Opinnäytetyö (AMK)

Tietojenkäsittely

2018

Antti Pakkanen

ASP.NET YHTEENSOPIVIEN HASH-ALGORITMIEN VERTAILU JA IMPLEMENTAATIO

– Case Visma Tampuuri Oy

TURKU AMK
TURKU UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖ (AMK) | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Tietojenkäsittely

2018 | 30

Antti Pakkanen

ASP.NET YHTEENSOPIVIEN HASH-ALGORITMIEN VERTAILU JA IMPLEMENTAATIO

Mitä enemmän yhteiskunta riippuu tietotekniikasta, sitä tärkeämpää on tietojärjestelmien tietoturva. Käytetyin autentikaation muoto on tunnus ja salasana, jonka järjestelmän kehittäjä on salannut käyttäjän tietojen suojelemiseksi. Kuitenkin useasti monet kehittäjät eivät ole tietoisia siitä, miten salasanoja tulisi oikeaoppisesti salata. Hoitamalla käyttäjiensä salasanojen suojauksen oikeaoppisesti, kehittäjä tukkii järjestelmässään suuren tietoturva-aukon.

Opinnäytetyön tarkoituksena oli selventää, miten salasanoja salataan oikeaoppisesti sekä selvittää tämänhetkiset suositukset ja vaatimukset. Työssä tutkittiin, miten hash-salaus eroaa kryptauksesta sekä vertailtiin eri hash-algoritmien soveltuvuutta salasanojen salaukseen. Työssä myös tutkittiin hash-algoritmin implementointia ja tutkittiin, kuinka suuri vaikutus sillä on koko järjestelmään.

Työ toteutettiin tutkimalla asiantuntevien yhteisöjen, kuten OWASP ja IETF -organisaatioiden tuottamaa materiaalia. Nämä materiaalit sisälsivät kirjallisia suosituksia ja säännöksiä, jonka perusteella vertailtiin eri hash-algoritmien soveltuvuutta salasanojen salaukseen.

Salasanojen salauksesta tuotettiin toimiva ratkaisu ASP.NET-pohjaiseen järjestelmään. Tämä implementaatio oli tehty tutkimustulosten perusteella. Työ toi toimeksiantajalle ja sen parissa työskenteleville osapuolille osaamista oikeaoppisesta salasanojen salauksesta sekä salasanojen nykystandardeista.

ASIASANAT:

algoritmi, funktio, hash, salasana, autentikaatio, salaustekniikka, tietoturva

BACHELOR´S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Information Technology

2018 | 30

Antti Pakkanen

COMPARISON & IMPLEMENTATION OF ASP.NET COMPATIBLE HASHING ALGORITHMS

The more society relies on information technology, the more importance information security has in our day-to-day lives. The most widely used authentication method in software development is using a username and password for authenticating the user. Software developers are too often ignorant to what the orthodox way of securing passwords is. By implementing a secure solution for storing user passwords, the developer can fix a great vulnerability in these applications.

The purpose of this thesis is to clarify to developers what the recommended specifications for storing passwords securely are. This thesis focuses on educating about current regulations and algorithms of the password hashing technology. The analysis goes through comparing different hashing algorithms and implementing an example solution for an ASP.NET-based application. Even though this example is focused on ASP.NET, it is beneficial for any developer to view it.

The material used as source for the study includes knowledge from encryption technology experts and modern studies on hashing technology. In addition, the author discloses his own experiences from a password hashing implementation he worked on for Visma Tampuuri Limited.

KEYWORDS:

algorithm, function, hash, password, authentication, encryption technology, information Security

SISÄLTÖ

KUVAT

SANASTO

ASP.NET	Active Server Pages dotNet, Microsoftin .NET-sovelluskehyksen komponentti, jolla rakennetaan verkkosivustoja
PBKDF	Password-based key derivation function, funktio, jolla voidaan generoida pseudorandomeja avaimia.
C#	C-sharp, .NET-alustalle kehitetty ohjelmointikieli.
OWASP	Open Web Application Security Project, yhteisö, joka tuottaa www-sovelluksia varten niiden tietoturvaa kohentavia materiaaleja.
ASIC	Application Specific Integrated Circuit, mikropiiri, joka on suunniteltu johonkin tiettyyn käyttötarkoitukseen.
HMAC	Hash-Based Message Autchetication Code, viestissä oleva data, joka on tarkoitettu sen eheyden todennukseen.
SHA	Secure Hash Algorithm, standardisoitu ja kaikkein yleisin hash-algoritmi.
PHC	Password hashing competition, kilpailu, jossa valittiin tehokkain hash-funktio.
Brute-force hyökkäys	Hyökkäys tietojärjestelmää vastaan, jossa käydään kaikki mahdolliset salasanat läpi ja näin yritetään selvittää käyttäjän salasanaa.
Dictionary hyökkäys	Brute-forcen muoto, jossa yritettävät salasanat ovat valmiiksi kirjattu listaan joiden oletetaan todennäköisimmin olevan käyttäjän salasana.

Side-channel hyökkäys	Hyökkäys järjestelmää vastaan, joka hyödyntää järjestelmän kokoonpanon tietoa. Näihin kuuluvat esim. virrankulutus ja ajoitustiedot.
IETF	Internet Engineering Task Force, organisaatio, joka vastaa www-protokollien standardoinnista.
RFC	Request for Comments, IETF:n julkaisema internet standardi.
KDF	Key derivation function, funktio, joka generoi yhden tai useamman salaisen avaimen pseudorandomilla funktiolla.

1 JOHDANTO

Opinnäytetyön tavoitteena on saada aikaan mahdollisimman tietoturvallinen implementaatio salasanojen tallennukselle olemassa olevaan ASP.NET -verkkosovellukseen. Ratkaisu tulee olla yhteensopiva nykyisen järjestelmän version kanssa. Tavoitteena on myös selvittää nykyaikaiset standardit salasanojen salaukselle ja soveltaa näitä standardeja implementaatiota rakentaessa.

Heikot ratkaisut salasanojen säilytykselle laskevat koko järjestelmän ja sitä käyttävien asiakkaiden tietoturvaa. Hyökkääjät voivat siepata kirjautumis- tai muita autentikointipyyntöjä ja tätä kautta aiheuttaa käyttäjällä tai järjestelmälle haittaa. Siksi on tärkeää varmistaa, ettei salasanojen kaappaus olisi mahdollista teknisellä tasolla.

