Denne guide er oprindeligt udgivet på Eksperten.dk

COMPUTERWORLD

# Password hashing og salt

## Denne artikel beskriver hvorfor hashing og salt er godt, når man skal gemme passwords.

## Den forudsætter et vist kendskab til programmering og matematik generelt.

Skrevet den **24. jan 2013** af **arne_v** I kategorien **Programmering / Generelt** | ⭐⭐⭐⭐⭐

Historie:
V1.0 - 26/12/2012 - original
V1.1 - 29/12/2012 - ret stavefejl etc.
V1.2 - 23/01/2013 - rette flere stacefejl etc.

### Indledning

Det er efterhånden almindeligt kendt at at det er bedre at gemme hashed passwords fremfor passwords i en bruger database og at det er godt at bruge salt i hashing.

Denne artikel vil forklare hvorfor og give detaljer omkring hvordan det bør gøres.

### Formål

Hvilke trusler beskytter vi imod ved brug af hashing og salt?

At nogen forsøger at logge ind ved at gætte password? Nej! Hashing gør ingen forskel overhovedet.

At web site stjæler password og bruger det til at logge ind på andre web sites hvor samme person har brugt samme password? Nej! Koden kan altid gemme det ikke hashede password et andet sted.

At hackere får adgang til databasen, finder password og bruger det til at logge ind på samme site? Nej! Med adgang til databasen er der ikke noget behov for at logge ind.

At hackere får adgang til databasen, finder password og bruger det til at logge ind på andre web site hvor samme person har brugt samme password? Ja!

At hackere får adgang til et dump/extract med brugernavne+passwords og bruger det til at logge ind på samme site? Ja!

(det sidste lyder umiddelbart usandsynligt, men det sker altså igen og igen ude i virkeligheden)

### Eksempler

Jeg vil regne lidt på forskellige scenarier.

Jeg vil bruge følgende notation:

KWPW = kendt ord som password
RPW(n,[]) = tilfældigt password med længde n og de angivne valide tegn

F1 = finde en bestemt bruger
FS = finde en tilfældig bruger
FA = finde alle

m = antal brugere på web site

Hvis der skal beregnes hashes vil jeg antage at der kan beregnes en milliard per sekund (hvilket vist i skrivende stund er hvad en highend GPU kan klare med SHA-256).

Hvis der skal laves tabel opslag enten i lokal database eller via web service kald vil jeg antage 100 opslag per sekund (lyder lavt men remote access og/eller rainbow transformeringer tager altså lidt tid).

Og lad os antage at der er 1 million kendte ord (hvilket nok passer fint for dansk og engelsk lagt sammen med ord over en vis længde).

**Hashing og salt - definitioner**

En hash funktion som man bruger til hashing af passwords er en funktion som konverterer en streng af vilkårlig længde til em værdi med fast længde (X bits eller Y bytes).

hashed password = hash funktion ( password )

Kendte hash funktioner er:
* MD5
* SHA-1
* SHA-256

En god hash funktion har egenskaberne:
* det er meget nemt at beregne hash udfra input
* det er meget vanskeligt at beregne input udfra hash

Det er den sidste egenskab som gør det velegnet til at gemme passwords med.

Nogen gange kaldes hashing også for en en-vejs kryptering p.g.a. denne egenskab.

Et salt er en værdi som konkataneres med input førend hash funktionen kaldes.

hashed med salt password = hash funktion ( salt concat password )

**Hashing og salt - bruger databaser**

Implementering af hashed med salt passwords i bruger databaser er simpel.

Ved oprettelse af bruger:
* genererer tilfældigt salt
* beregn hash af salt og password
* gem salt og hash i hvert sit felt i bruger databasen

Ved login:
* hent salt og hash fra databasen
* beregn hash af salt fra databasen og det angivne password
* sammenlign det beregnede hash med hash fra database
* ens => login OK, forskellig => login fejl

**Hashing og salt - krølle**

Da output fra hash funktionen har en fast længde og input kan have i princippet en uendelig stor længde, så følger der at der et et uendeligt antal input med samme hash.

Passwords kan ikke være uendeligt lange, men der er stadigvæk mulighed for flere passwords med samme hash.

Så i princippet kan man logge ind med flere forskellige passwords.

Men sandsynligheden for at ramme rigtigt er meget lille.

SHA-256 bruger 256 bit hash values, så chancen for at et tilfældigt valgt password virker mod en bruger database hvor der er brugt SHA-256 er 1 ud af 2^256 eller 1 ud af 1.16*10^77.

