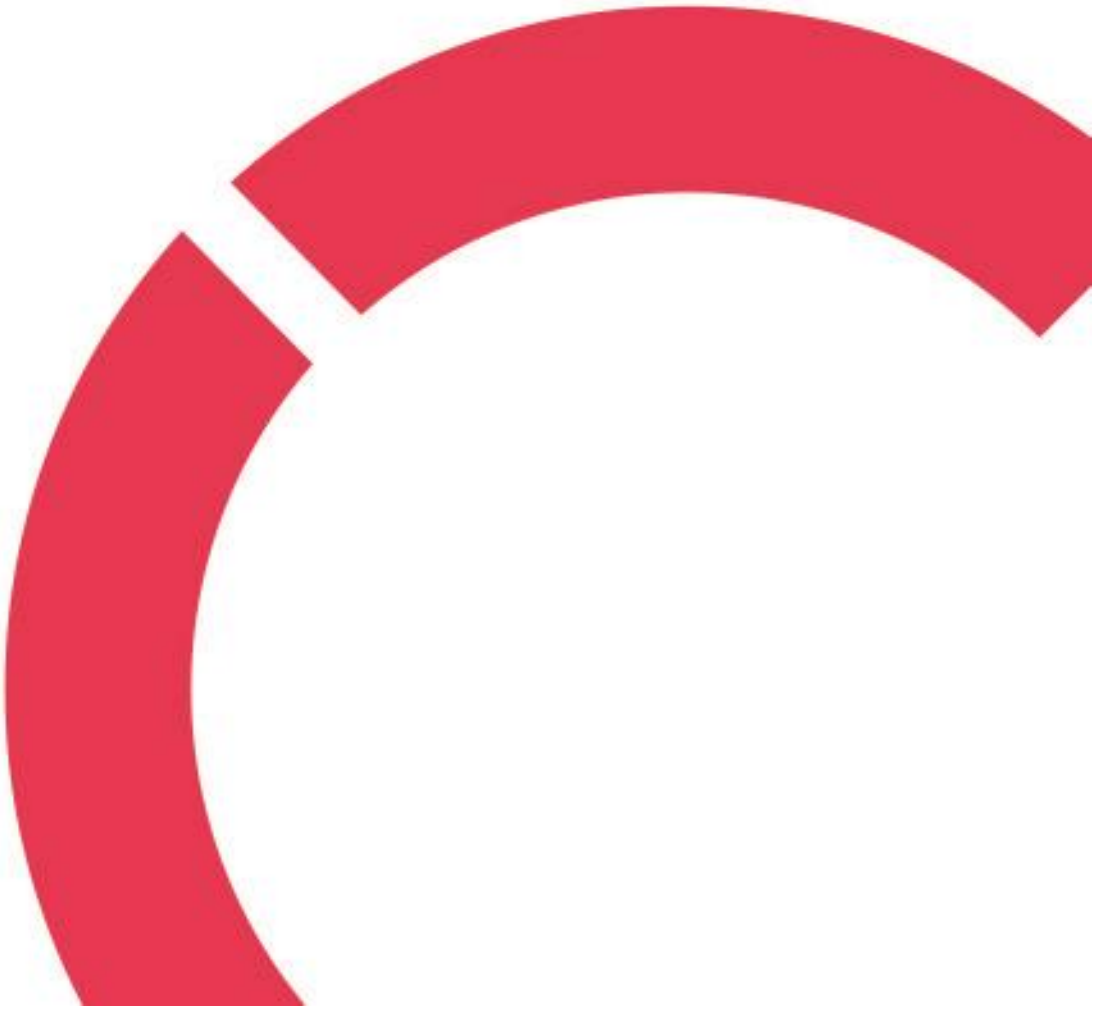
Axel Eloniemi

AJANTASAISTEN KRYPTOGRAFISTEN MENETELMIEN KÄYTÖN MERKITYKSESTÄ

Opinnäytetyö
CENTRIA-AMMATTIKORKEAKOULU
Tieto- ja viestintätekniikan koulutusohjelma
Huhtikuu 2025



TIIVISTELMÄ OPINNÄYTETYÖSTÄ

Centria-ammattikorkeakoulu	**Aika** Huhtikuu 2025	**Tekijä** Axel Eloniemi
Koulutus Insinööri (AMK), tieto- ja viestintätekniikka		☒ AMK ☐ YAMK
Työn nimi AJANTASAISTEN KRYPTOGRAFISTEN MENETELMIEN KÄYTÖN MERKITYKSESTÄ		
Työn ohjaaja Panu Wirkkala		**Sivumäärä** 28 + 3
Työelämäohjaaja Ville Heikkiniemi		

Työn toimeksiantajana toimi Centria-ammattikorkeakoulu. Kyseessä on toiminnallinen opinnäytetyö, jossa tutkittiin käytännönläheisesti tilannetta, jossa yksittäinen hyökkääjä rajoitetuin resurssein on saanut käyttäjätunnuksia ja salasanoja sisältävän tietokannan haltuunsa. Näistä salasanat olivat työssä suojattava tieto. Työssä tutkittiin kolmea erilaista suojaustasoa, joiden suoma suoja perustui erilaisten kryptografisten algoritmien käyttöön. Tutkittiin, miten suojaustasolta seuraavalle siirtyminen vaikutti onnistuneen brute force -hyökkäyksen toteuttamiseen tarvittavaan aikaan, ja toisaalta pyrittiin muodostamaan arvio “riittävästä suojaustasosta”, eli käytännössä selvittämään, millainen kryptografisin menetelmin toteutettava suojaus aiheuttaisi sen, että hyökkääjän näkökulmasta hyökkäystä ei ole enää mielekästä toteuttaa, sillä siihen kuluva aika on liian pitkä.

Pohjan opinnäytetyön tietoperustalle muodostivat oleellisimmilta osin alan kirjallisuus sekä erilaiset artikkelit. Tietoperustassa pyrittiin ensin luomaan hyvä yleiskuva työssä käsiteltävistä aiheista, jonka jälkeen siirryttiin juuri tässä työssä käytettävien menetelmien yksityiskohtaisempaan tarkasteluun. Työn varsin rajatun pituuden vuoksi ei kuitenkaan pyritty perusteelliseen selvitykseen menetelmien perimmäisestä toiminnasta tai niiden käytännön toteutuksesta.

Työn toiminnallisessa osuudessa käytettiin erästä tietoturvatestaukseen tarkoitettua työkalua kahden ensimmäisen demonstraation toteutuksessa. Viimeinen demonstraatio oli enimmäkseen teoreettista tarkastelua, ja siinä arvioiden pohjana käytettiin itse kirjoitetun koodin avulla saatua dataa.

Saatujen tulosten perusteella todettiin, että työssä tutkitun MD5-algoritmin käyttö ei sovellu lyhyiden salasanojen suojaukseen, mutta se voisi antaa teoriassa kohtuullisen suojan pidempiä salasanoja käytettäessä, jos pyritään suojautumaan ainoastaan brute force -hyökkäyksiltä. Tosiasiassa tulee kuitenkin ottaa huomioon myös muut mahdolliset hyökkäystyypit sekä se, ettei MD5-algoritmin käyttöä enää suositella, joten päädyttiin johtopäätökseen, ettei kyseistä algoritmia tule käyttää, jos sen käytön voi välttää. Työssä tutkittujen, Argon2id- ja Advanced Encryption Standard -algoritmien käytön sen sijaan todettiin antavan erittäin hyvän suojan myös lyhyitä salasanoja suojattaessa, ja niinpä niiden käytön todettiin olevan suositeltavaa MD5-algoritmin käytön sijaan.

Asiasanat
kryptografia, tietoturva, tietoturvatestaaminen

ABSTRACT

Centria University of Applied Sciences	**Date** April 2025	**Author** Axel Eloniemi
Degree programme Bachelor of Engineering, Information Technology		
Name of thesis ABOUT THE IMPORTANCE OF USING UP-TO-DATE CRYPTOGRAPHY		
Centria supervisor Panu Wirkkala		**Pages** 28 + 3
Instructor representing commissioning institution or company Ville Heikkiniemi		

Centria University of Applied Sciences was the commissioner of this functional thesis. The scenario being studied in this thesis was one where a singular attacker with limited resources has managed to gain access to a database containing usernames and passwords, from which the passwords were the information to be protected. In the thesis, three different levels of solely cryptography-based protection were studied. The aim of the study was to investigate, how moving from one level of protection to the next affected the time required for pulling off a successful brute-force attack. There was also an intention to create an estimate on what could be considered "a sufficient level of protection", meaning, which kind of cryptographic protection would prolong the attack so that the attack was no longer considered practical by the attacker.

The theoretical framework of the thesis was based largely on literature from the field of study as well as in different kinds of articles. In the theoretical part of the thesis, the big picture was first explored before moving to the details of the exact cryptographic methods used in this thesis. However, because of the rather limited length of the thesis, it was not intended to dive very deeply into the specifics.

In the functional part of the thesis, a certain tool meant for security testing was used in the first two demonstrations. In the third demonstration, the data on which the estimations were based on, was attained by running some self-written code.

Based on the results gained, it was concluded that the MD5 algorithm is not suitable to be used in protecting short passwords, such as those used in the thesis. However, it was found that in theory, when protecting longer passwords, the MD5 algorithm could provide a sufficient level of protection, if the only threat to be protected against was a brute-force attack. However, in reality, it should be considered that there are many other types of attacks, in addition to the fact that the use of the MD5 algorithm is not recommended anymore. Hence, it was concluded that the MD5 algorithm should not be used, if it is viable to not do so. Furthermore, it was pointed out that using the Argon2id and Advanced Encryption Standard algorithms provides a great level of protection even when using shorter passwords, and hence it was noted that their use would be recommended instead of using the MD5 algorithm.

Key words
cryptography, information security, security testing

KÄSITTEIDEN MÄÄRITTELY

ALGORITMI

Kokoelma ohjeita, joita seuraamalla voidaan tuottaa jostakin syötteestä tietty tuloste.

AVOIN LÄHDEKOODI

Tarkoittaa tietokonekoodia, joka on julkaistu (yleensä tietyllä tapaa lisensoituna) vapaasti saataville, tarkasteltavaksi ja muokattavaksi.

BITTI

Tietotekniikassa pienin käytettävä yksikkö, jolla mitataan tiedon määrää. Esitetään usein binäärilukuna, eli nollana tai ykkösenä.

BRUTE FORCE -HYÖKKÄYS

Hyökkäystyyppi, jossa pyritään ns. “raakaa voimaa” käyttäen selvittämään esimerkiksi jokin salasana – merkkijono – jostakin merkkijonoavaruudesta, systemaattisesti kokeilemalla jokaista mahdollista merkkijonoa, kunnes oikea löytyy.

FOR-SILMUKKA

Tarkoittaa koodissa lohkoa, jonka sisällä oleva koodi suoritetaan määritelty määrä kertoja, ennen kuin silmukasta poistutaan.

KRYPTOGRAFIA

Tarkoittaa tieteenalaa, joka tutkii ja harjoittaa tiedon selväkielisen muodon kätkemistä eri keinoin.

SALATEKSTI

Suomennos sanasta “ciphertext”, joka tarkoittaa selvätekstiä, joka on jotakin salausmenetelmää käyttämällä muutettu muotoon, josta ei voida suoraan päätellä sen sisältöä.

SELVÄTEKSTI

Suomennos sanasta “cleartext”, joka tarkoittaa ihmiselle luettavaa, selkokielistä merkkijonoa.

SÄIE

Eräänlainen suoritusyksikkö, joka mahdollistaa eri tehtävien ajamisen rinnakkain.

TAVU

Tarkoittaa tyypillisesti kahdeksaa bittiä.

