

Benytte SQL-Parameters i alle kald til database, fra serveren.

### Buffer overflow

**Hvad er det?**

En type af angreb, som bygger på, at visse sprogkræver at man manuelt håndtere hukommelse.

Sprog hvor man selv håndtere hukommelses allokering kan kaldes *low level*.

**Hvordan fungerer det?**

Det fungerer ved, at man skriver så meget ind i *stackens* hukommelses buffer, at man overskriver returreferencerne. Derved kan returreferencen ændres til noget nyt og utilsigtet. Denne nye reference kan udnyttes og potentielt indeholde en reference til noget ondsindet kode.

Men kan da lave et stykke ondsindet kode, som er placeret på den retur reference, som

Et eksempel er, at man kan konstruere en streng, som fylder mere end den mængde hukommelse som er allokeret til denne variabel. Det som går udover mængden af hukommelse der er allokeret, vil da erstatte, eller på anden måde påvirke, de variabler, som kommer umiddelbart efter denne i hukommelsen.

Et specifikt eksempel er f.eks. *Heartbleed bug*, som var en fejl i *OpenSSL*. Denne gjorde at man reelt kunne opsnappe og lytte til folks kommunikation.

**Hvad kan man gøre for at undgå det?**