Opinnäytetyössä esitellään millaisia kokoonpanoja alan asiantuntijat suosittelevat salasanojen salaukseen ja vertailen eri ratkaisuja keskenään, joista valitsen parhaiten soveltuvan ratkaisun tähän järjestelmään. Samalla kerron mitä eri asioita tulee ottaa huomioon valintaa tehtäessä.

Tutkimusmuoto on tapaustutkimus, joka keskittyy tutkimaan enimmäkseen suosituimpia salauksen ratkaisuja. Tarkoituksena ei ole siis tutkia jokaista mahdollista hash-algoritmia. Tutkimuksessa on myös tarkoitus käytännönläheisesti huomioida, mitä kaikkea järjestelmässä pitää muuttaa, jotta uusi algoritmi voidaan ottaa käyttöön. Tarkoituksena on tuottaa oikeaoppisesti rakennettu salasanojen salaus ASP.NET-pohjaiseen järjestelmään. Implementaation näkemisestä voi olla apua myös muissa ympäristöissä.

Opinnäytetyö käyttää lähteinään tämänhetkisiä asiantuntijoiden suosituksia ja ajankohtaisia tutkimuksia hash-algoritmien käytöstä.

2 SALASANOJEN TALLENNUKSEN NYKYSTANDARDIT

Maailmassa, jossa jatkuvasti murretaan vanhoja salausmenetelmiä ja algoritmejä, tulee valita mahdollisimman ajan tasalla oleva algoritmi. Hyvä esimerkki murretusta algoritmistä on suosittu hash-algoritmi SHA-1, joka murrettiin vuonna 2017. Jokainen algoritmi tullaan todennäköisesti murtamaan ajan myötä, joten kannattaa valita uusin algoritmi, jota hakkerit eivät ole vielä monta vuotta yrittäneet murtaa. (Whalley 2016)

Salasanoihin sallittuja merkkejä ei tulisi mitenkään rajoittaa järjestelmässä ja niiden maksimipituus tulisi asettaa melko pitkäksi. Liian pitkät salasanat voivat kuitenkin aiheuttaa suoritusongelmia verkkopalvelun palvelinkoneella, joten järkevä raja kannattaa silti asettaa. Suosituksena 160 merkkiä on hyvä salasanojen maksimiraja. (Goodin 2013)

Jotkin järjestelmät kryptaavat kaksisuuntaisesti käyttäjiensä salasanat, ja kuten myöhemmin opinnäytetyössä tullaan huomaamaan niin tämä ei ole hirveästi selkokielistä tekstiä turvallisempi menetelmä. Sen sijaan salasanojen salaukseen tulisi käyttää hash-algoritmiä. Jokainen järjestelmään tallennettava salasana tulisi salata valitsemallasi hash-algoritmillä ja ”suolata” satunnaisella merkkijonolla.

Hash-algoritmissa käytetty suola tulisi olla vahvasti satunnaistettua dataa. Suolan tarkoituksena on estää, ettei 2 eri hashia näytä samalta ja lisätä hajanaisuutta salasanaan olematta riippuvainen alkuperäisen salasanan monimutkaisuudesta. (Steven 2018)

Suomen viestintävirasto on laatinut vahvuusvaatimukset luottamuksellisen tiedon suojaamiseen kryptografisesti. Vaatimusdokumentissa määritellään vähimmäisvaatimukset, joita viestintävirastossa toimiva salaustuotteiden hyväksyntäviranomainen käyttää arvioidessaan salaimen soveltuvuutta turvallisuusluokitellun tiedon suojaamiseen.

Dokumentissa on määritelty hash-funktion kryptografinen vahvuus bitteinä riippuen halutusta suojaustasosta. Tämän opinnäytetyön osalta tulee ottaa

huomioon ”tiivistefunktio” eli hash-funktioita koskevat vaatimukset. (Viestintävirasto 2015)

Hash-funktioilta vaaditaan järjestelmän riskiluokasta riippuen joko 128-, 192- tai 256-bittistä vahvuutta, kuten nähdään kuvassa 1. Dokumentissa mainitaan, ettei kuvan 1 vaatimustaulukossa olevat hash-funktiot ole sellaisenaan suositeltavia salasanojen tallennukseen. Sen sijaan viestintävirasto suosittelee scrypt-, bcrypt- tai PBKDF2-algoritmiä. (Viestintävirasto 2015)

Tiivistefunktio, käyttötarkoitus digitaaliset allekirjoitukset ja "hash-only" -sovellukset

Tiivistefunktiolla on törmäyksettömyysvaatimus.

Kansallinen suojaustaso/kryptovahvuus	ST IV	ST III	ST II
kryptografinen vahvuus törmäyksettömyys-hyökkäystä vastaan bitteinä	128	192	256
algoritmi: SHA-2	SHA-256	SHA-384	SHA-512

Tiivistefunktio, käyttötarkoitus HMAC, avainten ja satunnaislukujen generointi

Tiivistefunktio, kun suojaustarve on vain alkukuvaan.