Det er så lille en sandsynlighed at ingen har problemer med det.

**Hashing og salt - hvorfor**

Lad os først forklare de basale mekanismer:

hashing : beskytter mod at password læses direkte

salt : beskytter mod tabel opslag af nemme passwords

forskellig salt for hver bruger: vanskeliggør brute force til at finde nogle eller alle passwords

Tabel opslag består i at slå password op via hash i en stor database som indeholder passwords og tilhørende hashes. Der findes tabeller med kendte ord og korte tilfældige passwords for de gængse hash algoritmer. Og bemærk at kloge hoveder har fundet måder at gemme sådanne tabeller uden at gemme alle rækker. Sådanne tabeller er også kendt som rainbow tables.

Brute force består i at generere alle mulige passwords (med en given længde) og hashe dem og se om hash matcher.

**Hashing og salt - lidt beregning**

Ovenstående kan illustreres ved at beregne gennemsnitlig tid for at finde password i forskellige scenarier.

Der antages at m er 10000, n er 10 og [] er [A-Za-z0-9] (alternativt n er 8 og [] er [A-Z0-9]).

Password gemt plain text:

| | metode | tid |
|---|---|---|
| F1 KWPW | aflæs 1 | <1 sekund |
| FS KWPW | aflæs 1 | <1 sekund |
| FA KWPW | aflæs m | <1 sekund |
| F1 RPW(n,[]) | aflæs 1 | <1 sekund |
| FS RPW(n,[]) | aflæs 1 | <1 sekund |
| FA RPW(n,[]) | aflæs m | <1 sekund |
| F1 alt RPW(n,[]) | aflæs 1 | <1 sekund |
| FS alt RPW(n,[]) | aflæs 1 | <1 sekund |
| FA alt RPW(n,[]) | aflæs m | <1 sekund |

Ingen kommentarer er nødvendige.

Password gemt hashed uden salt:

| | metode | tid |
|---|---|---|
| F1 KWPW | slå op 1 | <1 sekund |
| FS KWPW | slå op 1 | <1 sekund |
| FA KWPW | slå op m | 1 minut 40 sekunder |
| F1 RPW(n,[]) | sizeof([])^n/2 hashes | 13.3 år |
| FS RPW(n,[]) | sizeof([])^n/2/m hashes | 11.6 timer |
| FA RPW(n,[]) | m*sizeof([])^n/4 hashes | 66.5 tusind år |
| F1 alt RPW(n,[]) | slå op 1 | <1 sekund |
| FS alt RPW(n,[]) | slå op 1 | <1 sekund |
| FA alt RPW(n,[]) | slå op m | 1 minut 40 sekunder |

Opsummering:
* kendte ord passwords er meget sårbare
* korte tilfældige passwords er meget sårbare
* lange tilfældige passwords er delvist sårbare

Password gemt hashed med salt - samme salt for alle brugere:

| | metode | tid |
|---|---|---|
| F1 KWPW | 1000000/2 hashes | <1 sekund |
| FS KWPW | 1000000/2/m hashes | <1 sekund |
| FA KWPW | m*1000000/4 hashes | 2.5 sekunder |
| F1 RPW(n,[]) | sizeof([])^n/2 hashes | 13.3 år |
| FS RPW(n,[]) | sizeof([])^n/2/m hashes | 11.6 timer |
| FA RPW(n,[]) | m*sizeof([])^n/4 hashes | 66.5 tusind år |
| F1 alt RPW(n,[]) | sizeof([])^n/2 hashes | 23.5 minutter |
| FS alt RPW(n,[]) | sizeof([])^n/2/m hashes | <1 sekund |
| FA alt RPW(n,[]) | m*sizeof([])^n/4 hashes | 81.6 dage |

Opsummering:
* kendte ord passwords er meget sårbare
* korte tilfældige passwords er sårbare
* lange tilfældige passwords er delvist sårbare

Password gemt hashed med salt - forskellig salt for hver brugere:

| | metode | tid |
|---|---|---|
| F1 KWPW | 1000000/2 hashes | <1 sekund |
| FS KWPW | 1000000/2 hashes | <1 sekund |
| FA KWPW | m*1000000/2 hashes | 5 sekunder |
| F1 RPW(n,[]) | sizeof([])^n/2 hashes | 13.3 år |
| FS RPW(n,[]) | sizeof([])^n/2 hashes | 13.3 år |
| FA RPW(n,[]) | m*sizeof([])^n hashes | 133 tusind år |
| F1 alt RPW(n,[]) | sizeof([])^n/2 hashes | 23.5 minutter |
| FS alt RPW(n,[]) | sizeof([])^n/2 hashes | 23.5 minutter |
| FA alt RPW(n,[]) | m*sizeof([])^n hashes | 163 dage |

Opsummering:
* kendte ord passwords er meget sårbare
* korte tilfældige passwords er sårbare
* lange tilfældige passwords er ikke sårbare

**Effekt af at forbyde kendte ord**

Hashing uden eller med salt kan ikke beskytte kendte ord passwords.