1 JOHDANTO

Yhä edelleen kasvavissa määrin verkottuneessa maailmassamme miltei jokainen jonkinlaista palvelua tarjoava taho – niin julkinen kuin yksityinenkin – kerää asiakkaistaan tietoja, myös arkaluontoisia sellaisia. Nämä tiedot tallennetaan vaihtelevissa määrin suojatussa muodossa erilaisiin tietokantoihin, joita voidaan ajatella jonkinlaisina digitaalisina varastoina. Tietojen joukossa voi olla esimerkiksi osoitetietoja, henkilötunnuksia, käyttäjätunnuksia, salasanoja, tai vaikkapa potilaskertomuksia. On selvää, että tällaisia tietoja tulisi suojata parhain käytettävissä olevin keinoin väärinkäytöltä, ja niin tyypillisesti tehdäänkin. Siitä huolimatta tietoon tulee jatkuvasti uusia tietovuotoja ja -murtoja, joista toisten takana saattaa olla valtiollinen toimija valtavine resursseineen, kuten epäiltiin esimerkiksi vuonna 2015 olleen, kun vähintään neljän miljoonan Yhdysvaltain valtionhallinnon entisen ja nykyisen työntekijän henkilötiedot varastettiin (Harala 2015), kun taas joissain muissa tapauksissa hyökkääjä saattaa olla yksittäinen henkilö, joka on onnistunut varastamaan tiedot esimerkiksi yksinkertaisesti siitä syystä, että tietoja ei ole käsitelty ja/tai säilytetty turvallisesti. Niin sanottu Vastaamon tapaus on eräs esimerkki jälkimmäiseksi mainitusta. Kyseisessä tapauksessa hyökkääjä sai käsiinsä useamman kymmenen tuhannen ihmisen potilastiedot, koska ne sisältävä tietokanta sijaitsi palvelimella, jonka palomuurisuojaus oli ilmeisesti laiminlyöty täysin (Hämäläinen 2021).

Erinäisten tietokantojen tietoja suojataan – tai niitä ainakin tulisi suojata – monin tavoin, mutta tämän opinnäytetyön aiheena ja tarkoituksena on tarkastella erästä tiettyä tietojen suojauksen ulottuvuutta – tietojen selväkielisen muodon kätkemistä kryptografian avulla. Tarkoituksena on tarkastella käytännönläheisesti ylempänä mainitun kaltaista tapausta, jossa hyökkääjänä toimii yksittäinen henkilö rajoitetuin resurssein – ja niinpä ensimmäiseksi tutkimuskysymykseksi muodostuukin, millainen kryptografisin keinoin saavutettava suojaus voidaan katsoa riittäväksi tällaista uhkaa vastaan, mikäli lähtökohtana on, että hyökkääjä on jo onnistunut saamaan tietokannan käsiinsä. Työssä pyritään määrittelemään “riittävä suojaustaso” pitkälti luomalla arvio siitä, kuinka kauan onnistuneen hyökkäyksen toteuttaminen – eli tietokannan tietojen muuttaminen selväkieliseen muotoon – kunkin valitun suojaustason kohdalla mahdollisesti veisi aikaa, ja toisaalta, millaista aikaa voidaan pitää epäkäytännöllisen pitkänä hyökkääjän kannalta. Tähän suoraan liittyen voidaan muodostaa myös toinen tutkimuskysymys: miten suojaustason parantaminen vaikuttaa onnistuneeseen hyökkäykseen vaadittavaan aikaan?

On selvää, että tietokantoja ylläpitävät henkilöt eivät tyypillisesti julkisesti julista, mitä kryptografisia algoritmeja he tietokantojensa suojaukseen käyttävät, joten on vaikeaa täsmällisesti arvioida, missä määrin mitkäkin algoritmit ovat tällä hetkellä käytössä, ja niinpä täsmällisiä simulaatioita todellisista

reaalimaailman tilanteista on hankalaa luoda. On toisaalta myös epäkäytännöllistä lähteä käytännössä demonstroimaan sellaisten algoritmien suomaa suojaa, jonka murtaminen annetuilla resursseilla, tai edes edistyneintä tämänhetkistä teknologiaa käyttäen, ei käytännössä onnistu – kuten modernien kryptografisten algoritmien kanssa tyypillisesti on (Hollister 2024) – joten käytännön demonstraatioon on valittu sellainen toteutus, että demonstraatio voidaan toteuttaa järkevässä ajassa. Tämän lisäksi on kuitenkin tarpeellista tutkia myös ajantasaisten suositusten mukaisten algoritmien tuomaa turvaa, joten sitä tutkitaan työssä käytännön demonstraation jälkeen teoreettisesti, ja verrataan tuloksia keskenään.

Työn toimeksiantajana on Centria-ammattikorkeakoulu, ja työ tehdään osana hanketta, joka kantaa nimeä “Kyberturvallisuuskoulutuksen ja siihen liittyvän yhteistyön kehittäminen korkeakouluissa”.

2 TIEDOSTA, TIETOTURVASTA, SEKÄ TYÖN MERKITYKSESTÄ

Tietoa, jonka ei soisi joutuvan vääriin käsiin, on monenlaista: kyseessä voivat olla esimerkiksi potilastiedot, tai vaikkapa liike- tai pankkisalaisuudet. Ennen digitaalisen tietojenkäsittelyn aikaa tällaiseen tietoon käsiksi pääseminen on voinut vaatia tietoa muun muassa sen säilyttämispaikasta, sekä kykyä päästä kyseiseen säilytyspaikkaan (esimerkiksi jonkinlaiseen holviin) fyysisesti paikalle anastamaan kyseessä oleva tieto. Nykyaikana taas, kun tietoja syntymäpäiväjuhlien aikatauluista valtionsalaisuuksiin säilytetään pitkälti digitaalisessa muodossa, ei tällainen fyysinen pääsy ole hyökkääjälle useinkaan enää välttämätöntä. Tämän vuoksi kaikessa digitaalisen tiedon säilyttämisessä olisi syytä noudattaa hyviä tietoturvakäytänteitä.

2.1 Tiedosta käsitteenä

Kun puhutaan tietoturvasta, on varsin olennaista tietää ja ymmärtää, mitä turvattavalla “tiedolla” tarkoitetaan, joten seuraavaksi määritelläänkin sen merkitys tämän työn kontekstissa, käyttäen apuna erästä informaatiotieteissä usein käytettävää mallia, niin kutsuttua viisauden hierarkiaa. Kyseisessä mallissa on neljä tasoa, jotka ovat järjestyksessä alimmasta ylimpään data, informaatio, tieto ja viisaus. Datalla tarkoitetaan raaka-aineistoa, informaatiolla taas jotakin datasta jalostettua ja käyttökelpoista, joka vastaa esimerkiksi kysymykseen “kuka” taikka “mitä”. (Nurmi & Pyykönen 2022.) Informaationa voidaan ajatella esimerkiksi jostakin tietokannasta löytyviä tietoja jostakin asiasta taikka henkilöstä – vaikkapa käyttäjätunnus on informaatiota. Informaatiosta taas voidaan edelleen jalostaa mallin mukaista tietoa, joka voidaan nähdä tietynlaisina ohjeina, jotka vastaavat kysymyksiin “kuinka” ja “miten”. Viisaus taas voidaan nähdä vielä edelleen mallin mukaisesta tiedosta jalostuvana, kokonaisvaltaisena ymmärryksenä jostakin asiasta ja seikoista, jotka siihen vaikuttavat. (Nurmi & Pyykönen 2022.) Mallin ylimmät tasot ovat kuitenkin epäolennaisia työn kannalta, eikä niitä siksi käsitellä tässä sen enempää, mutta lienee kuitenkin tarpeellista todeta, että mallin termi “tieto” (englanninkielisessä mallissa “knowledge”) ei vastaa suoraan tietoturvassa suojeltavaa tietoa. Sen sijaan tietoturvassa keskitytään tyypillisesti suojelemaan mallin mukaista informaatiota, ja niinpä tämän työn kontekstissa termiä “tieto” voidaan pitää synonyymina termin “informaatio” kanssa. Huomautetaan myös, että termi “tietoturva” on englanniksi “information security” – suomenkielisellä sanalla “tieto” voidaan siis tarkoittaa sekä informaatiota että mallin mukaista tietoa, mikä tekee termistä sinällään hieman epätäsmällisen.

Joka tapauksessa voidaan ajatella, että tässä työssä mallin mukaista dataa vastaavat salatussa muodossa taikka tiivistemuodossa olevat merkkijonot, jotka eivät sellaisenaan – näennäisen satunnaisina kokoelmina merkkejä ja symboleita – kerro paljoakaan, mutta jotka voidaan jalostaa käyttökelpoiseksi informaatioksi, mikäli salaus puretaan taikka onnistutaan murtamaan, tai vastaavasti tiivistemuotoisen merkkijonon kohdalla onnistutaan muuttamaan takaisin selväkieliseen muotoon. Kuten sanottua, tätä informaatiota on tässä kontekstissa mielekästä kutsua myös tiedoksi.

2.2 Tietoturvasta käsitteenä ja sen kytkeytymisestä opinnäytetyöhön

Lienee syytä myös tarkentaa, että kun puhutaan tietoturvasta, viitataan kolmeen asiaan: tiedon luottamuksellisuuteen, tiedon eheyteen, sekä tiedon käytettävyyteen. Tiedon luottamuksellisuudella tarkoitetaan sitä, että vain niiden tahojen, joilla on siihen oikeus, tulisi voida käyttää tietoa. Tiedon eheydellä vastaavasti tarkoitetaan sitä, että vain niiden tahojen tulisi voida muuttaa tietoa, joilla on siihen oikeus. Tiedon käytettävyydellä taas tarkoitetaan sitä, että tiedon tulisi olla saatavilla sen käsittelyyn oikeutetuille tahoille. (Kyberturvallisuuskeskus 2024a.)

Näistä kolmesta työ liittyy ennen muuta ensimmäiseen – tietojen luottamuksellisuus nimittäin määritelmänsä mukaisesti vaarantuu, mikäli hyökkääjä pääsee käsiksi selväkieliseen tietokannan tietoon, sillä hänellä ei ole oikeutta käyttää kyseisiä tietoja. Voidaan tietysti myös tietyin ehdoin ajatella, että tiedon eheys saattaa niin ikään vaarantua, sillä hyökkääjä saattaisi kyetä myös muuttamaan tietoja, mutta sen, kuten myös tiedon käytettävyydelle mahdollisesti aiheutuvien ongelmien pohtiminen on tämän työn kannalta epäoleellista.

2.3 Työn merkityksestä

Koska ei ole syytä olettaa, että johdannossa kuvaillun kaltaiset hyökkäykset yhtäkkiä vain loppuisivat, on tietoturvallisten toimintatapojen varmistaminen ja niiden ajan tasalla pitäminen hyvin tärkeää – ja eräänä tärkeänä osana tätä voidaan pitää tietokantojen turvaamista erilaisia kryptografisia menetelmiä käyttäen. Jotta tämä olisi mahdollista, tarvitaan kuitenkin tietoa ja koulutusta aiheesta. Monilla yhteiskunnan aloilla tähän on jo havahduttu ja koulutusta lisätty, ja niin on tarkoitus tehdä nyt myös työn toimeksiantajan, Centria-ammattikorkeakoulun toimesta. Valmisteilla on siis uusi, kyberturvallisuutta käsittelevä opintojakso, jonka osana tätä työtä on tarkoitus hyödyntää.

Työ on suunniteltu niin, että opintojakson valmistuessa sen suorittavien opiskelijoiden on mahdollista esimerkiksi toistaa työssä tehdyt testit ja näin todentaa käytännönläheisesti asianmukaisten menetelmien käyttämisen merkitys. Työtä voidaan osaltaan myös käyttää esimerkiksi kirjallisena oppimateriaalina.

3 TYÖSSÄ KÄYTETTÄVISTÄ KRYPTOGRAFISISTA MENETELMISTÄ YLEISESTI

Kuten edellisessä luvussa todettiin, digitaalista aikaa edeltävinä ajanjaksoina, kun tieto oli esimerkiksi musteella fyysiselle paperille kirjoitettuja sanoja, tiedon luottamuksellisuudesta voitiin huolehtia kenties hieman yksinkertaisemmin – tieto voitiin lukita esimerkiksi holviin vain niiden saataville, joilla on holviin avain. Avaimenhaltijat voitiin siis olettaa tahoiksi, joilla on oikeus käsitellä kyseessä olevaa tietoa.