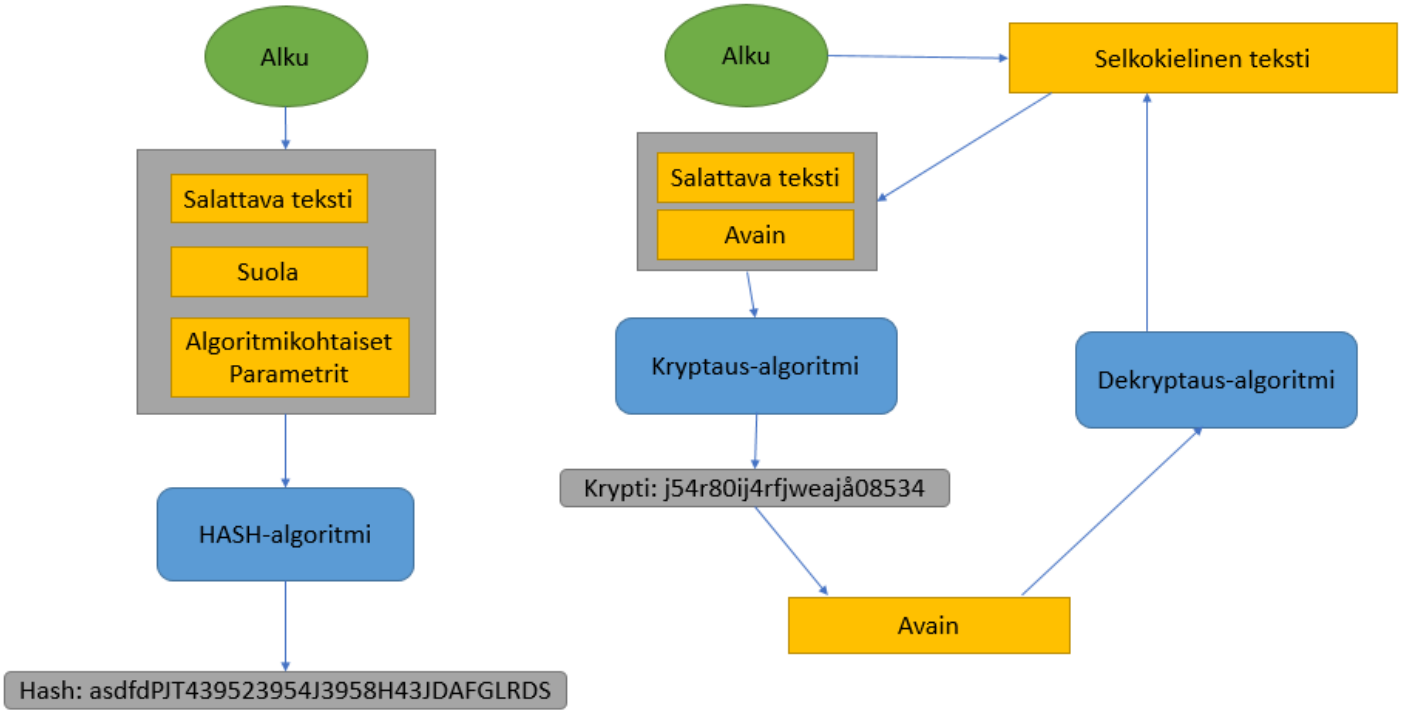
Kansallinen suojaustaso/kryptovahvuus	ST IV	ST III	ST II
kryptografinen vahvuus alkukuvahyökkäystä vastaan bitteinä	128	192	256
algoritmi: SHA-2	SHA-224	SHA-224	SHA-256

Kuva 1. Hash-funktioiden törmäyksettömyysvaatimukset. (Viestintävirasto 2015)

3 HASH JA MUUT SALAUSMENETELMÄT

Usein hash-algoritmilla salaus sekoitetaan perinteiseen avaimella toimivaan kryptaukseen. Nämä kaksi ovat kuitenkin täysin eri lajia. Kryptatun merkkijonon saa selkokieliseksi, mikäli käyttäjällä on siihen tarvittava avain. Tämä mahdollistaa sen, että esimerkiksi unohtuneen salasanan pystyy näyttämään käyttäjälle selkokielisenä, mikäli hänellä on sen kryptin purkuun tarvittava avain. (Jackson 2013)

Kyseessä on siis kaksisuuntainen salaus, mutta hash onkin yksisuuntainen. Kun salaat merkkijonon hash-algoritmilla (esim. salasanan) niin saat tuloksena hyvin saman näköisen merkkijonon mitä kryptikin, mutta ilman avainta sen avaamiseen. Tämä on hash-algoritmin generoima hash, jossa on yleensä ainakin käyttäjän yksilöllinen salasana ja suola sisällä. Ainoa tapa todeta hashin sisältävän tietyn tekstin on hashata teksti uudestaa ja vertailla niistä tulleita hasheja keskenään. Hashaus täytyy tehdä aina samalla algoritmilla, sillä eri algoritmit tuottavat erinäköisiä hasheja. Kryptauksen ja hash-algoritmin erot on havainnollistettu kuvassa 2.



Kuva 2. Hash-algoritmin ja kryptauksen eroavaisuus. (Antti Pakkanen 2018)

Hash-algoritmia kannattaa lähteä valitsemaan vasta perusteellisen perehtymisen ja järjestelmän kartoituksen jälkeen. Implementoinnin alkuvaiheessa havaitut virheet tulevat säästämään kehittäjälle paljon aikaa ja rakennettu toteutus toimii myös varmemmin. Kaikki huomioimattomat seikat voidaan siis joutua korjaamaan jälkeenpäin.

4 HASH-ALGORITMIEN VERTAILU

Algoritmien valinnassa kannattaa aloittaa tutustumalla asiantuntijoiden ehdotuksiin. Open Web Application Security Project (OWASP) on yhteisö, joka tuottaa www-sovelluksia varten niiden tietoturvaa kohentavia materiaaleja. Materiaalit sisältävät dokumentaatiota, työkaluja, videoita ja keskustelufoorumeita www-sovellusten tietoturvasta. Tässä luvussa käydään läpi OWASP-järjestön suosittelemia hash-algoritmeja ja verrataan niitä keskenään.

4.1 Bcrypt

Bcrypt on vuonna 1999 salasanojen salaukseen suunniteltu algoritmi, joka pohjautuu Blowfish-salakirjoitusalgoritmiin. Bcrypt käyttää suolaa ja on tunnetusti hyvä algoritmi rainbow-table- ja brute-force-hyökkäyksiä vastaan. Bcrypt on mahdollista implementoida melkein minkä tahansa ohjelmointikielen kanssa; bcrypt -implementaatioita löytyy käytetyimmistä kielistä. Algoritmi on C# ja ASP.NET yhteensopiva, joten algoritmi on mahdollinen kandidaatti opinnäytetyön yhteydessä toteutettavaan järjestelmään. (Shelley & Stolarczyk 2002)

Bcryptin paras ominaisuus on sen vahvuus dictionary-hyökkäystä vastaan. Dictionary hyökkäyksessä hyökkääjän täytyy generoida järjestelmän omalla hash-algoritmilla valmiit hashit talteen, joita sitten syötetään järjestelmään yrittäen pakottaa sisäänkirjautuminen. Bcrypt hidastaa tätä prosessia huomattavasti, koska hashien generointi kestää niin kauan. (Youtube 2014)

Niels Provos ja David Mazières suunnittelivat bcryptin siten, että algoritmin käyttäjä pystyisi mielivaltaisesti säätämään miten monta kertaa salattava teksti vietäisiin algoritmin läpi. Mikäli käyttäjä säätää kierrosten määrän korkeaksi tulee uuden hashin ja suolan generoinnista paljon hitaampaa. Tämä hitaus on bcryptin suurin vahvuus, sillä se auttaa torjumaan brute-force-hyökkäyksiä, kuten dictionary-hyökkäys.

4.2 Scrypt

Scrypt algoritmi on vuonna 2009 Colin Percivalin kehittämä ja alun perin suunniteltu Tarnap-pilvipalveluun. Algoritmia käytetään Denial of Service -hyökkäysten estämiseen kryptovaluutoissa, joista tunnetuin on Litecoin. Scrypti käyttää salasanapohjaista KDF-funktiota ja algoritmi on suunniteltu olemaan raskas suorittaa. Pääasiassa raskas eli laskennallisesti intensiivinen algoritmi tarkoittaa, että brute-force -hyökkäykset eivät ole yhtä tehokkaita näitä algoritmeja vastaan. Scrypt on laskennallisesti intensiivinen pääasiassa tämän suuren muistinkäytön takia.