De bør derfor forbydes.

Det kan gøres ved hjælp af en af:
* opslag i ordbog
* regel om et mix af store bogstaver, små bogstaver, tal og special tegn
* kun tillade genererede passwords

Den midterste løsning er mest praktisk.

**Effekt af større password længde**

Lad os sammenligne F1 og FA for forskellige password længder stadig med m=10000.

| | tid |
|---|---|
| F1 RPW(8,[A-Za-z0-9]) | 30.3 timer |
| FA RPW(8,[A-Za-z0-9]) | 34.6 år |
| F1 RPW(10,[A-Za-z0-9]) | 13.3 år |
| FA RPW(10,[A-Za-z0-9]) | 133 tusind år |
| F1 RPW(12,[A-Za-z0-9]) | 51.1 tusind år |
| FA RPW(12,[A-Za-z0-9]) | 511 millioner år |

Password bør mindst være 10 tegn lange - gerne længere.

**Effekt af flere valide tegn**

Lad os sammenligne F1 og FA for forskellige set af valide password tegn
stadig med m=10000.

| | tid |
|---|---|
| F1 RPW(10,[A-Z0-9]) | 21.2 dage |
| FA RPW(10,[A-Z0-9]) | 580 år |
| F1 RPW(10,[A-Za-z0-9]) | 13.3 år |
| FA RPW(10,[A-Za-z0-9]) | 133 tusind år |
| F1 RPW(10,[A-Za-z0-9ÆØÅæøå+-_,.?!$%#]) | 132 år |
| FA RPW(10,[A-Za-z0-9ÆØÅæøå+-_,.?!$%#]) | 1.32 millioner år |

Giver ikke helt så store forbedringer som længere passwords, men giver dog noget.

**Effekt af gentage hashing**

Kloger hoveder har bemærket at hurtige hash funktioner ikke er en fordel i denne sammenhæng.

En måde at lave en hash funktion langsommere på er at hashe flere gang.

5 gange:

output = hash ( hash ( hash ( hash ( hash ( input ) ) ) ) )

Lad os sammenligne F1 og FA for m=10000 n=10 og [A-Za-z0-9].

| | gentagelser | tid |
|---|---|---|
| F1 | 1 | 13.3 år |
| FA | 1 | 133 tusind år |
| F1 | 10 | 133 år |
| FA | 10 | 1.33 millioner år |
| F1 | 100 | 1.33 tusind år |
| FA | 100 | 13.3 millioner år |
| F1 | 1000 | 13.3 tusind år |
| FA | 1000 | 133 millioner år |

Bemærk at gentagelse af hashing ikke er gratis, da det også øger CPU forbruget ved helt normal login, så man skal finde et kompromis.

**Flere computere**

Hvis det er meget interessant at finde et password kan der blive sat flere computere på opgaven med at brute force.

| | tid/1 PC | tid/100 PCere | tid/10000 PCere |
|---|---|---|---|
| F1 RPW(8,62) 1 rep | 30.3 timer | 18.2 minutter | 10.9 sekunder |
| F1 RPW(10,36) 1 rep | 21.2 dage | 5.09 timer | 3.05 minutter |
| F1 RPW(10,62) 1 rep | 13.3 år | 48.5 dage | 11.7 timer |
| F1 RPW(10,78) 1 rep | 132 år | 1.32 år | 4.82 dage |
| F1 RPW(10,62) 100 rep | 1.33 tusind år | 13.3 år | 48.5 dage |

```
F1 RPW(10,78) 100 rep   13.2 tusind år  132 år          1.32 år
F1 RPW(12,62) 1 rep     51.1 tusind år  511 år          5.11 år
F1 RPW(12,78) 1 rep     804 tusind år   8.04 tusind år  80.4 år
```

RPW(n,z) er en forkortelse for RPW(n,[z tegn])

Så hvor man måske vil betragte RPW(10,[A-Za-z0-9]) 1 rep som værende godt nok til eksperten.dk, så skal man måske op i RPW(10,[A-Za-z0-9ÆØÅæøå+-_,.?!$%#]) 100 rep eller RPW(12,[A-Za-z0-9]) 1 rep for en internet bank.