Toinen tapa, jolla tiedon luottamuksellisuudesta on voitu huolehtia ennen digitaalisen tietojenkäsittelyn aikaa, on tiedon salaaminen. Se tarkoittaa sitä, että alkuperäistä selväkielistä viestiä muutetaan niin, ettei alkuperäinen sisältö ole enää luettavissa ennen kuin salaus puretaan, eli ennen kuin salattu viesti muutetaan jälleen selväkieliseen muotoon (Schneier 1996, 1). Esimerkkinä tällaisista varhaisista menetelmistä voidaan mainita kuuluisa, niin kutsuttu Caesar-salakirjoitus (englanniksi “Caesar Cipher”), jossa salattavan tekstin jokainen merkki korvataan sillä merkillä, joka on aakkosissa kolmen merkin päässä alkuperäisestä merkistä oikealle mentäessä (Schneier 1996, 11). Se kuuluu siis toisin sanoen yksinkertaisiin korvaussalausmenetelmiin, eli menetelmiin, joissa jokainen merkki salattavassa viestissä korvataan jollakin toisella merkillä (Hoffstein, Pipher ja Silverman 2008, 2).

Myös digitaalisen tiedon aikana tiedon luottamuksellisuus voidaan pyrkiä varmistamaan käyttämällä salausta, tai sen kaltaista menetelmää, tiivistämistä, jota ei täsmällisesti ottaen voida salaukseksi luokitella, mutta jossa selväkielinen tieto joka tapauksessa muutetaan muotoon, josta alkuperäistä sisältöä ei voida lukea. Caesar-salakirjoituksen kaltaisia menetelmiä ei kuitenkaan enää nykyään käytetä – tai ei ainakaan tulisi käyttää – koska kyseisen kaltainen salaus on ilmeisen helppoa murtaa. Sen sijaan käytetään monimutkaisempia matemaattisia menetelmiä, joissa hyödynnetään esimerkiksi lukuteoriaa, abstraktia algebraa sekä todennäköisyyslaskentaa. (Hoffstein, Pipher ja Silverman 2008, xiii-xiv.)

Tämän työn kannalta merkityksekkäät kryptografiset – eli annetun viestin sisällön kätkemiseen pyrkivät (Hoffstein, Pipher ja Silverman 2008, 2) – menetelmät ovat symmetrinen salaus sekä tiivistefunktiot, ja niitä avataankin seuraavaksi. Tässä luvussa kerrotaan niistä yleisesti, kun taas luvussa 4 käsitellään juuri tässä työssä käytettäviä algoritmeja yksityiskohtaisesti.

3.1 Symmetrinen salaus

Symmetrisellä salauksella tarkoitetaan salausmenetelmiä, joissa sekä salaus että sen purku suoritetaan käyttäen samaa avainta, jonka täytyy siis olla sekä salauksen suorittajan että sen purkajan tiedossa. Tätä avainta voidaan peilata aiemman holviesimerkin avaimeen – mikäli hyökkääjä saa avaimen haltuunsa, on hän kykenevä avaamaan salauksen ja näin pääsemään selväkieliseen tietoon käsiksi (olettaen tietysti, että hänellä on salattu tieto hallussaan tai kyky saada se haltuunsa). Matemaattisesti symmetristä salausta voidaan kuvailla seuraavalla tavalla: olkoot K kaikkien mahdollisten salausavaimien muodostama joukko, k joukosta K mielivaltaisesti valittu salausavain, C kaikkien mahdollisten selvätekstien muodostama joukko, c joukosta C mielivaltaisesti valittu selväteksti, ja S kaikkien mahdollisten salatekstien muodostama joukko – nyt salausavainta k käyttämällä voidaan selvätekstistä c muodostaa eräs joukkoon S kuuluva salateksti s, ja näin suorittaa salaus. Salaus voidaan vastaavasti myös purkaa käyttäen salausavainta k, ja muodostaa salatekstistä s selväteksti c. (Hoffstein, Pipher ja Silverman 2008, 37.)

Eräitä symmetrisiä salausalgoritmeja ovat DES ja AES (Hoffstein, Pipher ja Silverman 2008, 499), joita käsitellään myöhemmin tarkemmin.

3.2 Tiivistefunktiot

Tiivistefunktiolla tarkoitetaan sellaista funktiota F, jolle annetaan syötteenä mielivaltaisen suuri määrä informaatiota (merkitään tätä kirjaimella i), ja joka palauttaa tulosteena määrätyn pituisen bittijonon b (Hoffstein, Pipher ja Silverman 2008, 472). Toisin sanoen, tulosteen koko y ei riipu syötteen koosta x. Tässä sekä x että y ovat jotakin luonnollisia lukuja, jotka edustavat jotakin määrää bittejä. Käytettävästä algoritmista riippuu, mikä y on.

Tiivistefunktiolla tulee olla tietyt ominaisuudet, jotta ne olisivat käyttökelpoisia kryptografisiin tarkoituksiin. Ensinnäkin funktion F(i) arvo tulisi kyetä tuottamaan nopeasti ja helposti, mieluusti lineaarisessa ajassa. Vastaavasti käänteisoperaation suorittaminen, eli bittijonon b muuttaminen takaisin informaatioksi i tulisi olla vaikeaa ja hidasta – eli operaatio tulisi tulla suoritetuksi mieluusti eksponentiaalisessa ajassa. Lisäksi tulisi olla vaikeaa löytää kahta erilaista merkkijonoa M_1 ja M_2 niin, että $F(M_1)$ ja $F(M_2)$ tuottaisivat saman tulosteen – tällaisen ominaisuuden ollessa olemassa sanotaan, että tiivistefunktio on “collision resistant”. (Pieprzyk, Hardjono & Seberry, 472.) Kyseiselle termille ei valitettavasti liene olemassa järkevää ja vakiintunutta suomennosta.

Kuvatun kaltaisiin tiivistefunktioihin lukeutuu esimerkiksi MD5-algoritmi (Pieprzyk, Hardjono & Seberry, 257), jota käsitellään yksityiskohtaisemmin myöhemmin. Käytettäessä sitä, taikka jotakin muuta tiivistefunktiota salasanan tietokantaan tallennuksen yhteydessä niin, että tietokantaan ei tallenneta itse salasanaa, vaan vain sen tiivistemuoto, käytetään tyypillisesti myös niin kutsuttua suolaa. Suolalla tarkoitetaan (mieluusti satunnaista) merkkijonoa, joka voidaan lisätä esimerkiksi salasanan perään, ennen kuin salasana ajetaan tiivistefunktion lävitse (Rankin 2018, 17).

4 TYÖSSÄ KÄYTETTÄVÄT KRYPTOGRAFISET ALGORITMIT SEKÄ OHJELMISTOT

Tässä luvussa esitellään otsikon mukaisesti työssä käytettävät kryptografiset algoritmit sekä oleellisimmat käytettävät ohjelmistot, kuten kahdessa ensimmäisessä suojauskategoriassa käytettävä MD5-algoritmi, ja kyseisen algoritmin suoman suojan testaamiseen käytettävä ohjelmisto, Hashcat. Ensin käydään läpi valitut tiivistealgoritmit, jonka jälkeen siirrytään valitun salausalgoritmin esittelyyn. Viimeisenä esitellään käytettävät ohjelmistot. Kukin algoritmi käydään läpi omassa alaluvussaan, kun taas kaikki työssä käytettävät ohjelmistot esitellään yhdessä ja samassa alaluvussa, johtuen siitä, että ensimmäiseksi mainittujen lyhyt esittely vaatii hieman enemmän tilaa kuin jälkimmäisten, joiden tarkastelu samassa alaluvussa on tarkoituksenmukaisempaa kuin niiden tarkastelu erikseen.

4.1 Käytettävät algoritmit

Seuraavissa alaluvuissa on lyhyesti kuvailtu työssä käytettäviä algoritmeja, sekä myös niistä mahdollisesti löytyviä heikkouksia. Kaksi ensimmäistä alalukua käsittelee MD5- ja Argon2-tiivistealgoritmeja, ja viimeinen taas erästä symmetristä salausalgoritmia, AES:ia. Viimeiseksi mainitun kohdalla on katsottu, että koska toinen symmetrinen salausalgoritmi, DES, liittyy historiallisesti läheisesti AES:iin, on siitä luontevaa ja myös tarkoituksenmukaista kertoa samassa yhteydessä. Kaikkien algoritmien kohdalla käsittely pidetään kuitenkin suhteellisen pintapuolisena työn rajatun pituuden vuoksi, eikä esimerkiksi niiden toimintaa käydä kovinkaan yksityiskohtaisesti läpi. Tarkoituksena on pikemminkin vain muodostaa lukijalle yleiskuva työssä käytettävien algoritmien kirjosta sekä siitä, millaisia ominaisuuksia taikka puutteita niihin liittyy.

4.1.1 MD5-tiivistealgoritmista

MD5 (jossa MD on lyhenne sanoista “Message Digest”) on edeltäjänsä, MD4-algoritmin paranneltu versio, eli tiivistefunktio, joka muuttaa sille annetun syötteen 512-bitin lohkoiksi, jotka on jaettu kuudeksitoista 32-bitin alalohkoksi. Näistä lohkoista luodaan sitten tulosteeksi neljä 32-bitin lohkoa, jotka yhdistetään yhdeksi yhtenäiseksi 128-bitin tiivisteeksi. (Schneier 1996, 435-436.)

MD5:n käyttöä ei kuitenkaan enää suositella muun muassa siksi, että on osoitettu, ettei se täytä kryptografisiin tarkoituksiin käytettäville tiivistefunktioille asetettua vaatimusta (jota sivuttiin aiemmin), jonka mukaan tulisi olla vaikeaa löytää kahta erilaista syötettä niin, että tiivisteeksi saadaan sama arvo. Itse asiassa jo vuonna 2006 osoitettiin, että tällainen niin kutsuttu “törmäys” (collision) voidaan löytää

noin yhdessä minuutissa tavanomaista kuluttajatietokonetta käyttäen. (Turner 2011.) Aikaisemmin se oli kuitenkin laajalti käytössä, ja siitä syystä se on valittu mukaan työhön – demonstroimaan matalaa suojaustasoa, jota se puutteidensa vuoksi nykyisellään tarjoaa.

4.1.2 Argon2-tiivistealgoritmista ja sen Argon2id-versiosta

Argon2, vuoden 2015 Password Hashing Competition –kilpailun voittaja (Password Storage Cheat Sheet), on esimerkiksi salasanojen tiivistemuotoon muuttamiseen – mutta ei pelkästään siihen – tarkoitettu tiivistealgoritmi (Biryukov, Dinu, Khovratovich & Josefsson 2023). Siitä on kolme erilaista versiota, joista versiota Argon2id on suositeltavinta käyttää, sillä se tarjoaa näistä kolmesta parhaiten tasapainotetun suojauksen (Password Storage Cheat Sheet). Myös Biryukov, Dinu, Khovratovich ja Josefsson (2023) suosittelevat sitä ensisijaisena vaihtoehtona kaikissa ympäristöissä.

Argon2 on pitkälti konfiguroitavissa: sen käyttäjä voi sille syötettävien parametrien avulla valita, kuinka paljon muistia algoritmi saa käyttää, kuinka monta kierrosta algoritmin tulee suorittaa, sekä kuinka monta säiettä käytetään. Myös Argon2:en tuottaman syötteen (salasanojen tapauksessa tiivisteen) koon voi käyttäjä valita 4 tavun ja $2^{32} - 1$ tavun väliltä. Huomattavaa on myös, että Argon2:ta käytettäessä tiivistemuotoon muutettavan salasanan tulee olla pituudeltaan enintään $2^{32} - 1$ tavua. (Biryukov, Dinu, Khovratovich & Josefsson 2023.)

4.1.3 AES-salausalgoritmista

Jotta saataisiin parempi kokonaiskuva AES:ista, puhutaan ennen sen esittelyä hetki DES:istä. DES (joka on lyhenne sanoista “Data Encryption Standard”) on siis symmetrinen lohkosalausalgoritmi, mikä tarkoittaa sitä, että se salaa syötettyä tietoa lohko kerrallaan, tarkemmin ottaen 64-bittinen lohko kerrallaan, ja tuottaa kustakin lohkosta 64-bittistä salattua tietoa, salatekstiä. DES käyttää 56-bittistä avainta, jonka varassa koko algoritmin tuoma turva on. (Schneier 1996, 270.) Huomattakoon, että jo DESin julkaisuhetkellä oli selvää, että näin lyhyt avain aiheuttaa ongelmia, sillä hyökkääjä voi yksinkertaisesti kokeilemalla etsiä oikean avaimen kaikkien mahdollisten avainten joukosta, koska avaimia on suhteellisen vähän – 2^{56} mahdollista avainta (Pieprzyk, Hardjono & Seberry, 104-105). Jo vuonna 1999 demonstroitiinkin, kuinka DES-salaus voitiin murtaa näin alle 24 tunnissa (Hoffstein, Pipher ja Silverman 2008, 500).