Scryptin suuri muistintarve tulee pseudorandomisti generoiduista merkkijonoista, joita luodaan algoritmissa valtavia määriä. Algoritmin muodostama hash luodaan tästä pseudorandomista datasta, joten ne täytyy säilyttää tietokoneen keskusmuistissa algoritmin suorituksen aikana. Pseudorandomina funktiona algoritmissa käytetään HMAC-SHA256-funktiota. (Percival 2009)

Toisin kuin bcrypt, scrypt ottaa parametreiksi muutakin kuin vain salattavan tekstin, suolan ja kierrosten määrän. Scrypt-algoritmiin voidaan syöttää myös laskutoimitukseen varattu prosessori- ja muistiresurssit, suoritettavien instanssien määrä, muodostettavan avaimen pituus, käytettävän hashin pituus, Smix-funktion avaimen pituus sekä käytettävän muistilohkon koon. (Valsorda 2017)

Scrypt on siis suuri muistisyöppö, mutta sen parametrejä voidaan tarvittaessa säätää keventämään resurssien varausta. Scryptin huonoin puoli on sen kovakoodattu HMAC-funktio, joka on HMAC-SHA256. Tätä funktiota ei siis pystytä kehittäjän osalta muuttamaan, mikäli halutaan keventää tai hidastaa salauksen prosessia. Scryptiin löytyy C# ja ASP.NET yhteensopivia kirjastoja. (Brady 2017)

4.3 PBKDF2

Scryptin kaltaisesti PBKDF2 on salasanapohjainen KDF-funktio ja sitä käytetään osana Scryptin algoritmia. PBKDF2 eroaa edeltäjästään, PBKDF1-algoritmistä siten, että PBKDF2 mahdollistaa algoritmin käyttäjän valita omavaltaisesti miten paljon dataa tämä voi generoida. Kuten muutkin KDF-funktiot PBKDF2 käyttää pseudorandomia funktiota generoimaan hashin. PBKDF2 ottaa parametreikseen käytettävän pseudorandomin funktion, salattavan tekstin, suolan, iteraatioiden määrän ja halutun hashin pituuden. Pseudokoodina PBKDF2 voisi näyttää tältä, kun sitä käytetään WPA2-salauksessa: **`DK = PBKDF2(HMAC-SHA1, passphrase, ssid, 4096, 256)`** (IETF rfc8018 2017)

Edellä olevassa esimerkissä on käytetty pseudorandomina funktiona HMAC-SHA1-funktiota. HMAC-SHA1 on toistaiseksi tietoturvallinen, mutta saattaa tulevaisuudessa tulla murretuksi, joten kannattaa käyttää uudempia funktioita kuten HMAC-SHA256-funktiota.

Salasanojen salaukseen ASP.NET toteutuksia löytyy PBKDF2-algoritmillä jopa siihen sisäänrakennetuista luokista. Cryptography-kirjasto sisältää ”Rfc2898DeriveBytes”-luokan, joka käyttää PBKDF2-algoritmiä satunnaisavainten generointiin. Luokka on kuitenkin rajoitettu käyttämään HMAC-SHA1-funktiota salausten generoinnissa. On siis suositeltavaa käyttää kirjastoa, joka tukisi vahvempia HMAC-tyyppejä. CryptSharp-kirjasto on .NET yhteensopiva kirjasto, joka sisältää SHA-256- ja SHA512 -funktiot. (OWASP rfc2989 2015)

Kirjastoja siis löytyy ASP.NET-ympäristöön ja algoritmin tuottama avain on laskennallisesti intensiivinen, joten brute-force-hyökkäykset ovat pieni vaara. Tämä algoritmi ei kuitenkaan ole yhtä raskas mitä Scrypt-algoritmi, sillä se ei käytä suuria määriä muistia toisin kuin Scrypt. Brute-force hyökkäysten kannalta Scrypt olisi siis parempi vaihtoehto. (Karonen 2013)

4.4 Argon2

Argon2 on Alex Biryukovin, Daniel Dinun ja Dmitry Kovratovichin Luxemburin Yliopistossa kehittämä algoritmi. Algoritmi kehitettiin osana Jean-Philippe Aumassonin aloittamaa salasanojen salauskilpailua (PHC), jossa algoritmi valittiin kilpailun voittajaksi. Algoritmi on myös OWASP-yhteisön ensisijaisesti suosittelema algoritmi salasanojen salaukselle. Algoritmista on kolme eri varianttia: Argon2d, Argon2i ja Argon2id. Käydään ensin läpi varianttien yhteiset piirteet ja myöhemmin tarkennetaan varianttien eroavaisuudet. (PHC 2015)

Vaikka Argon2 onkin salasanojen salauskilpailun voittaja, on kuitenkin hyvä pitää mielessä, että kaikkein suosituimmat algoritmit tulevat olemaan suurien murtoyritysten alaisena. Krakkerit luonnollisesti haluavat päästä murtautumaan mahdollisimman moneen järjestelmään ja mikäli algoritmista löydetään heikkous niin krakkerit tulevat sitä hyödyntämään. Hyvänä esimerkkinä tällaisesta tilanteesta on aiemmin laajalti käytetty SHA1-algoritmi, joka murrettiin kuten aikaisemmin luvussa 2 mainittiin.

