HAMK
Hämeen ammatti-
korkeakoulu

C# pohjaisen tiedostosalausohjelman toteutus

AES: ja PBKDF2:n käyttö

Ammattikorkeakoulututkinnon opinnäytetyö
Sähkö- ja automaatiotekniikka, insinööri (AMK)
Kevät 2025
Jere Järvinen

HAMK
Hämeen ammatti-
korkeakoulu

Sähkö- ja automaatiotekniikka, insinööri (AMK) Tiivistelmä
Tekijä Jere Järvinen Vuosi 2025
Työn nimi C# pohjaisen tiedostosalausohjelman toteutus: AES:n ja PBKDF2:n käyttö
Ohjaaja Juhani Henttonen

Tämä opinnäytetyö käsittelee tiedostojen salausta ja purkua AES-salausalgoritmilla, hyödyntäen PBKDF2-avaimenjohdannaisfunktiota ja SHA-256 hajautusalgoritmia. Työn tavoitteena oli kehittää turvallinen ja käyttäjäystävällinen ohjelma tiedostojen suojaamiseen, erityisesti yksityishenkilöiden ja yritysten tarpeisiin. Ohjelma toteutettiin C#-ohjelmointikielellä ja .NET-kirjastoilla, käyttäen komentorivipohjaista käyttöliittymää.

Päätuloksena syntyi ohjelma, joka kykenee käsittelemään tiedostoja tehokkaasti ja luotettavasti, säilyttäen tiedostojen alkuperäisen tiedostomuodon salauksen ja salauksen purkamisen jälkeen. Työssä myös opittiin vahvojen salasanojen merkitys ja tärkeys, sillä ohjelma ei mahdollista tiedostojen purkua ilman oikeaa salasanaa.

Ohjelma täyttää modernit salausstandardit, mutta sen käytössä on huomioitava tietyt käyttörajoitukset, jotka tässä työssä, mutta myös ohjelman käyttöohjeissa läpi selkeästi. Työ tarjoaa hyvän ja yksinkertaisen pohjan jatkokehitykselle, esimerkiksi graafisen käyttöliittymän tekemiselle tai uudelle, monimutkaisemmalle ohjelmalle.

Avainsanat kryptografia, salaus, salauksen purku, tietoturva
Sivut 34 sivua

HAMK
Hämeen ammatti-
korkeakoulu

Electrical and Automation Engineering
Author Jere Järvinen
Subject Implementation of a C#-based File Encryption Program: Utilizing AES and PBKDF2
Supervisor Juhani Henttonen

Abstract
Year 2025

This thesis focuses on the file encryption and decryption using the AES encryption algorithm, utilizing the PBKDF2 key derivation function and the SHA-256 hashing algorithm. The objective was to develop a secure and user-friendly program for file protection, catering particularly to the needs of individuals and organizations. The program was implemented in C# programming language using .NET libraries and features a command-line interface.

The outcome of the project is a program that is capable of processing files efficiently and reliably while preserving the original file format after encryption and decryption. The program and thesis also emphasized the importance of strong passwords, as the program does not allow files to be decrypted without the correct password.

The program adheres to modern encryption standards but includes certain usage limitations, which are clearly outlined both in this thesis and in the program's user manual. The program and its code provide a solid and straightforward foundation for further development, such as creating a graphical user interface or building new, more complex application.

Keywords Cryptography, encryption, decryption, information security
Pages 34 pages