Tästä ongelmasta päästäänkin suoraan AES:iin (joka on lyhenne sanoista “Advanced Encryption Standard” (Hoffstein, Pipher ja Silverman 2008, 45)), joka on DESin korvaajaksi suunniteltu, niin ikään

symmetrinen lohkosalausalgoritmi. Algoritmin kehitysprosessi oli sinänsä mielenkiintoinen, koska se oli omalla tavallaan epätyypillinen – algoritmin kehitykseen ei osallistunut vain esimerkiksi jokin määritelty valtiollinen taho, vaan kaikki vuonna 1997 NIST:n (National Institute of Standards) järjestämään avoimeen kisaan osallistuneet alan osaajat. Kisan voittavasta algoritmista oli nimittäin tuleva AES, ja sen voitti belgialainen kaksikko Rijndael-salausalgoritmillaan. (Hoffstein, Pipher ja Silverman 2008, 501.) Siksi AES siis tunnetaan myös nimellä Rijndael.

Joka tapauksessa, AES siis DESistä poiketen salaa annettua syötettä 128-bittisinä lohkoina, ja sen avaimen kooksi voidaan valita 128, 192, tai 256 bittiä (Hoffstein, Pipher ja Silverman 2008, 501) DESin 56-bittisen avaimen sijaan. Tämä tarkoittaa sitä, että kun DESissä mahdollisia avaimia oli 2^{56}, on niitä AESin kohdalla parhaimmillaan 2^{256}, mikä on valtava muutos. AES, samoin kuin edeltävän alaluvun Argon2id, on valittu työssä käytettäväksi syistä, joista kerrotaan tarkemmin seuraavassa luvussa.

4.2 Käytettävät ohjelmistot

Seuraavaksi kuvaillaan lyhyesti oleellisimpia työssä käytettäviä ohjelmistoja. Niiden lisäksi työssä käytetään myös Python-ohjelmointikieltä, sen tiettyjä kirjastoja sekä zsh-komentotulkkia, mutta niiden tarkempaa kuvailua tässä ei katsota tarpeelliseksi.

Ensimmäinen työn kannalta oleellinen ohjelmisto on SQLite. Se on avoimen lähdekoodin (License) kevyt tietokantaohjelmisto, joka ei vaadi erillistä palvelinta, vaan sitä käytettäessä koko tietokanta tulee tallennetuksi tavalliseen levytiedostoon. Se on laajalti yhteensopiva erilaisten tietokonearkkitehtuurien ja käyttöjärjestelmien kanssa, mikä lienee eräs syy sille, miksi se on tällä hetkellä maailman käytetyin tietokanta. (About SQLite.)

Kyseinen ohjelmisto valittiin työssä käytettäväksi juuri keveytensä, helppokäyttöisyytensä ja suoraviivaisen toteutuksena vuoksi. Työn kannalta ei tosin ole merkitystä, mitä tietokantaa käytetään, sillä työssä tutkittavat suojaustekniikat eivät ole valitusta tietokannasta riippuvia.

Toinen ohjelmisto, jota työssä käytetään, on nimeltään Hashcat. Se on avoimen lähdekoodin ohjelmisto, joka on tarkoitettu erilaisilla tiivistealgoritmeilla suojattujen salasanojen selvittämiseen (GitHub – hashcat/hashcat: World’s fastest and most advanced password recovery utility.). Tätä ohjelmistoa käytetään myöhemmin demonstroitaessa hyökkäystä tietokantaa vastaan, jonka salasanat on suojattu eri keinoin. Tämä ohjelmisto valittiin, sillä se on nimenomaisesti suunniteltu tällaista hyökkäystä varten, ja sopii siksi erinomaisesti tähän tarkoitukseen.

Lisäksi Python-koodin kirjoittamisessa on käytetty apuna tekoälyä. Tämä ei kuitenkaan tarkoita, että käytetty koodi olisi suoraan tekoälyn generoimaa, vaan ohjeita on lähinnä pyydetty liittyen siihen, kuinka työssä käytettävät kirjastot (joita allekirjoittanut ei ole ennen käyttänyt) toimivat, ja miten niitä käytetään. Tekoälyltä on siis saatu esimerkkikoodia ja selvityksiä sen toiminnasta, joihin pohjaten liitteiden koodi on kirjoitettu. Tiedot on pyritty vielä tarkistamaan kirjastojen dokumentaatiosta. Tekoälyä ei ole käytetty itse opinnäytetyön kirjoittamisessa.

5 TYÖN TOTEUTUKSEN KUVAUS JA TULOKSET

Ennen työn toteutusta asetettiin voimaan tietyt oletukset, joista kerrotaan seuraavaksi. Lähtökohtana oletetaan ensinnäkin, että hyökkääjä toimii yksin, ja että hänellä on käytettävissään vain yksi tavanomainen tietokone. Kaikkien suojaustasojen kohdalla hän on onnistunut selvittämään, mitä tiivistealgoritmia, ja viimeisen suojaustason kohdalla myös mitä salausalgoritmia, on käytetty. Yhden suojaustason kohdalla hän on myös onnistunut saamaan ennalta tietoonsa jotakin olennaista suojattavasta tiedosta – tästä lisää myöhemmin. Lisäksi oletuksena on, että hän on jo saanut käsiinsä tietokannan, joka on kuitenkin niin sanotusti "levossa", eli viimeisen suojaustason kohdalla, jossa on käytetty koko tietokannan kattavaa salausta, on hyökkääjän ensin murrettava salaus, jotta hän voi päästä käsiksi tiivistemuotoisiin salasanoihin, joiden selväkielinen muoto on työssä suojattava tieto.

Työssä tutkitaan siis tietokantaa, joka sisältää kuvitteellisten käyttäjien käyttäjätunnukset ja salasanat. Käyttäjätunnukset on generoitu niin, että kunkin käyttäjän käyttäjätunnus on satunnainen merkkijono, jonka perään on lisätty numero niin, että ensimmäisen käyttäjän kohdalla se on 1, toisen kohdalla 2, ja niin edelleen. Kuten sanottua, tieto, jota työssä pyritään suojaamaan, on käyttäjän salasana, joten käyttäjätunnuksen muoto tai sisältö ei ole oleellinen. Salasanat taas on generoitu niin, että niistä kukin on täysin satunnainen, isoista ja pienistä ASCII-kirjaimista sekä numeroista 0-9 koostuva merkkijono, jonka pituus on kuusi merkkiä. Isoja ASCII-kirjaimia on 26, samoin kuin pieniä, ja lisäksi numeroita on 10, joten kokonaisuudessaan käytössä on $26 + 26 + 10 = 62$ merkkiä. Koska salasanan pituus on kuusi merkkiä, mahdollisia salasanoja on kokonaisuudessaan 62^6. Huomautettakoon, että alun perin tarkoituksena oli käyttää viittätoista merkkiä, jota Kyberturvallisuuskeskus suosittelee tällä hetkellä "nyrkkisääntönään" (Kyberturvallisuuskeskus 2023), mutta alustavien laskelmien tekemisen myötä kävi selväksi, että näin pitkää salasanaa ei voida käytännössä testata järkevässä ajassa, joten merkkien määrää oli välttämätöntä lyhentää huomattavasti, jotta demonstraatio voitiin toteuttaa alusta loppuun saakka. Työssä siis oletetaan, että kuvitteelliset käyttäjät ovat kukin valinneet kuusimerkkisen salasanan, jossa käytettävät merkit kuuluvat edellä mainittuun joukkoon. Oletuksena on myös, että hyökkääjä on onnistunut selvittämään, mitä merkkejä salasanaan saattaa kuulua. Vaikka onkin yleisesti tiedossa, että salasana on sitä turvallisempi, mitä pidempi se on (Kyberturvallisuuskeskus 2024b), pidettiin työssä käytettävät salasanat tarkalleen samanpituisina kaikkien suojaustasojen kohdalla, sillä tutkimuksen kohteena oli nimenomaan erilaisten tiiviste- ja salausalgoritmien tuoma suoja, joten salasanojen pituuden muuttaminen eri suojaustasojen välillä olisi voinut vääristää tuloksia. Kaikki tutkittavat hyökkäykset ovat brute force -hyökkäyksiä, ja niinpä kiinnostus tässä suuntautuu erityisesti siihen, kuinka pitkään teoriassa kuhunkin hyökkäykseen voi enimmillään kulua aikaa, sillä koska hyökkääjä

ei voi tietää, löytyvätkö salasanat ensimmäisten kokeiltavien, vai kenties vasta viimeiseksi kokeiltavien merkkijonojen joukosta, on hänen varauduttava jälkimmäiseen, eli huonoimpaan mahdolliseen vaihtoehtoon.

5.1 Suojaustasot

Suojaustasoja on kaikkiaan kolme, ja ne on pyritty valitsemaan niin, että vaadittava aika onnistuneelle hyökkäykselle kasvaa eksponentiaalisesti mitä korkeampaan suojaustasoon siirrytään, mikäli hyökkäykseen käytettävät resurssit pysyvät vakiona, kuten tässä työssä pysyvät. Niiden kuvaukset ovat seuraavanlaiset:

1. **Heikko suojaus – taso 1:** tietokannassa salasana on muutettu tiivistemuotoon käyttäen MD5-algoritmia ilman suolaa.
2. **Heikko suojaus – taso 2:** tietokannassa salasana on muutettu tiivistemuotoon edelleen käyttäen MD5-algoritmia, mutta tällä kertaa on ennen tiivisteen luontia generoitu suola, joka on lisätty salasanan perään. Tämä yhdistelmä on sitten muutettu tiivistemuotoon. Huomattavaa on kuitenkin se, että generoitu suola on helposti arvattava, sillä se on satunnaisesti generoitu (ei-negatiivinen) 2-numeroinen luku, joten mahdollisia kombinaatioita on vain 10^2, eli 100. Näin lyhyt suola on valittu siitä syystä, että demonstraatioon ei kulu kohtuutonta määrää aikaa. Tässä oletetaan, että hyökkääjä on saanut tietoonsa salasanan olevan kuusi merkkiä pitkä. Lisäksi oletetaan, että hyökkääjä on saanut tietoonsa, että salasana on suolattu, ja suolana on käytetty jotakin (ei-negatiivista) kaksinumeroista lukua. Tämä on perusteltua työn kannalta siksi, että näin olettamalla voidaan rajata kokeiltavia salasanoja, ja näin ollen odottaa demonstraation suorittamiseen kuluvan huomattavasti vähemmän aikaa kuin ilman oletusta.
3. **Tämänhetkisiä suosituksia mukaileva suojaustaso:** tässä suojaustasossa käytettävät algoritmit on valittu OWASPin suositusten mukaan. OWASP on voittoa tavoittelematon säätiö, joka pyrkii toiminnallaan parantamaan ohjelmistojen turvallisuutta (About the OWASP Foundation). Sen julkaisuista eräs, “Password Storage Cheat Sheet”, keskittyy salasanojen oikeaoppiseen ja turvalliseen säilyttämiseen. Sen (Password Storage Cheat Sheet) mukaan Argon2, ja tarkemmin ottaen sen versio, Argon2id, on muutaman muun julkaisussa mainitun algoritmin lisäksi sellainen, jota tulisi ensisijaisesti harkita käytettäväksi salasanojen suojaamisessa. Niinpä tämän suojaustason kohdalla käytetään sitä salasanojen tiivistemuotoon muuttamiseen. Se luo myös automaattisesti suolan (Password Storage Cheat Sheet),

joten suolaa ei tarvitse sen koommin tässä pohtia. Toisessa julkaisussaan, nimeltään “Cryptographic Storage Cheat Sheet”, OWASP (Cryptographic Storage Cheat Sheet) suosittelee levossa olevan tiedon suojaamiseksi ensisijaisesti käytettäväksi AES-algoritmia niin, että valitaan 256-bittinen avain. Tämän suojaustason kohdalla seurataan tätä suositusta, ja koko tietokanta salataan käyttäen mainittua algoritmia. Tämä siksi, että vaikka suojattu tieto, salasana, onkin jo tiivistemuodossa eikä siksi ole hyökkääjälle suoraan luettava, pidetään tietokannan salaamista tämän lisäksi tärkeänä toimenpiteenä, joka tarjoaa lisäsuojaa, ja jonka voidaan katsoa myös tietyllä tapaa ehkäisevän ennalta hyökkäyksiä, sillä salauksen purku voi osoittautua hyökkääjälle hyvin haastavaksi (Pieprzyk, Hardjono & Seberry, 529-530). Salauksen käyttö aiheuttaa siis tässä sen, että ennen kuin hyökkääjä voi päästä tutkimaan tiivistemuotoisia salasanoja, on hänen ensin murrettava salaus.