Argon2 ottaa vastaan useampia parametrejä: salattava teksti, suola, rinnakkaisprosessien määrä, salatun tekstin pituus, käytettävän muistin määrä, iteraatioiden määrä, argon versionumero ja generoidun hashin Argon-variantti (i, d tai id). Algoritmissa on myös mahdollista antaa parametreinä avain ja täytedataa, mutta nämä ovat vapaaehtoisia. (Biryukov, Dinun, Khovarovich 2017)

Argon2-kirjastoja on kehitetty jo useammalla eri kielellä ja .NET-kirjastoja on myös olemassa. Algoritmi on suorituskyvyltään erittäin tehokas, sillä se pystyy käyttämään paljon muistia nopeassa ajassa. Suuri muistinkäyttö hidastaa ASIC-tyyppisiä murtovälineitä eli sovelluskohtaisia mikropiirejä, joilla voidaan yrittää murtaa tietyn sovelluksen salattuja viestejä. (Biryukov, Dinun, Khovarovich 2017)

Argon2 on hyvin skaalautuva algoritmi, sillä muistin ja ajan käyttöä pystyy säätämään tämän parametreistä. Rinnakkaisprosesseja on mahdollista käyttää 224 säikeessä, mutta yli 8 säiettä ei ole suositeltua suoritusongelmien varalta. Algoritmi on myös optimoitu x86-arkkitehtuurille, joka estää sen, ettei

sovelluskohtaisten mikropiirien käyttö olisi nopeampaa tai halvempaa murtaa algoritmilla salattua dataa. (Biryukov, Dinun, Khovarovich 2017)

4.4.1 Argon2d

Argonin d-variantti on optimoitu tilanteisiin, joissa hyökkääjä ei pysty käyttämään suoraan järjestelmän keskusmuistia tai prosessoria. Eli tilanteissa joissa side-channel -hyökkäykset eivät ole mahdollisia ovat optimaalisia paikkoja tälle variantille. Tämä on Argon-varianteista nopein ja on enimmäkseen tarkoitettu suojaamaan backend-palvelimilla ja kryptovaluuttaa kaivavilla koneilla olevaa dataa. (IETF Argon2 2017)

4.4.2 Argon2i

Argonin i-variantti on varianteista hitain sen suuren muistinkäytön takia. Algoritmi on siis massana ajettuna hidas eli se torjuu brute-force -hyökkäyksiä tehokkaasti, mutta silti toimii normaalin käyttäjän näkökulmasta hyvinkin nopeasti. Tästä syystä variantti soveltuu parhaiten salasanojen salaukseen. (IETF Argon2 2017)

4.4.3 Argon2id

Argonin id-variantti on i- ja d-varianttien hybridi. Algoritmi toimii samalla periaatteella mitä i-variantti ensimmäisellä generoinnin kierroksella, jonka jälkeen se omaa d-variantin piirteet. Se siis suojaa side-channel sekä brute-force -hyökkäyksiltä. Variantti on optimoitu yleisimpiin tilanteisiin, jossa hyökkääjä pystyy hyväksikäyttämään hyökkäyskohteensa prosessointiresursseja. (IETF Argon2 2017)

Vaikka id-variantti onkin Argonin päävariantti niin se ei keskity yksinomaan salasanojen suojaamiseen, joka viittaisi siihen, että opinnäytetyön tavoitteiden saavuttamiseen i-variantti soveltuisi paremmin. (IETF Argon2 2017)

5 JÄRJESTELMÄN MUUTOSTEN JA RIIPPUVUUKSIEN KARTOITTAMINEN

Riippumatta työn luonnosta tulee ohjelmistokehittäjän aina ennen ohjelmoinnin ja muiden teknisten muutosten tekemistä, kartoittaa järjestelmän riippuvuudet kehitettävän toiminnon tiimoilta. Autentikointia vaativa järjestelmä tulee vaikuttamaan jokaiseen käyttäjään, joten riippuvuuksien kartoitus on sitäkin tärkeämmässä roolissa. Salasanojen tiimoilta tämä voidaan jakaa viiteen eri osa-alueeseen:

- käyttäjäkokemus
- kirjautuminen
- käyttäjän luonti
- käyttäjätietojen päivitys
- tietokanta.

5.1 Käyttäjäkokemus

Ideaalissa tilanteessa käyttäjä ei joko huomaa lainkaan muutosta tai tämän käyttäjäkokemus paranee. Salasanojen salauksen kannalta muutoksesta kannattaa informoida käyttäjää, jotta tämän ei tarvitse huolestua käyttäjätilinsä turvallisuudesta. Tässä tilanteessa vanhoista salasanoista haluttiin päästä kokonaan eroon tietokannassa ja haluttiin myös kohentaa käyttäjien salasanojen vahvuutta. OWASP suosittelee salasanaksi minimissään 10 merkkiä, jossa on erikoismerkki, numero sekä isoja ja pieniä kirjaimia sekaisin. (Keary 2017)

Jotta vahvemmat salasanat saataisiin käyttöön heti, tulee käyttäjää vaatia uusimaan salasanansa. Tämän jälkeen salasana voidaan salata sekä suolata hash-algoritmilla ja tallettaa tietokantaan. Yksi tapa tähän olisi vaatia seuraavalla kirjautumiskerralla käyttäjää muuttamaan salasanaansa.

Valtaosa järjestelmän käyttäjistä voivat kuitenkin olla varovaisia salasanojensa suhteen ja syystäkin, sillä salasanojen kalasteluyritykset ovat erittäin yleisiä. Näistä syistä johtuen on tärkeää, että käyttäjiä informoidaan jo etukäteen muutoksista. Saadessaan tiedotteen järjestelmän ylläpitäjältä etukäteen on käyttäjäkokemus paljon parempi, kun käyttäjä voi varautua salasanan vaihtoon.

5.2 Kirjautuminen

Järjestelmään kirjautuminen sisältää käyttäjäkokemuksen lisäksi myös autentikoinnin. Kehittäjän vastuulla on varmistaa, että autentikointi tulee toimimaan salasanojen muutoksista huolimatta. Hash-algoritmeissa on usein sisäänrakennettu vertailufunktio, jolla pystytään vertailemaan kahta erillistä hashia. Tätä vertailufunktiota voidaan käyttää autentikointiin. Funktiot palauttavat yleensä boolean-tyyppisen arvon. Kirjautumisen kannalta kehittäjä keskittyy pääasiassa kytkemään uuden verifiointifunktion vanhaan kirjautumismetodiin.

5.3 Käyttäjätietojen luonti ja päivitys

WWW-sovelluksissa tulee myös tunnistaa se, että mitä muutoksia mahdollisesti käyttäjätietojen hallinnoinnin lomakkeille tulee. Salasanan tallennuksen lisäksi, kun nostetaan salasanojen vahvuusvaatimuksia, tulee näitä vaatimuksia myös tukea säännöllisillä lausekkeilla eli regexeillä. Regex mahdollistaa sen, ettei käyttäjä voi asettaa tililleen järjestelmän vaatimuksia rikkovia salasanoja. (OWASP regex 2018)

Aiemmat päivitystoiminnot tulee kytkeä uuden hash-algoritmin funktioihin, jotka luodaan mieluiten erillisen luokan sisälle.