Sisällys

Kuvat, taulukot ja kaavat

1 Johdanto

Digitaalisen maailman laajeneminen on tehnyt tietoturvasta yhden aikamme keskeisimmistä haasteista. Yksityisyys ja arkaluontoisten tietojen suojaaminen on tärkeää niin yksityishenkilöille kuin yrityksille. Erityisesti tilanteissa, joissa käsitellään luottamuksellista tietoa, on olennaista varmistaa, että tallennettu, lähetetty tai vastaanotettu data on luotettavasti suojattu ja eheä. Tietojen suojaaminen asianmukaisilla kryptografisilla menetelmillä on keskeistä varmistettaessa, että arkaluontoiset tiedot pysyvät luottamuksellisina ja suojattuna luvattomalta käytöltä. Tietoturvan merkitys korostuu erityisesti ympäristössä ja tilanteissa, joissa yhä useammat toiminnot, kuten taloudelliset transaktiot, terveydenhuollon järjestelmät, viestintä sekä henkilökohtaisten tietojen tallennus tapahtuvat digitaalisessa muodossa. Yritysten liikesalaisuuksien sekä pankkitilitietojen ja yksityishenkilön henkilötunnusten suojaaminen on välttämätöntä, jotta voidaan estää vakavat haitat, kuten identiteettivarkaudet, tietomurrot ja taloudelliset menetykset. Tässä työssä käsitellään edellä mainittujen syiden takia erilaisten salausalgoritmien, funktioiden ja standardien käyttöä, kehitystä ajansaatossa ja niiden toimintaperiaatteita. Näiden menetelmien kehittäminen ja tutkiminen on koko ajan tärkeämpää maailman digitalisoitumisen myötä.

Advanced Encryption Standard (AES) ja Password-Based Key Derivation Function 2 (PBKDF2) ovat kaksi merkittävää kryptografista menetelmää suojata dataa ja ne tarjoavat tehokkaan keinon tietojen suojaamiseen. Toteutus vaiheessa tutkitaan C#-ohjelmointikielellä toteutetun tiedostosalausohjelman suunnittelua ja toteutusta hyödyntämällä sen sisäänrakennettuja .NET-kirjastoja. Näiden kirjastojen avulla voidaan käyttää AES:n ja PBKDF2:n lisäksi myös SHA-256 hajautusalgoritmia ja tästä syystä C# valikoitui ohjelmointikieleksi. AES:ää pidetään alan yhtenä parhaana algoritmina erityisesti sen eri avainpituuksien vuoksi, jotka mahdollistavat vahvan suojan erilaisia tietoturvatarpeita varten. PBKDF2 puolestaan on olennainen osa tiedostosalausjärjestelmän turvallisuutta, sillä se parantaa avainten suojaa lisäämällä salasanan pohjalta luotuun avaimeen iteraatioita ja satunnaisuutta suolan avulla. PBKDF2:n avulla avaimen murtaminen brute force-hyökkäyksillä on huomattavasti vaikeampaa, sillä jokainen murtoyritys vaatii huomattavasti enemmän laskentatehoa. SHA-256 käytetään työssä varmistamaan tiedoston eheys ja lisäämällä tiedostojen suojausta yhdistämällä sen tuottama hash-arvo tiedostosalausprosessiin. Tällä tavalla voimme pyrkiä luomaan luotettavan ratkaisun tiedostosalausohjelmaan, joka vastaa nykyaikaisia tietoturvan vaatimuksia.

2 Tietoturvan perusteet

Tietoturva on nykypäivänä olennainen osa digitaalista maailmaa, sillä lähes kaikki tiedot ja viestintä tapahtuvat sähköisesti. Tietoturvan tavoitteena on suojata tietoja luvattomalta pääsyltä tai tuhoutumiselta. Keskeisimpiä tietoturvan osa-alueita ovat tietojen luottamuksellisuus, eheys ja saatavuus, joita kutsutaan myös usein CIA-periaatteeksi. CIA tulee sanoista ”Confidentiality, Integrity and Availability”. Tämän periaatteen perustana on se, että tieto on vain oikeutettujen käyttäjien saatavilla, pysyy muuttumattomana ja tarkkana sekä on tarvittaessa käytettävissä. Tästä syystä tietoturvassa on tärkeää käyttää vahvoja salausalgoritmeja, jotka suojaavat dataa ja tuovat turvallisuutta. Tietoturvan ja suojauskeinojen kehityksen myötä myös hyökkäysmenetelmät ovat kehittyneet ja monimutkaistuneet vuosien saatossa. Ymmärtääkseen paremmin tässä työssä läpi käytävien algoritmien ja standardien tärkeyden avataan yleisiä hyökkäysmenetelmiä pintapuolisesti. (Safestate, n.d.)