Suojaustasoista kahden ensimmäisen tuomaa suojaa demonstroidaan käytännössä luomalla kuvauksen kaltainen tietokanta, ja sitten hyökkäämällä sitä vastaan käyttäen edellä kuvailtua Hashcat-ohjelmistoa, tarkoituksena paljastaa suojattava tieto, eli tietokannan sisältämät salasanat selväkielisessä muodossa. Vastaavasti suojaustasoista viimeisen kohdalla sen tuomaa suojaa tutkitaan teoreettisella tasolla, yrittäen luoda mahdollisimman täsmällinen arvio siitä, kuinka kauan tällaisen suojauksen murtamiseen kuluisi aikaa annetuilla resursseilla. Aika on oletettavasti hyvin pitkä, joten sitä ei ole mielekästä lähteä tutkimaan käytännössä. Luvussa 5.3 analysoidaan saatuja tuloksia tarkemmin.

5.2 Hyökkäys

Tämä alaluku on jaettu kolmeen omaan alalukuunsa, joista kussakin käsitellään hyökkäystä kutakin suojaustasoa käyttäen suojattua tietokantaa vastaan. Kuten sanottu, kahdessa ensimmäisessä demonstraatio tehdään käytännössä, ja viimeisessä taas suojauksen tuoma turva osoitetaan teoreettisesti. Hyökkäyksistä kaksi ensimmäistä on toteutettu allekirjoittaneen opinnäytetyön ohjaajan valvonnassa, ja kaikki kokeellinen tehtiin tässä hänen tietokoneellaan (MacBook Pro, jossa M2 Pro -siru ja 16 GB keskusmuistia). Hyökkäykseen tarvittava aika riippuu siitä, kuinka monta tiivistettä voidaan tarkistaa per sekunti, mikä taas riippuu käytettävästä tietokoneesta. Niinpä käytettäessä eri tietokonetta saatetaan päästä eri tulokseen, joten nämä nimenomaiset tulokset pätevät vain tälle kyseiselle tietokoneelle. Vastaavasti viimeisen hyökkäyksen kohdalla kokeellinen data on hankittu käyttäen allekirjoittaneen omaa tietokonetta (Mac Mini, jossa tavallinen M4-siru ja 16 GB keskusmuistia).

Aiemmin esitettyjen oletusten lisäksi kahdessa ensimmäisessä alaluvussa on oletettu, että ennen hyökkäyksen aloittamista hyökkääjä on siirtänyt tiivistemuotoiset salasanat tietokannasta tekstitiedostoihin,

jotta hän voi antaa ne syötteeksi käyttämälleen ohjelmistolle, Hashcatille. Tämä oletus on tehty siksi, että demonstraation toteutuksen yhteydessä nähtiin tarpeettomaksi askeleeksi luoda ensin tietokanta, luoda sitten salasanat ja niiden tiivisteet, lisätä tiivisteet tietokantaan, ja sitten hakea ne tietokannasta ja siirtää tekstitiedostoon. Sen sijaan liitteenä olevassa Python-koodissa (LIITE 1) on luotu salasanat ja suolat, luotu tiivisteet kustakin salasanasta ilman suolaa, ja vastaavasti myös suolan kanssa, jonka jälkeen ne on lisätty omiin tekstitiedostoihinsa.

5.2.1 Hyökkäys heikon suojauksen omaavaa tietokantaa vastaan (Heikko suojaus – taso 1)

Tämä hyökkäys toteutettiin syöttämällä Python-koodin (LIITE 1) luoma tekstitiedosto, joka sisälsi 10 tiivistemuotoista salasanaa, Hashcatille komentoikkunassa. Kussakin selvätekstimuotoisessa salasanassa oli kuusi merkkiä, kuten aiemmin todettiin. Suolaa ei ollut tässä lisätty ennen tiivistemuotoon muuttamista. Hyökkäysdemonstraatiossa käytettiin seuraavaa komentoa:

```
hashcat -m 0 -a 3 -O hashes_no_salt.txt "?1?1?1?1?1?1" -1
"?u?l?d" -i
```

Komennossa valinta “-m 0” tarkoittaa sitä, että kyseessä on MD5-tiiviste (Example_hashes [hashcat wiki]). Seuraavaksi tuleva valinta “-a 3” taas tarkoittaa sitä, että kyseessä on “mask attack”, mikä tarkoittaa sitä, että Hashcatille annetaan jokin “maski”, jonka perusteella Hashcat käy läpi tietyn merkkiavaruuden mahdolliset salasanat (Mask_attack [hashcat wiki]). Sitä seuraava “-O” tarkoittaa, että tässä käytetään optimoitua kerneliä (hashcat v4.0.0 2017). Seuraava “hashes_no_salt.txt” tarkoittaa Hashcatille annettua tekstitiedostoa, joka sisältää murrettavat tiivistemuotoiset salasanat. "?1?1?1?1?1?1" puolestaan tarkoittaa aiemmin mainittua maskia – tässä siis määritellään, että salasana on enintään kuusi merkkiä pitkä, ja jokainen merkki voi olla seuraavassa valinnassa (-1 "?u?l?d") määritellysti joko pieni ASCII-kirjain (“?l”), iso ASCII-kirjain (“?u”) taikka numero (“?d”). Viimeisenä valintana on “-i" (sama asia kuin “--increment”), joka tarkoittaa sitä, että Hashcat kokeilee ensin määritellyssä merkkiavaruudessa yksimerkkiset salasanat, sitten kaksimerkkiset salasanat, ja niin edelleen. (Mask_attack [hashcat wiki].) Viimeinen valinta on otettu mukaan siksi, että vaikka hyökkääjä tietääkin millaisia merkkejä salasana saattaa sisältää, hän ei tiedä, kuinka monta merkkiä pitkä kukin salasana on, joten hänen täytyy aloittaa työnsä yksimerkkisistä salasanoista.

Hyökkäys onnistui, eli Hashcat sai selvitettyä jokaisen sille annetun tiivisteen, jotka oli luotu edellä kuvatulla tavalla generoiduista, suolaamattomista salasanoista. Aikaa tähän kului yhteensä 17 sekuntia, josta 12 sekuntia kului täsmälleen kuusimerkkisten salasanojen läpikäymiseen. Tässä Hashcat kykeni

tarkistamaan parhaimmillaan noin $4.080 \cdot 10^9$ tiivistettä per sekunti, joskin luku vaihteli huomattavasti, ja oli aluksi (yksimerkkisiä salasanoja tarkistettaessa) alimmillaan vain noin 10 000 tiivistettä per sekunti.

Tämän jälkeen testattiin vielä sellaista tapausta, jossa hyökkääjä tietäisi salasanan pituuden olevan täsmälleen kymmenen merkkiä (käytettävän merkistön pysyessä samana), jonka kohdalla Hashcatilta saatiin arvioksi, että tällaisen hyökkäyksen onnistunut toteuttaminen kestäisi noin 6 vuotta ja 109 päivää. Tämän testin kohdalla Hashcat pystyi tarkistamaan noin $4.2225 \cdot 10^9$ tiivistettä per sekunti.

5.2.2 Hyökkäys heikon suojauksen omaavaa tietokantaa vastaan (Heikko suojaus – taso 2)

Tämä hyökkäys on muutoin identtinen yllä kuvatun hyökkäyksen kanssa, mutta tässä ennen salasanan tiivistemuotoon muuttamista sen perään on lisätty satunnaisesti generoitu, kaksinumeroinen suola. Tämä pieni lisäys tehtiin, jotta voitiin tutkia, millä tavalla se vaikuttaa aikaan, joka onnistuneen hyökkäyksen toteuttamiseen kuluu. Tässä hyökkäysdemonstraatiossa käytettiin seuraavaa komentoa:

```
hashcat -m 0 -a 3 -O hashes_with_salt.txt
"?1?1?1?1?1?1?d?d" -1 "?u?l?d" -i
```

Tämä komento on maskia lukuun ottamatta identtinen edellisen hyökkäyksen komennon kanssa, joten samoja valintoja ei selitetä uudelleen. Valittu maski ("?1?1?1?1?1?1?d?d") tarkoittaa tässä sitä, että hyökkääjä arvelee salasanan olevan kuusi merkkiä (joista jokainen voi aiemmin mainittuun tapaan olla pieni tai iso ASCII-kirjain, taikka numero) pitkä, johon on vielä lisätty numeerinen suola, joka on kaksi merkkiä pitkä. Valinta “-i" oli tässä itse asiassa tarpeeton, sillä oletuksena oli, että hyökkääjä tietää salasanan olevan täsmälleen kuusi merkkiä pitkä, ja vastaavasti suolan olevan täsmälleen kaksinumeroinen luku. Siksi oleellista tässä on vain Hashcatin viimeiseen kierrokseen (jossa Hashcat kävi läpi täsmälleen kuusimerkkisiä salasanoja, joiden perään oli lisätty täsmälleen kaksinumeroinen suola) kulunut aika.

Tämäkin hyökkäys onnistui. Hyökkäykseen kului kokonaisuudessaan 23 minuuttia ja 54 sekuntia, josta täsmälleen kuusimerkkisten salasanojen, joiden perään oli lisätty täsmälleen kaksinumeroinen suola, läpikäymiseen kului 21 minuuttia ja 16 sekuntia. Tämän hyökkäyksen kohdalla Hashcatin tarkistamien tiivisteiden määrä per sekunti vaihteli jonkin verran, ja oli suurimmillaan viimeisen kierroksen (kun Hashcat kokeili kuusimerkkisiä salasanoja, joiden perään oli lisätty kaksinumeroinen suola) aikana. Tuolloin nopeus oli noin $4.2215 \cdot 10^9$ kokeiltua tiivistettä per sekunti.

Onnistuneen hyökkäyksen jälkeen testattiin vielä sitä, kauanko aikaa kuluisi sellaisessa tapauksessa, että hyökkääjä tietäisi salasanan olevan pituudeltaan täsmälleen kuusi merkkiä (joista kukin merkki voi edelleen olla pieni tai iso ASCII-kirjain, taikka numero), ja suolan olevan 8-numeroinen merkkijono. Tätä testiä ei tietenkään voitu ajaa loppuun asti, mutta Hashcat antoi arvion, jonka mukaan hyökkäyksessä kestäisi kaikkiaan noin 42 vuotta ja 288 päivää. Tässä Hashcat kykeni kokeilemaan noin $4.2065 \cdot 10^9$ tiivistettä per sekunti. Havaittiin, että Hashcatin laskema arvio on itse asiassa hyvin lähellä arviota koko merkkiavaruuden läpikäymiseen kuluvasta ajasta, sillä tässä tapauksessa merkkiavaruuden koko on $62^6 \cdot 10^8$, joten käyttämällä edellä mainittua tietoa siitä, kuinka monta tiivistettä voidaan tarkistaa per sekunti, päädytään arvioon, että koko merkkiavaruuden läpikäymiseen kuluisi noin $\frac{62^6 \cdot 10^8}{4.2065 \cdot 10^9} \approx 1.3503 \cdot 10^9$ sekuntia, mikä vastaa noin 42 vuotta ja 298 päivää.