5.4 Tietokannan muutokset

Jokainen autentikointia vaativa järjestelmä tarvitsee tietokannan, minne käyttäjätiedot talletetaan. On olemassa teknologioita, jotka mahdollistavat

käyttäjää kirjautumaan samoilla tunnuksilla moneen eri järjestelmään. Tunnetuin näistä on Google OAuth, joka kuitenkin vaatii, Google-tokenin tallennuksen tietokantaan käyttäjäkohtaisesti. Tätä kutsutaan kertakirjautumiseksi (engl. single sign-on) ja sen ideana on, että vältetään toistuvat autentikointitarkastukset. Käyttäjä pystyy käytännössä kirjautumaan Google-tunnuksillaan kertakirjautumista tukeviin järjestelmiin. Kertakirjautumiselle tulee rakentaa toimiva ratkaisu, mikäli kehittäjä haluaa tämän käyttöön järjestelmässään. (Google 2018)

Tietokanta on siis välttämätön ja perinteisessä autentikoinnissa salasana luetaan tietokannasta. Hash-algoritmilla todennetussa autentikoinnissa tietokantaan tulee lisätä käyttäjän hashille ja suolalle sarakkeet käyttäjätietojen tauluun. Vertailemissamme algoritmeissa suolaa käsitellään byte-array -tyyppisinä, joten tietokantaan tallennettaessa sarakkeen tulee olla binary-tyyppiä. Riippuen algoritmista hash-kolumnin tyyppi tulee olemaan joko tekstiä (varchar) tai binääristä (binary). Kolumnien datan pituuksien rajoituksessa tulee ottaa huomioon algoritmin hashlen-parametrin arvo.

Kauaskatseisena tulisi myös lisätä sarake salasanan vanhentumisen päivämäärälle. Tämä mahdollistaa salasanojen helpon nollauksen. Nollaus mahdollistaa salasanojen salaustekniikan ja vahvuusvaatimusten helpomman muuttamisen.

6 KUSTOMOIDUN HASH-LUOKAN RAKENNUS

Vaikka hash-algoritmeja on useampia niin niiden ohjelmallinen käyttö on hyvinkin samakaltaista. Käyttö eroaa lähinnä syötettävien parametrien määrästä, datatyypistä ja tarkoituksesta.

Tässä luvussa luodaan toimivan luokan salasanojen salaukselle käyttämällä Argon2i-algoritmia. Ohjelmointikielenä käytämme C#-kieltä, mutta valmiita Argon2i-kirjastoja on olemassa myös muilla kielillä. Kun yhteensopiva kirjasto on valittu niin se voidaan tuoda projektiin referenssinä. Esimerkkinä käytämme ”Liphsoft.Crypto.Argon2”-kirjastoa.

6.1 Satunnaislukugeneraattori

Ensimmäinen asia mitä voimakkaiden hashien generointiin tarvitaan, on satunnaistettu suola. Tätä varten tulee luoda funktio, joka generoi näennäisesti satunnaisia merkkijonoja. Microsoftin sisäänrakennettuihin System-kirjastoihin sisältyy satunnaislukugeneraattori-luokka, jolla tämä voidaan toteuttaa.

Kuvassa 3 olevassa funktiossa luodaan 64 alkiota omaava byte-array, jonka jälkeen alkiot täytetään satunnaisella datalla. Alkioiden määrällä määritetään, miten helppoa suola on arvata tai selvittää brute-force-hyökkäyksellä. Tämä on kuitenkin käytännöllistä vain, jos suola on erittäin lyhyt. Suositeltu pituus on 64 alkiota tai enemmän. (Kaliski 2000)

```
///Randomgenerator for salt
private static readonly RandomNumberGenerator Rng = System.Security.Cryptography.RandomNumberGenerator.Create();

/// <summary>
/// Generate individual salt for user
/// </summary>
public byte[] SaltGenerator()
{
    byte[] saltBytes = new byte[64];
    Rng.GetBytes(saltBytes);
    return saltBytes;
}
```

Kuva 3. Satunnaislukugeneraattori C#-kielellä. (Antti Pakkanen 2018)

Näin ollaan luotu yksittäiselle käyttäjälle satunnaisen suolan, jota voidaan käyttää salasanaa luodessa ja verifioidessa. Suolan generointi on tarkoitus suorittaa vain salasanoja luodessa tai päivittäessä niitä.

6.2 Hash-funktio ja sen parametrit

Argon2-algoritmi ottaa vastaan parametrit iteraatioiden määrän (time cost), muistin määrän (memory cost), säikeiden määrän (parallelism), Argon-variantin ja halutun hashin pituuden. Suositeltavat parametrien arvot riippuvat tietokoneen suoritustehosta; kehittäjä haluaa, ettei funktion suoritus ole liian nopea, jotta brute-force -hyökkäykset eivät olisi tehokkaita.

Salasanojen salaukseen tulee Argon-varianttina käyttää Argon2i-algoritmia, sillä tämä on hitain varianteista. Säikeitä tulee käyttää tuplaten sen verran mitä prosessorin ytimiä on allokoitu algoritmille. Eli mikäli kehittäjä haluaisi 4-ytimisen prosessorin käyttävän tämän kaikkia ytimiä niin parametriksi tulisi asettaa 8. Muistinkäytöksi RFC-standardeissa suositellaan 4 Gt backend-autentikointiin ja 1 Gt frontend-autentikointiin. Hashin pituudeksi riittää 16 tavua, mutta se saa olla pidempi. Voimakkailla palvelinkoneilla iteraatioiden määrä tulisi säätää kymmeniin, mutta hitaimmilla koneilla 3-4 iteraatiota riittää. (Schlawack 2015)

Kuvassa 4 luodaan uusi ilmentymä algoritmista parametreilla, jotka ovat järjestelmän resurssien mukaisia. Tämän jälkeen luodaan funktio joka käyttää algoritmia hashin generointiin. Funktio tarvitsee käyttäjän syöttämän salasanan ja tälle generoidun suolan, joka joko generoidaan uutena tai haetaan tietokannasta. Tämä funktio suoritetaan aina autentikoinnin ja tallennuksen yhteydessä.

```
/// <summary>
/// New instance of the Argon2 PasswordHasher
/// </summary>
private PasswordHasher PwHasher = new Liphsoft.Crypto.Argon2.PasswordHasher(30, 4096, 12, Argon2Type.Argon2i, 128);

///<summary>
///Hashes the given password.
///</summary>
///<param name="inputPassword">Password inputted by the user.</param>
///<param name="saltBytes">Generated salt for the user</param>
///<returns>Argon2 hash as string</returns>
public string GenHash(string inputPassword, byte[] saltBytes)
{
    byte[] pwdBytes = Encoding.UTF8.GetBytes(inputPassword);
    string genHash = PwHasher.Hash(pwdBytes, saltBytes);
    return genHash;
}
```