10000 PC'ere lyder ikke af meget, men det er altså 10000 PC'ere med highend grafik-kort. Det vil kræve 1 million hr. og fru Jensen PC'ere at gøre det samme.

**Hash algoritmer**

MD5 og SHA-1 er stadig ret udbredte.

NIST anbefalinger er ret klare:

* MD5 har været forældet i årevis
* SHA-1 betragtes som forældet siden 2010
* SHA-256 er godkendt indtil 2030

Vær imidlertid klar over at hash algoritmer kan have forskellige svagheder:

* pre-image attack => bestemme input udfra output
* collision attack => bestemme 2 input med samme output

De fundne svagheder i MD5 og SHA-1 er collision attacks ikke pre-image attack (bortset fra et ikke praktisk anvendeligt pre-image attack på MD5).

Så MD5 og SHA-1 er ikke katastrofale at bruge for passwords.

Men jeg vil alligevel klart anbefale SHA-256 fordi:
1) fund af svagheder i en algoritme øger sandsynligheden for fund af flere svagheder i nærmeste fremtid og det kan godt betyde at succesfulde collision attacks øger sandsynligheden for succesfulde pre-image attacks i nærmeste fremtid
2) det er langt nemmere at verificere:
 - at der ikke bruges MD5 eller SHA-1 i noget kode end:
 - at de steder hvor der bruges MD5 eller SHA-1 er collision attack ikke et problem

**OWASP**

Der eksisterer en organisation som hedder OWASP (Open Web Application Security Project) med det formål at højne sikkerhedsniveauet på internettet.

De anbefaler:
* password længde 10-128 (passphrases bør være 20-128)

* password skal have mindst 1 uppercase, mindst 1 lowercase, mindst 1 digit og
  mindst 1 specialtegn
* brug både et per user salt gemt i databasen og et system salt gemt udenfor databasen
* gem hashed password og salt hver for sig
* gentag hash 64000 gange

https://www.owasp.org/index.php/Authentication_Cheat_Sheet

https://www.owasp.org/index.php/Password_Storage_Cheat_Sheet

Deres anbefalinger er lidt skrappere end mine.

Men husk "Just because I'm paranoid doesn't mean they're not out to get me.".

**Opsummering**

"Must have" for web sites:

* gem passwords hashed med SHA-256
* brug forskellig tilfældig salt med mindst 64 bit for hver bruger
* minimums længde 10 på passwords
* forhindre brug af kendte ord via regler for passwords

"Nice to have" for web sites:

* tillad ikke-engelske bogstaver og special tegn i passwords
* hash 10-100 gange (check PBKDF2 og bcrypt for mere på dette område)

"Must have" for brugere:

* minimums længde 10 på passwords
* brug aldrig kendte ord som password
* brug aldrig samme password på forskellige web sites

**Kommentar af olebole d. 28. dec 2012 | 1**

<ole>

*<| *<| *<| *<| *<| ... Fem nissehuer for den julegave!

Hvor er det dog befriende med en gennemarbejdet, saglig guide efter en lang ørkenvandring af reklamespam og pre-newbie-aspirant-skriblerier. Tak!

Endnu en velforklaret, veldokumenteret og velillustreret guide fra arne_v(id). Den er letforståelig og rigtig god at kunne linke til. Der kommer jævnligt mere eller mindre fantasifulde bud på emnet, så er det rart med en 'nail-hitter'  =)

Tak, fordi du (gang på gang) gør det!

/mvh
</bole>

**Kommentar af rapsac d. 04. jan 2013 | 2**

Rigtig lækker guide.

Super fedt med reelle tal på hvor lang tid det vil tage at brute-force, det sætter lidt det hele i perspektiv; hvor lidt man rent faktisk skal opdatere for at opnå en langt større sikkerhed. Tror måske jeg hopper væk fra MD5 nu :-)

**Kommentar af magic-mouse d. 08. jan 2013 | 3**

Super guide, let at læse let og forstå.

Samtidig er statestikken skræmmende, og et godt eksempel (som man til og med ikke er bange for at vise til sin chef) som argument for hvorfor man skal kigge på sine sikkerheds anordninger.

5 stjerner her fra. Keep up the good work!

**Kommentar af bilbodog d. 20. jul 2015 | 4**

Super guide. Elsker, at der er relateret til virkeligheden. Det sætter et helt andet perspektiv på tingene. Jeg smutter hvert fald fra MD5 nu og benytter mig nok af SHA-256, Hashing og Salt.

Takker og bukker.