5.2.3 Hyökkäys tämänhetkisiä suosituksia mukaillen suojattua tietokantaa vastaan – teoreettinen tarkastelu

Kuten sanottua, tässä hyökkäyksessä tutkitaan tietokantaa, jonka salasanat on muutettu tiivistemuotoon Argon2id-tiivistealgoritmia käyttäen, ja lisäksi koko tietokantatiedosto on salattu käyttäen AES-salausalgoritmia 256-bittisellä avaimella. Kuten niin ikään jo mainittua, tätä hyökkäystä tarkastellaan vain teoreettisella tasolla. Lisäksi luodut arviot ovat vain suuntaa antavia – eri tietokoneen käyttäminen, testikoodin optimoiminen, kokonaan eri ohjelmointikieleen vaihtaminen, tai muiden muutosten tekeminen voi antaa erilaisen arvion.

Koska tässä tapauksessa hyökkääjä on saanut tietokannan haltuunsa salatussa muodossa, on hänen ensin purettava salaus, ennen kuin hän voi tehdä mitään muuta. Tässä osiossa, kuten työssä yleensäkin, tutkitaan brute force -hyökkäystä, joten hyökkääjän on kokeiltava jokaista mahdollista avainta, kunnes oikea löytyy. Pahimmassa tapauksessa, jossa hyökkääjä joutuu käymään jokaisen avaimen läpi ennen oikean löytymistä, joutuu hän kokeilemaan 2^{256} avainta. Se, missä ajassa tämä teoreettisesti olisi mahdollista, voidaan näin ollen laskea, jos tiedetään, kuinka monta avainta voidaan kokeilla – eli kuinka monta salauksen purkuoperaatiota (kukin uudella avaimella) voidaan suorittaa – jonkin tietyn ajan sisällä. Oletetaan, että hyökkääjä kykenee tarkistamaan merkityksettömän lyhyessä ajassa kunkin salauksen purkuoperaation jälkeen, onko salaus purettu onnistuneesti vaiko ei, joten siihen kuluvaa aikaa ei tässä oteta huomioon. Tässä on pyritty luomaan karkea arvio teoreettisesta murtamisajasta kokeilemalla, kuinka monta AES-salauksenpurkuoperaatiota 256-bittistä avainta käyttäen voidaan suorittaa

yhden sekunnin aikana allekirjoittaneen tietokoneella käyttäen Python-ohjelmointikieltä. Liite 2 sisältää tässä käytetyn koodin, jossa ensin luodaan tietokanta, lisätään siihen generoidut käyttäjät, ja sitten salataan tietokanta AES:ia käyttäen. Sen jälkeen suoritetaan for-silmukassa sata kertaa satunnaisen 256-bittisen avaimen generointi (tällä simuloidaan sitä, kuinka hyökkääjä joutuisi generoimaan uuden avaimen ennen jokaista uutta yksittäistä salauksen purkamisen yritystä – tässä ei ole merkitystä sillä, että avain on satunnainen eikä edellistä bittijonoa seuraava bittijono, koska tässä tapauksessa ollaan kiinnostuneita vain aika-arvion luonnista, eikä tosiasiallisesti edes yritetä murtaa salausta), sekä salauksenpurkuoperaatio kutakin luotua avainta käyttäen. Tämä tehdään sata kertaa siitä syystä, että saataisiin hieman parempi arvio siitä, kuinka kauan yhden satunnaisen avaimen generoinnissa ja salauksen purkamisessa (tai sen yrityksessä) keskimäärin kestää.

Tulokseksi saatiin, että aikaa edellä kuvattuun operaatioon kuluu noin 0.000050 sekuntia. Niinpä yhdessä sekunnissa voidaan suorittaa noin $1 \div 0.000050 = 20\,000$ tällaista operaatiota, jonka aikana luodaan satunnainen avain ja yritetään purkaa tietokannan salaus tätä avainta käyttäen. Nyt voidaan helposti laskea, kuinka kauan aikaa hyökkääjällä pahimmassa tapauksessa kuluisi salauksen murtamiseen, mikäli siis oletetaan, että myös hyökkääjä voi suorittaa vain 20 000 mainitun kaltaista operaatiota sekunnissa. Aikaa salauksen murtamiseen kuluisi noin $\frac{2^{256}}{20000} \approx 5.7896 \cdot 10^{72}$ sekuntia, mikä on noin $1.8359 \cdot 10^{65}$ vuotta.

Vaikka jo edellisen tuloksen perusteella on selvää, että tämän salauksen murtaminen annetuilla resursseilla ei ole mahdollista siihen vaadittavan ajan vuoksi, tarkastellaan silti vielä sitä, kuinka kauan hyökkääjällä kestäisi selvittää tietokannan kymmenen Argon2id-tiivisteen (eli tiivistemuotoisten salasanojen) selväkielinen muoto, mikäli hän olisi jotenkin onnistunut murtamaan salauksen kaikesta huolimatta. Huomattavaa tässä on edellisen alaluvun hyökkäykseen verrattuna se, että Argon2id käyttää kyllä suolaa, mutta se on tallennettu tiivisteen mukana selväkielisenä, eikä hyökkääjän tarvitse sitä siis erikseen selvittää. Hyökkääjän on kuitenkin haettava kunkin tiivisteen mukana toimitettu suola, ja jokaisen salasanaehdokkaan kohdalla luotava kymmenen erillistä tiivistettä, joita verrata tietokannasta haettuihin kymmeneen salasanatiivisteeseen. Tähän kuluvaa aikaa tutkitaan liitteestä 3 löytyvän Python-koodin avulla, jossa kahta for-silmukkaa käyttäen tehdään seuraavat toimenpiteet: luodaan satunnainen salasana (jonka merkkimäärä ja käytettävä merkkiavaruus ovat samat kuin aiemminkin), ja sitten kunkin kymmenen salasanatiivisteen kohdalla haetaan tiivisteeseen liittyvä suola, ja luodaan satunnaisesta salasanasta tiiviste käyttämällä tätä suolaa. Sitten verrataan luotua tiivistettä tietokannasta saatuun salasanatiivisteeseen ja tarkistetaan, ovatko ne yksi ja sama tiiviste vaiko eivät – mikäli ne ovat, oikea salasana on löydetty. Kaikki tämä tapahtuu siis yhden ulomman for-silmukan kierroksen aikana,

ja nimenomaan tähän yhteen kierrokseen kuluva aika on tässä tutkittava seikka, jonka avulla pyritään laskemaan, kuinka kauan onnistuneen hyökkäyksen toteutus voisi pahimmillaan kestää. Ulompi for-silmukka suoritetaan 100 kertaa, ja tähän kuluneen ajan avulla lasketaan, kuinka kauan yhden kierroksen ajaminen keskimäärin kestää. Argon2id-tiivisteen muodostamisessa käytettävät parametrit on valittu OWASPin suositusten (Password Storage Cheat Sheet) mukaisesti: käytettävän muistin määräksi on valittu 46MiB, säikeiden määräksi 1, ja kierrosten määräksi 1.

Tulokseksi saatiin, että ulomman for-silmukan yhden kierroksen suorittamiseen kuluu aikaa noin 0.1512 sekuntia. Näin ollen yhdessä sekunnissa voidaan suorittaa noin $1 \div 0.1512 = \frac{1250}{189} \approx 6.61$ tällaista kierrosta. Se taas tarkoittaa, että kun mahdollisten salasanojen kokonaismäärä on edellä mainitulla tavalla 62^6, kaikkien kymmenen tarkasteltavan tiivistemuotoisen salasanan selväkielisen muodon selvittämiseen kuluisi pahimmassa tapauksessa noin $62^6 \div \frac{1250}{189} \approx 8.5882 \cdot 10^9$ sekuntia, eli noin 272 vuotta, jos siis kävisi niin, että koko merkkiavaruus jouduttaisiin käydä läpi ennen kuin kaikki salasanat olisi saatu selville – ja vielä tarkemmin ottaen kyseessä olisi tapaus, jossa kaikki tietokannan salasanat olisivat viimeisten tarkastettavien salasanaehdokkaiden joukossa. Tämä pätee tosin vain siinä tapauksessa, että hyökkääjä tietää salasanan pituuden olevan täsmälleen 6 merkkiä. Jos hän sen sijaan on tietoinen vain käytettävästä merkkiavaruudesta, mutta ei salasanan pituudesta, hän joutuisi käymään läpi ensin kaikki yksimerkkiset salasanat, sitten kaksimerkkiset salasanat ja niin edelleen, kunnes lopulta kuusimerkkiset salasanat läpi käytyään hän olisi onnistunut hyökkäyksessä. Taulukossa 1 on esitetty arviot siitä, kuinka kauan aikaa kuhunkin vaiheeseen kuluisi, jos oletetaan, että edellä laskettu, sekunnissa tarkastettavien salasanojen määrä pysyisi vakiona.

TAULUKKO 1. Arviot eri pituisten salasanojen läpikäymiseen kuluvasta ajasta (Argon2id).

Salasanassa käytettävien merkkien määrä	Arvio kuluvasta ajasta (pahimmassa tapauksessa)
1	$62^1 \div \frac{1250}{189} = 9.3744$ sekuntia
2	$62^2 \div \frac{1250}{189} \approx 5.8121 \cdot 10^2$ sekuntia
3	$62^3 \div \frac{1250}{189} \approx 3.6035 \cdot 10^4$ sekuntia
4	$62^4 \div \frac{1250}{189} \approx 2.2342 \cdot 10^6$ sekuntia
5	$62^5 \div \frac{1250}{189} \approx 1.3852 \cdot 10^8$ sekuntia

Yhteensä aikaa kuluisi pahimmassa tapauksessa kaiken kaikkiaan noin $\sum_{k=1}^{6}\left(62^k \div \frac{1250}{189}\right) \approx 8.7290 \cdot 10^9$ sekuntia, eli noin 277 vuotta, jos hyökkääjä joutuisi käymään läpi kaikki salasanat, joiden pituus on yhdestä kuuteen merkkiä, ja jos olisi edelleen niin, että kaikki tietokannan salasanat löytyisivät vasta viimeisten tarkastettavien salasanaehdokkaiden joukosta. Lisäksi huomataan, että koska testikoodissa luodaan ja tarkistetaan jokaisella kierroksella kymmenen Argon2id-tiivistettä, ja koska yhdessä sekunnissa voidaan suorittaa noin $\frac{1250}{189}$ tällaista kierrosta, tarkoittaa se sitä, että yhdessä sekunnissa voidaan luoda ja tarkistaa noin $10 \cdot \frac{1250}{189} = \frac{12500}{189} \approx 66.14$ tiivistettä. Tämä tieto on varsin hyödyllinen, sillä sitä voidaan verrata myöhemmin MD5-algoritmin vastaavaan.

5.3 Saatujen tulosten analysointi

Ilmeisin ja ensimmäinen asia, joka todettiin tehtyjen demonstraatioiden perusteella, oli se, että pienet, hyökkääjän toimintaa vaikeuttavat muutokset lisäävät kuin lisäävätkin hyökkäykseen tarvittavaa aikaa eksponentiaalisesti, kuten arveltiin. Tämä havaittiin luvuissa 5.2.1 ja 5.2.2 kuvailtujen hyökkäysdemonstraatioiden yhteydessä. Nimittäin luvun 5.2.1 hyökkäyksessä Hashcatin viimeisessä kierroksessa (jossa Hashcat kokeili täsmälleen kuusimerkkisiä salasanoja) kului vain 12 sekuntia, mutta jo kaksinumeroisen suolan lisääminen luvun 5.2.2 hyökkäyksessä aiheutti sen, että kyseisen hyökkäyksen viimeisessä kierroksessa (jossa Hashcat kokeili täsmälleen kuusimerkkisiä salasanoja, joiden perään oli lisätty täsmälleen kaksinumeroinen suola) aikaa kuluikin jo 21 minuuttia ja 16 sekuntia. Toisin sanoen luvun 5.2.2 hyökkäyksen viimeiseen kierrokseen tarvittava aika kasvoi noin 106-kertaiseksi verrattuna luvun 5.2.1 hyökkäyksen viimeiseen kierrokseen. Sama trendi jatkui myös luvun 5.2.3 hyökkäyksen teoreettisessa tarkastelussa, jossa havaittiin, että pahimmassa tapauksessa AES-salauksen murtamiseen kuluisi ensin noin $1.8359 \cdot 10^{65}$ vuotta, jonka jälkeen Argon2id-tiivisteiden murtamiseen pitäisi käyttää vielä noin 277 vuotta.