Kuva 4. Argon2 hash-funktio parametreineen C#-kielellä. (Antti Pakkanen 2018)

6.3 Autentikaatio

```
///<summary>
///Verifies the inputted password
///</summary>
///<param name="inputUsername">Username inputted by the user</param>
///<param name="inputPassword">Password inputted by the user.</param>
///<returns>true or false</returns>
public bool VerifyPassword(string inputUsername, string inputPassword)
{
    string dbHash = GetHashByUsername(inputUsername);

    if (PwHasher.Verify(dbHash, inputPassword))
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

Kuva 5. Argon2 Verify-metodin käyttö C#-kielellä. (Antti Pakkanen 2018)

Käyttäjän kirjautumisen yhteydessä haetaan käyttäjän tallennettu hash tietokannasta ja verrataan sitä käyttäjän syöttämään salasanaan. Verifioinnin

yhteydessä salasana hashataan uudestaan samalla suolalla, jolloin saadaan identtiset hashit. Valmiissa kirjastoissa on useimmiten valmiit metodit tätä varten, kuten näemmä kuvassa 5 olevasta esimerkkitoteutuksesta. Metodi hoitaa käyttäjän autentikoinnin, jonka ympärille voidaan rakentaa sisäänkirjautumisen muu toiminnallisuus. Näihin kuuluu muun muassa käyttöliittymän muutokset.

7 KÄYTTÖLIITTYMÄN MUUTOKSET

Salasanojen salausmenetelmien muutokset, tietokannan muutokset, suolageneraattorit ja muut mahdollinen logiikka pystytään rakentamaan ilman, että käyttäjä huomaa mitään. Kuitenkin salausalgoritmeista huolimatta on järjestelmäsi heikoin lenkki salasana sekä käyttäjä. (Kaseya 2018)

Salasanat eivät ole vahva suojausmenetelmä, mikäli käyttäjä ei aseta itselleen monimutkaista salasanaa. OWASP suosittelee minimiksi 10 merkkiä salasanan pituudeksi, mutta valtaosa käyttäjistä käyttää alle 10 merkin salasanaa. Tästä syystä on siis järjestelmän vaatimusten vastuulla pitää vahvaa standardia yllä. (wpengine 2018)

Käyttöliittymä tulee siis muuttumaan lähinnä salasanan vahvuusvaatimusten osalta. Vaatimukset tulee näyttää käyttöliittymässä selkeästi listattuna ja jollakin huomattavalla värillä, kuten punainen. Vaatimukset pystytään rakentamaan salasanan muokkauslomakkeelle käyttämällä säännöllistä lauseketta (regex). Kuten kaikissa säännöllisissä lausekkeissa, tulee kehittäjän määritellä vaatimukset manuaalisesti.

OWASP suosittelee seuraavia vaatimuksia:

- 1 iso ja pieni kirjain
- 1 erikoismerkki
- 1 numero
- minimipituus 10
- vain 2 identtistä merkkiä perättäin.

Regexinä vaatimukset näyttäisivät seuraavalta: ***^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*?&])([A-Za-z\d@$!#%*?&])\2?(?!\2)){10,}$***

Tämä regex sallii kirjaimet a-z sekä A-Z -kirjaimet. Regex sallii 0-9-numerot. Erikoismerkit pitää yksitellen määrittää ja lopuksi tarkistetaan, ettei merkkijonossa ole yli kahta identtistä merkkiä perättäin. (Keary E. 2017)

ASP.NET-ympäristöön implementaatio onnistuu käyttämällä ympäristön validaattoreita. Tässä tapauksessa voidaan käyttää ”RegularExpressionValidator”-validaattoria johon voidaan antaa parametriksi validoitava tekstikenttä ja ”ValidationExpression”-parametriksi haluttu säännöllinen lauseke, kuten näemme kuvasta 6. Salasanan vaatimukset voidaan kertoa validaattorien ”ErrorMessage”-parametriä käyttäen.

```
<asp:Label ID="lblSalasana" runat="server" Text="Salasana:" AssociatedControlID="salasana"></asp:Label>

<asp:TextBox ID="salasana" runat="server" TextMode="Password"></asp:TextBox>

<asp:RegularExpressionValidator ID="salasanaValidator" runat="server" ControlToValidate="salasana" Display="Dynamic"
ErrorMessage="Minimi pituus on 10 merkkiä. Tulee sisältää isoja sekä
            pieniä kirjaimia ja yhden erikoismerkin tai numeron.
            Vain 2 perättäistä identtistä merkkiä sallittu."
ValidationGroup="tunnukset"
ValidationExpression="^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*?&])(([A-Za-z\d@$!#%*?&])\2?(?!\2)){10,}$">
</asp:RegularExpressionValidator>
```

Kuva 6. Asp-validaattorin käyttö säännöllisten lausekkeiden kanssa.

Kun käyttäjä yrittää tallentaa validoitavassa kentässä olevan salasanan tarkistetaan validaattorin avulla vaatimusmäärittelyt. Mikäli vaatimuksia ei täytetä, näytetään ”ErrorMessage”-parametrin sisältö käyttöliittymässä ja estetään tallennus. Kun vaatimuksiin päästään niin tallennus jatkuu normaalisti.

8 LOPUKSI

Aloittaessani opinnäytetyötä lähtötasoni hash-algoritmien tietämykseen oli melko pieni. Ennen työn aloitusta kuvittelin hashien toimivan samalla periaatteella mitä perinteinen kryptaus pariavainten kanssa. Vasta ymmärrettyäni yksisuuntaisen salauksen periaatteen käsitin, että kyseessä on erittäin optimaalinen tekniikka salasanojen salaukselle.

Käytännön työtä aloittaessani tajusin heti alusta, että muutos tulee vaikuttamaan kaikkiin mahdollisiin järjestelmän käyttäjiin, joten järjestelmän riippuvuuksien kartoitus oli erittäin ratkaisevassa roolissa. Kokemuksesta voin sanoa, että edes yhden seikan jättäminen ilman huomiota voi aiheuttaa rutkasti ylimääräisiä lisätöitä. Minulta oli nimittäin työssä jäänyt huomioimatta irrallinen kolmannen osapuolen toiminto, joka minun tulee nyt jälkikäteen korjata.

Myös järjestelmän rajoitukset oli otettava huomioon, mutta pienemmissä määrin, sillä vertailluille algoritmeille tehtyjä valmiita kirjastoja löytyy internetistä paljon. Kirjastojen yhteydessä oli yleensä myöskin liitteenä tarpeeksi kattava ohje sen käyttöön.

Tutkimus ja käytännön työ sai minut suhtautumaan salasanojen salaukseen ja myös yleisesti tietoturvaan paljon vakavammin sekä tarkemmin. Tunnen oppineeni paljon erilaisista hyökkäyksistä sekä salaustekniikoista. Suosittelen kaikkia kehittäjiä käyttämään ainakin muutaman päivän asiaan perehtymiseen, jotka ovat implementoimassa hash-algoritmia ensimmäistä kertaa. Suosittelen myös kertaamaan, mikäli viimeisestä kokemuksesta on yli vuosi.

Vertaillut algoritmit olivat suurimmaksi osakseen tasavertaisia. Suosittelen Argon2i-algoritmia ja mikäli se ei ole mahdollinen niin suosittelen PBKDF2-algoritmia. Argon2 sisälsi algoritmeista eniten valinnanvaraa, koska se oli ainoa algoritmi, joka oli jaettu eri variantteihin. Se myöskin voitti kilpailun, jossa kilpailtiin nimenomaan salasanojen salauksella.

Algoritmien ollessa melko tasavertaisia en lähtisi varta vasten vaihtamaan järjestelmän algoritmia esimerkiksi PBKDF2-algoritmistä Argoniin, vaikka Argon

onkin hitusen parempi. Tämä johtuu siitä, että uuden algoritmin implementointi on keskisuuren työn takana. Vain mikäli järjestelmässäsi on vielä käytössä heikko algoritmi kuten SHA-1 niin lähtisin vaihtamaan salaustekniikkaa. En myöskään suosittele muita SHA-algoritmeja salasanojen salaukseen, koska niitä ei ole suunniteltu yksinomaan salasanojen salaukseen. Algoritmit ovat nopeita ja tällöin altistuvat brute-force -tyyppisille hyökkäyksille. (Rietta 2016)

Salasanat tulee aina suolata ja suola tulee olla myös satunnainen. Suola on dictionary -hyökkäysten kannalta erittäin tärkeää, sillä tämä ehkäisee koko hyökkäyksen aivan täysin. Salasanojen monimutkaisuuden vaatimuksissa seuraisin OWASP:n suosituksia.

LÄHTEET

A.Biryukov, D. Dinun, D. Khovarovich 2017. Argon2: the memory-hard function for password hashing and other application. Viitattu 25.9.2018 https://github.com/P-H-C/phc-winner-argon2/blob/master/argon2-specs.pdf

Brady S. 2017. ScryptPasswordHasher. Viitattu 19.9.2018 https://github.com/scottbrady91/ScottBrady91.AspNetCore.Identity.ScryptPasswordHasher

Goodin D. 2013. Long passwords are good, but too much length can be a DoS hazard. Viitattu 10.9.2018 https://arstechnica.com/information-technology/2013/09/long-passwords-are-good-but-too-much-length-can-be-bad-for-security/

Google 2018. Using OAuth 2.0 to Access Google APIs. Viitattu 1.10.2018 https://developers.google.com/identity/protocols/OAuth2

IETF rfc8018 2017. Password-Based Cryptograhpy specification. Viitattu 20.9.2018 https://tools.ietf.org/html/rfc8018

IETF Argon2. 2017. The memory-hard Argon2 password hash and proof-of-work function. Viitattu 25.9.2018 https://tools.ietf.org/html/draft-irtf-cfrg-argon2-02

Jackson W. 2013 Why salted hash is as good for passwords as for breakfast. Viitattu 18.9.2018 https://gcn.com/articles/2013/12/02/hashing-vs-encryption.aspx

Kaliski B. 2002. Password-Based Cryptography specification. Viitattu 1.10.2018 http://www.ietf.org/rfc/rfc2898.txt

Karonen I. 2013. What is the difference between scrypt and pbkdf2. Viitattu 20.9.2018 https://crypto.stackexchange.com/questions/8159/what-is-the-difference-between-scrypt-and-pbkdf2

Kaseya 2018. The Weakest Link of Any Computer Security System Is the Password. Viitattu 3.10.2018 https://authanvil.com/blog/the-weakest-link-of-any-computer-security-system-is-the-password

Keary E. 2017. Authentication Cheat Sheet. Viitattu 28.9.2018 https://www.owasp.org/index.php/Authentication_Cheat_Sheet

OWASP regex 2018 Validation regex repository. Viitattu 28.9.2018 https://www.owasp.org/index.php/OWASP_Validation_Regex_Repository

OWASP Rfc2898 2015. Using Rfc2898DeriveBytes for PBKDF2. Viitattu 20.9.2018 https://www.owasp.org/index.php/Using_Rfc2898DeriveBytes_for_PBKDF2

Percival C. 2009. Stronger key derivation via sequential memory-hard functions. Viitattu 19.9.2018 http://www.tarsnap.com/scrypt/scrypt.pdf

PHC 2015. Passowrd hashing competition. Viitattu 25.9.2018 https://password-hashing.net/

Rietta F. 2016. Use Bcrypt or Scrypt instead of SHA* for you passwords please! Viitattu 10.10.2018 https://rietta.com/blog/2016/02/05/bcrypt-not-sha-for-passwords/

Schlawack H. 2015. Choosing Parameters. Viitattu 1.10.2018 http://www.ietf.org/rfc/rfc2898.txt

Shelley J. Stolarczyk P. 2002. bcrypt Viitattu 19.9.2018 http://bcrypt.sourceforge.net/

Steven J. 2018. Password Storage Cheat Sheet. Viitattu 10.9.2018 https://www.owasp.org/index.php/Password_Storage_Cheat_Sheet

Whalley A. 2016. SHA-1 certificates in Chrome. Viitattu 10.9.2018 https://security.googleblog.com/2016/11/sha-1-certificates-in-chrome.html

wpengine 2018. Unmasked: What 10 million passwords reveal about the people who choose them. Viitattu 8.10.2018 https://wpengine.com/unmasked/

Valsorda F. 2017. The scrypt parameters. Viitattu 19.9.2018 https://blog.filippo.io/the-scrypt-parameters/

Viestintävirasto 2015. Kryptografiset vahvuusvaatimukset luottamuksellisuuden suojaamiseen - kansalliset suojaustasot Viitattu 15.9.2018 https://www.viestintavirasto.fi/attachments/tietoturva/Kryptografiset_vahvuusvaatimukset_-_kansalliset_suojaustasot.pdf

Youtube 2014. Bcrypt & Password Security - An Introduction Viitattu 19.9.2018 https://www.youtube.com/watch?v=O6cmuiTBZVs