Havaittu eksponentiaalinen kasvu oli myös syy sille, miksi demonstraatiossa käytettävien salasanojen merkkimäärää vähennettiin alkuperäisen suunnitelman mukaisesta viidestätoista vain kuuteen. Merkkimäärän kasvaessa hyökkäykseen tarvittava aika nimittäin kasvaa hyvin nopeasti erittäin suureksi, kuten taulukosta 2 voidaan nähdä. Taulukon arvioita laskettaessa on oletettu, että nopeus, jolla tiivisteitä voidaan tarkistaa, säilyy jotakuinkin vakiona, ja sen arvoksi on asetettu $4.2225 \cdot 10^9$ tiivistettä per sekunti, mikä oli demonstraatioiden aikana Hashcatin korkein ilmoittama nopeus. Taulukko on hieman

erilainen verrattuna taulukkoon 1, sillä tässä taulukossa tarkastellaan salasanoja, joiden pituus on yhdestä merkistä viiteen merkkiin, yhdestä merkistä kymmeneen merkkiin, ja yhdestä merkistä viiteentoista merkkiin, sen sijaan että tarkasteltaisiin tarkalleen n-pituisia salasanoja.

TAULUKKO 2. Arviot eri pituisten salasanojen läpikäymiseen kuluvasta ajasta (MD5).

Salasanassa käytettävien merkkien määrä	Arvio kuluvasta ajasta (pahimmassa tapauksessa)
1–5	$\sum_{k=1}^{5}\left(\frac{62^k}{4.2225\cdot 10^9}\right) \approx 0.22$ sekuntia
1–10	$\sum_{k=1}^{10}\left(\frac{62^k}{4.2225\cdot 10^9}\right) \approx 6.41$ vuotta
1–15	$\sum_{k=1}^{15}\left(\frac{62^k}{4.2225\cdot 10^9}\right) \approx 5.87 \cdot 10^9$ vuotta

Kuten taulukosta 2 voidaan havaita, on viisitoistamerkkisten salasanojen MD5-tiivisteiden murtaminen pitkälti mahdotonta annetuilla resursseilla, koska auki kirjoitettuna luvun 5.2.1. mukaisen hyökkäyksen toteuttaminen kestäisi pahimmillaan noin 5870000000 vuotta, ja luvun 5.2.2 mukaisen hyökkäyksen toteutus siis vieläkin kauemmin. Niinpä vaikka edellä mainitusti MD5-algoritmin käyttö ei olekaan enää suositeltua, ovat sitä käyttäen luodut salasanatiivisteet silti paremmassa turvassa, mikäli salasanojen haltijoita, käyttäjiä, edellytetään valitsemaan esimerkiksi 15-merkkinen salasana, joka koostuu työssä käytettävän merkkiavaruuden merkeistä, eikä ole muutoin helposti arvattava. Mikäli käyttäjiä edellytetään käyttämään vielä suurempaa merkkiavaruutta, esimerkiksi siis käyttämään lisäksi erikoismerkkejä, on vaikutus vieläkin suurempi. Mikäli oletetaan, että työssä kuvatun kaltainen brute force -hyökkäys on ainoa uhka, jota vastaan on tarve suojautua, voitaisiin MD5-algoritmin käyttöä tällaisten salasanoihin kohdistuvien vaatimusten ollessa voimassa pitää riittävänä suojaustoimenpiteenä. Brute force -hyökkäys on kuitenkin todellisuudessa vain eräs monista uhista, joten on silti syytä seurata suosituksia ja olla käyttämättä MD5-algoritmia, mikäli mahdollista.

Vaikka MD5-algoritmin suomaa suojaa voidaankin näin huomattavissa määrin vahvistaa edellä mainituilla tavoilla, havaittiin myös, että tietokannan salasanojen suojaamisessa käytettävien tiivistealgoritmien valinnalla itselläänkin on suuri merkitys onnistuneen hyökkäyksen toteuttamiseen vaadittavaan aikaan. Sillä kuten mainittua, MD5-algoritmia käytettäessä tiivisteitä voitiin tarkastaa parhaimmillaan noin $4.2225 \cdot 10^9$ sekunnissa, kun taas Argon2id-algoritmin kohdalla vastaava luku oli vain noin $\frac{12500}{189}$ tiivistettä per sekunti. Toisin sanoen, MD5-tiivisteitä voitiin luoda ja tarkastaa noin $(4.2225 \cdot 10^9) \div \frac{12500}{189} = 63844200$ kertaa Argon2id-tiivisteitä nopeammin. Kuten edellä on käynyt ilmi, salasanatiivisteen selvittämiseen kuluva aika riippuu merkkiavaruuden koosta sekä siitä, kuinka monta tiivistettä

voidaan luoda ja tarkastaa jonkin ajan, esimerkiksi sekunnin, sisällä. Näin ollen on selvää, että mitä nopeammin annettua merkkiavaruutta voidaan käydä läpi, sitä nopeammin oikea salasana löydetään. Argon2id-algoritmin käyttäminen MD5-algoritmin sijaan siis vaikeuttaa brute force -hyökkäyksiä hyvin huomattavasti. Tehtyjen laskelmien suhteen on kuitenkin otettava huomioon, että kahta mainittua algoritmia testattiin käyttäen eri työkaluja, sillä Hashcat ei tukenut Argon2id-tiivisteitä. Lisäksi MD5-tiivisteisiin liittyvät testit tehtiin valvotusti työn ohjaajan tietokoneella, kuten mainittua, kun taas Argon2id-tiivisteisiin liittyvät testit tehtiin allekirjoittaneen tietokoneella. Tästä syystä mainitut tarkastusnopeudet eivät ole täysin vertailukelpoisia, mutta antavat silti suuntaa sille, kuinka suuri nopeusero on kyseessä.

Suurin yksittäinen demonstraatioissa havaittu vaikutus oli kuitenkin sillä, oliko tietokanta salattu vaiko ei. Salaamattoman tietokannan kohdalla hyökkääjä kykeni välittömästi hakemaan salasanatiivisteet tietokannasta aloittaakseen niiden murtamisen. Sen sijaan salatun tietokannan AES-salausta ei hyökkääjä kykenisi murtamaan edes koko elinaikanaan, mikäli hyökkäykseen käytettävät resurssit pysyisivät vakiona. Itse asiassa salausta ei saataisi murrettua todennäköisesti yhdenkään ihmisen elinaikana, ottaen huomioon, että maailmankaikkeutemme on nykytiedon valossa ollut olemassa noin 13,8 miljardia, eli $13.8 \cdot 10^9$ vuotta (Hawking 2019, 96), kun testissä käytetyn AES-salauksen murtamiseen laskettiin kuluvan pahimmassa tapauksessa näillä resursseilla $1.8359 \cdot 10^{65}$ vuotta. Toisin sanoen, salauksen murtaminen voisi kestää jopa noin $1.3304 \cdot 10^{55}$ kertaa kauemmin kuin universumi on tähän hetkeen mennessä ollut olemassa. Näin ollen voidaan todeta, että pelkkä AES-salauksenkin käyttäminen antaa varsin riittävän suojan tällaista hyökkäystä vastaan, mikäli hyökkääjän haltuunsa saama tietokanta on oletuksen mukaisesti levossa.

6 JOHTOPÄÄTÖKSET JA POHDINTA

Työn tavoitteena oli käytännönläheisesti tarkastella simuloituja yksittäisen henkilön toteuttamia brute force -hyökkäyksiä eri tavoin suojattuja tietokantoja vastaan, ja saatujen tietojen perusteella pyrkiä luomaan arvio siitä, millaista kryptografisin keinoin toteutettavaa suojausta voidaan pitää riittävänä tällaista uhkaa vastaan – toisin sanoen tarkoituksena oli selvittää, miten hyökkäykseen tarvittava aika voidaan kasvattaa niin suureksi, että hyökkääjä ei käytännössä pysty annetuin resurssein pääsemään käsiksi tietokannan suojattavaan tietoon, salasanoihin. Tähän liittyen pyrittiin myös selvittämään, kuinka työssä määritellyistä suojaustasoista seuraavaan siirtyminen vaikuttaa mainittuun aikaan.

Kuten luvussa 5.3 todettiin, työssä kuvatun kaltaista hyökkäystä vastaan suojautuessa MD5-tiivistealgoritmin käyttöä voidaan teoriassa pitää riittävänä, kun käytetään tarpeeksi pitkiä ja monimutkaisia salasanoja, mutta vain siinä tapauksessa, että tällainen brute force -hyökkäys on ainoa uhka, eikä muunlaisia hyökkäyksiä vastaan ole tarvetta suojautua. Työssä tutkimuksen kohteena olleiden, kuusimerkkisten salasanojen suojaamiseen MD5-algoritmi ei kuitenkaan missään määrin sovellu (paitsi mahdollisesti siinä tapauksessa, että on käytetty pitkää, satunnaista suolaa, joka ei ole hyökkääjän tiedossa), sillä sellaiset hyökkääjä kykenee saamaan selville vähäisilläkin resursseilla hyvin lyhyessä ajassa, kuten havaittiin. Mieluiten kannattaa seurata ajantasaisia suosituksia, ja käyttää MD5-algoritmin sijaan esimerkiksi työssä käytettyä Argon2id-algoritmia, ja lisäksi, jos mahdollista, salata tietokanta käyttäen esimerkiksi AES-algoritmia 256-bittisellä avaimella, kuten työssä on käytetty. Tämä antaa työssä kuvaillulla tavalla erittäin hyvän kryptografisen suojan, ja mielestäni siihen tulisi aina pyrkiä, silloinkin kun näyttää siltä, että laskennallisesti kohtuullinen suoja voitaisiin saavuttaa myös vähäisemmillä toimilla, ei-suositeltavaa algoritmia käyttäen. Mainittujen, suositeltavien algoritmien käyttö suojaa myös lyhyitä (kuten kuusimerkkisiä) salasanoja hyvin, ja on huomattavasti yksinkertaisempaa ottaa käyttöön vahvat kryptografiset suojaustoimet omassa järjestelmässään kuin valvoa, että kaikki käyttäjät varmasti käyttävät tarpeeksi pitkiä, monimutkaisia ja vaikeasti arvattavia salasanoja, vaikka tätäkään puolta ei toki tule unohtaa.

Kuten mainittua, suojaustasot pyrittiin ennen demonstraatioiden toteutusta valitsemaan niin, että edelliseltä suojaustasolta seuraavalle siirtyminen kasvattaisi hyökkäykseen tarvittavaa aikaa eksponentiaalisesti, ja näin tapahtuikin, mutta ajan kasvunopeus oli silti odottamattoman suuri, ja murrettavien salasanojen merkkimäärää jouduttiinkin tästä syystä vähentämään sekä ennen demonstraation toteutusta

että myös sen aikana, päätyen lopulta kuusimerkkisten salasanojen käyttöön. Tämä ei kuitenkaan haitannut, koska saatujen tulosten avulla voitiin laskea arviot myös pidempien salasanojen murtamiseen kuluvasta ajasta.

Käytetyt menetelmät olivat toimivia, joskin yhtenäisempi tapa suorittaa demonstraatiot olisi voinut tuottaa vielä paremmin vertailukelpoisia tuloksia. Toimeksiantaja ei asettanut erityisiä tavoitteita työlle, muuta kuin sen, että työtä voitaisiin hyödyntää työn alussa mainitun, uuden opintojakson yhteydessä. Tämä tavoite on mielestäni täyttynyt varsin hyvin, sillä työssä tehtyjä demonstraatioita voidaan kyseisen opintojakson aikana esimerkiksi toistaa, niitä voidaan muokata eri tarkoitukseen, tai käyttää muun työn ohessa oppimateriaalina.

Kaikkiaan työ oli mielestäni onnistunut ja palvelee tarkoitustaan hyvin.

7 YHTEENVETO

Molempiin tutkimuskysymyksiin onnistuttiin löytämään tyydyttävä vastaus. Todettiin, että ajantasaisten suositusten mukaisten kryptografisten algoritmien käytöllä on kuin onkin hyvin suuri vaikutus siihen, millainen tosiasiallinen suojaustaso voidaan saavuttaa esimerkiksi työssä käsiteltävänä olevan kaltaiselle tietokannalle. AES-salauksen käytön havaittiin olevan erityisen merkityksekäs suoja, mutta tätä havaintoa ei suinkaan kannata ajatella pelkästään tietokantojen suojaamisen kannalta, vaan sen perusteella voidaan yleisestikin todeta, että minkä tahansa suojattavan tiedon kohdalla – oli kyse sitten tietokantaan tallennetusta tiedosta, tekstitiedostosta, kuvasta, tai kokonaisesta tietojärjestelmästä – salaus kannattaa. Sillä vaikka emme voisikaan luottaa siihen, etteivät bittimme päädy vääriin käsiin, voimme kohtuudella luottaa kryptografiaan.

Työ oli varsin opettavainen kokemus niin sisältönsä, kuin myös itse prosessin puolesta. Työn tekeminen antoi myös hyvät lähtökohdat tulevien maisteriopintojeni aloittamiselle, joissa aion suuntautua kyberturvallisuuteen. Vaikka onkin tietysti selvää, että työssä käsiteltävät seikat olivat vain pieni pintaraapaisu aiheeseen, on ensimmäinen askel asiassa kuin asiassa kuitenkin usein eräs merkityksekkäimmistä.

LÄHTEET

About SQLite. SQLite. Saatavissa: https://www.sqlite.org/about.html. Viitattu 19.1.2025.

About the OWASP Foundation. OWASP. Saatavissa: https://owasp.org/about/. Viitattu 19.1.2025.

Biryukov, A., Dinu, D., Khovratovich, D. & Josefsson, S. 2023. *RFC 9106 – Argon2 Memory-Hard Function for Password Hashing and Proof-of-Work Applications.* Saatavissa: https://datatracker.ietf.org/doc/rfc9106/. Viitattu 20.1.2025.

Cryptographic Storage Cheat Sheet. OWASP. Saatavissa: https://cheatsheetseries.owasp.org/cheatsheets/Cryptographic_Storage_Cheat_Sheet.html. Viitattu 19.1.2025.

Example_hashes [hashcat wiki]. Hashcat. Saatavissa: https://hashcat.net/wiki/doku.php?id=example_hashes. Viitattu 17.2.2025.

GitHub – hashcat/hashcat: World's fastest and most advanced password recovery utility. GitHub. Saatavissa: https://github.com/hashcat/hashcat. Viitattu 19.1.2025.

Harala, S. 2015. Jättimäinen tietomurto Yhdysvalloissa – miljoonien virkamiesten tiedot vääriin käsiin. *Yle.* Saatavissa: https://yle.fi/a/3-8047480. Viitattu 20.1.2025.

hashcat v4.0.0. Hashcat. 2017. Saatavissa: https://hashcat.net/forum/thread-6965.html. Viitattu 7.3.2025.

Hawking, S. 2019. *Lyhyet vastaukset suuriin kysymyksiin.* Helsinki: Werner Söderström Osakeyhtiö.

Hoffstein, J., Pipher, J. & Silverman, J. 2014. *An Introduction to Mathematical Cryptography.* Second edition. New York: Springer.

Hollister, S. 2024. Google says its breakthrough quantum chip can't break modern cryptography. *The Verge.* Saatavilla: https://www.theverge.com/2024/12/12/24319879/google-willow-cant-break-rsa-cryptography. Viitattu 4.3.2025.

Hämäläinen, V. 2021. Uudet tiedot: Vastaamon potilaiden tiedot olivat ehkä jopa vuosia suojaamatta netissä – tietoturva-asiantuntija: "Älyvapaata". *Yle.* Saatavilla: https://yle.fi/a/3-11750220. Viitattu 20.1.2025.

Kyberturvallisuuskeskus. 2023. *Salasanat haltuun – Kuka käyttää tiliäsi?* Saatavissa: https://www.kyberturvallisuuskeskus.fi/fi/ajankohtaista/ohjeet-ja-oppaat/salasanat-haltuun. Viitattu 20.1.2025.

Kyberturvallisuuskeskus. 2024. *Tietoturvasääntely.* Saatavissa: https://www.kyberturvallisuuskeskus.fi/fi/toimintamme/saantely-ja-valvonta/tietoturvasaantely. Viitattu 9.11.2024.

Kyberturvallisuuskeskus. 2024. *Pidempi parempi – Näin teet hyvän salasanan.* Saatavissa: https://www.kyberturvallisuuskeskus.fi/fi/ajankohtaista/ohjeet-ja-oppaat/pidempi-parempi-nain-teet-hyvan-salasanan. Viitattu 20.1.2025.

License. SQLite. Saatavissa: https://www.sqlite.org/copyright.html. Viitattu 19.1.2025.

Mask_attack [hashcat wiki]. Hashcat. Saatavissa: https://hashcat.net/wiki/doku.php?id=mask_attack. Viitattu 17.2.2025.

Nurmi, M. & Pyykönen, J. 2022. Viisauden hierarkia. *Helsingin yliopiston blogi.* Saatavissa: https://blogs.helsinki.fi/yhdenvertainen-liikunnallinen-lahio/2022/03/03/viisauden-hierarkia/. Viitattu 18.1.2025.

Password Storage Cheat Sheet. OWASP. Saatavissa: https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html. Viitattu 19.1.2025.

Pieprzyk, J., Hardjono, T. & Seberry, J. 2003. *Fundamentals of Computer Security.* Berliini: Springer.

Rankin, K. 2018. *Linux Hardening in Hostile Networks.* Boston: Addison-Wesley.

Schneier, B. 1996. *Applied Cryptograhpy: Protocols, Algorithms and Source Code in C.* 20th anniversary edition. Indianapolis: Wiley.

Turner, S. 2011. *RFC 6151 – Updated Security Considerations for the MD5 Message-Digest and the HMAC-MD5 Algorithms.* Saatavissa: https://datatracker.ietf.org/doc/rfc6151/. Viitattu 19.1.2025.

```
import hashlib
import secrets
import string
from argon2 import PasswordHasher


h = PasswordHasher()
chars = string.ascii_letters + string.digits
digits = string.digits
hashes_no_salt = []
hashes_with_salt = []
argon_hash = ""
for i in range(10):
    password = ''.join(secrets.choice(chars) for k in range(6))
    print(f"Salasana numero {i+1}: {password}")
    if i == 0:

        argon_hash = h.hash(password)
    hash_no_salt = hashlib.md5(password.encode()).hexdigest()

    hashes_no_salt.append(hash_no_salt)
    salt = ''.join(secrets.choice(digits) for l in range(2))

    hash_with_salt = hashlib.md5((password + salt).en-
code()).hexdigest()

    print(f"Suola salasanalle numero {i+1}: {salt}")

    hashes_with_salt.append(hash_with_salt)
```

```
    print()

print(f"Argon2id hash on: {argon_hash}")


with open("argon_hash.txt", "w") as txt0:
    txt0.write(argon_hash.strip())
with open("hashes_no_salt.txt", "w") as txt1:
    for hash in hashes_no_salt:
        txt1.write(hash.strip() + "\n")
with open("hashes_with_salt.txt", "w") as txt2:
    for hash in hashes_with_salt:
        txt2.write(hash.strip() + "\n")
```

```
from Crypto.Cipher import AES

from Crypto.Util.Padding import pad, unpad

from Crypto.Random import get_random_bytes

import time

import os

import sqlite3

import secrets

import string

from argon2 import PasswordHasher


h = PasswordHasher()

chars = string.ascii_letters + string.digits

digits = string.digits

connection = sqlite3.connect('database.db')

cursor = connection.cursor()

cursor.execute('''CREATE TABLE IF NOT EXISTS users ( id INTEGER PRI-
MARY KEY AUTOINCREMENT, username TEXT UNIQUE, password_hash TEXT
)''')


#Luodaan käyttäjät käyttäjätunnuksineen ja salasanoineen, ja lisä-
tään ne tietokantaan

for i in range(10):

    username = ''.join(secrets.choice(chars) for h in range(16))

    username = f"{username}{i+1}"

    print(f"Käyttäjätunnus numero {i+1}: {username}")

    password = ''.join(secrets.choice(chars) for k in range(6))

    print(f"Salasana numero {i+1}: {password}")
```

```
    argon_hash = h.hash(password)


    cursor.execute('''

    INSERT INTO users (username, password_hash)

    VALUES (?, ?)''', (username, argon_hash))

connection.commit()

connection.close()


#32 tavua = 256 bittiä

key = get_random_bytes(32)


db_data = ''

#Luetaan tietokannan tiedot muuttujaan

with open('database.db', 'rb') as db:

    db_data = db.read()


#Salataan lukemamme tiedot käyttämällä AES:ia ja 256-bittistä
avainta

cipher = AES.new(key, AES.MODE_CBC)

encrypted_data = cipher.encrypt(pad(db_data, AES.block_size))


#Luodaan tiedosto, joka sisältää salatekstin

with open('encrypted', 'wb') as encrypted:

    encrypted.write(cipher.iv)

    encrypted.write(encrypted_data)
```

```
#Luetaan salateksti muuttujaan

with open('encrypted', 'rb') as encrypted1:

    iv = encrypted1.read(16)

    data_to_be_decrypted = encrypted1.read()

#Otetaan aikaa ja katsotaan, kauanko sadan decrypt-operaation suo-
rittamisessa kestää

start = time.time()


for i in range(100):

    test_key = get_random_bytes(32)

    cipher = AES.new(test_key, AES.MODE_CBC, iv=iv)

    d = cipher.decrypt(data_to_be_decrypted)

    try:

        decrypted = unpad(d, AES.block_size)

    except ValueError:

        #Väärä avain aiheuttaa usein ValueErrorin.

        #Ei tulosteta kuitenkaan mitään virheilmoitusta tai muuta,

        #sillä se lisäisi kuluvaa aikaa.

        pass


end = time.time()

total_time = end - start

average_time = total_time / 100

print(f"100 decrypt-operaatiossa kesti {total_time:.8f} sekuntia")

print(f"Keskimäärin yhdessä decrypt-operaatiossa kesti {ave-
rage_time:.8f} sekuntia")
```

```
#Tässä testataan vielä decrypt-operaatiota oikealla avaimella

cipher = AES.new(key, AES.MODE_CBC, iv=iv)

decrypted = unpad(cipher.decrypt(data_to_be_decrypted),
AES.block_size)

with open('decrypted_db.db', 'wb') as decrypted_db:

    decrypted_db.write(decrypted)
```

```
import time

import secrets

import string

import base64

from argon2 import PasswordHasher


#Käytetään OWASPin suosittelemia parametreja. Käytettävä kirjasto käyttää oletuksena Argon2id:tä, jota haluamme käyttää, joten sitä ei tarvitse erikseen määrittää

h = PasswordHasher(

    time_cost=1, #Kierrosten määrä

    memory_cost=47104, #Käytettävän muistin määrä

    parallelism=1, #Säikeiden määrä

)


chars = string.ascii_letters + string.digits


#Luodaan 10 salasanaa, ja niille tiivisteet, joista jälkimmäiset hyökkääjä olisi saanut tietokannasta

password_hashes = []


known_password_indexes = []


for a in range(10):

    password = ''.join(secrets.choice(chars) for k in range(6))

    hash = h.hash(password)

    password_hashes.append(hash)
```

```
#Tässä simuloidaan sitä, kuinka hyökkääjä luo salasanaehdokkaan, luo siitä tiivisteen käyttäen kutakin kymmenen salasanan suolaa, ja vertaa tiivisteitä toisiinsa. Ajetaan testi 100 kertaa.

start = time.time()

for i in range(100):

    test_password = ''.join(secrets.choice(chars) for l in range(6))

    for k in range(10):

        if k not in known_password_indexes:

            hash = password_hashes[k]

            parts = hash.split('$')

            salt64 = parts[4]

            padding = "=" * (-len(salt64) % 4)


            salt_before = f"{salt64}{padding}"

            salt_final = base64.b64decode(salt_before)

            test_hash = h.hash(test_password, salt=salt_final)

            if hash == test_hash:

                known_password_indexes.append(k)

                print(f"Tiiviste {hash} vastaa salasanaa {test_password}")

                print()


end = time.time()

total_time = end - start

average_time = total_time / 100

print(f"Yhdessä for-loopin kierroksessa kului keskimäärin: {average_time:.8f} sekuntia")
```