Hyökkäysmenetelmiä on maailmassa lukuisia, mutta niistä tunnetuimmat ovat etenkin brute force ja rainbow table-hyökkäykset. Brute force-hyökkäyksissä hyökkääjä yrittää murtaa salasanan tai salauksen kokeilemalla järjestelmällisesti kaikkia mahdollisia avaimia tai salasanoja. Tämä hyökkäysmenetelmä on erityisen tehokas keino murtaa heikoilla tai lyhyillä salasanoilla tehdyt salaukset. Heikko salasana on sellainen, joka on helposti arvattavissa, kuten yleiset salasanat ja yksinkertaiset numeroyhdistelmät. Esimerkiksi ”qwerty” tai ”12345”. Lyhyt salasana on sellainen, joka on helppo löytää käymällä salasanoja tai numerosarjoja läpi brute force-hyökkäyksessä. Rainbow table, eli sateenkaaritaulukkohyökkäyksissä käyttää valmiita taulukoita, joissa on yhdistetty yleiset salasanat niiden hash-arvoihin. Näitä valmiita taulukoita voidaan käyttää tehokkaasti sellaisia salauksia vastaan, joissa salausta ei ole tehty satunnaisen datan avulla. Monien muiden hyökkäysmenetelmien lisäksi on myös olemassa hyvin yleinen manipulointihyökkäys nimeltä phishing. Näissä hyökkäyksissä hyökkääjä huijaa uhrin paljastamaan arkaluontoisia tietoja, kuten salasanoja. Nämä yleensä toteutetaan väärennettyjen verkkosivujen tai sähköpostien kautta esittäytymällä toiseksi henkilöksi. (Stallings, 2017, ss. 89, 626; Alexander, 2004, s. 27; F-Secure, 2022)

2.1 Salausalgoritmit

Yleisesti salausalgoritmeja voidaan jakaa kahteen päätyyppiin, symmetrisiin ja epäsymmetrisiin salausmenetelmiin. Symmetrisessä salauksessa samaa avainta käytetään sekä tiedon salaamiseen että purkamiseen. Tämän tyyppinen salaus on yleensä hyvin nopea ja sopii suurien tietomäärien suojaamiseen. Tunnettuja symmetrisiä salausstandardeja ovat AES ja DES. Näistä etenkin AES on yleisesti tunnettu, käytetty ja suositeltu salausalgoritmi, kun taas DES on puolestaan vanhempi, AES:n edeltäjä. AES kehitettiin korvaamaan DES, joka on osoittautunut huomattavasti epävarmemmaksi standardiksi. Tämä epävarmuus johtui DES:n liian lyhyestä, vain 56-bittisestä avaimesta, joka on liian lyhyt suojaamaan nykyaikaisia brute force-hyökkäyksiä vastaan. AES osoittautui loistavaksi korvaajaksi pidempien avaimien ansiosta. On tärkeää myös huomata, että AES:n ja DES:n algoritmit eivät liity toisiinsa tai ole versioita toisistaan. (Delfs & Knebl, 2007, ss. 11–25)

Olemassa on myös epäsymmetrisiä salausalgoritmeja. Näissä käytetään kahta erilaista avainta, missä tiedon salaukseen käytetään julkista avainta ja salauksen purkamiseen käytetään salaista avainta. Tämä menetelmä on turvallisempi avainten hallinnan kannalta, mutta laskennallisesti hitaampi. Tunnettuja epäsymmetrisiä algoritmeja on RSA, eli Rivest-Shamir-Adleman ja ECC, eli Elliptic Curve Cryptography. Vaikka epäsymmetriset algoritmit tarjoavat paremman suojan, eivät ne ole kuitenkaan yhtä suosittuja niiden hitauden ja monimutkaisuuden takia. AES tarjoaa riittävän suojan turvalliseen suojaamiseen yhdestä avaimesta huolimatta. Epäsymmetrisiä algoritmeja käytetään yleensä esimerkiksi digitaalisiin allekirjoituksiin, mutta ei itse tiedon salaamiseen ja tämän takia myös tässä työssä tutustutaan tarkemmin symmetrisiin salauksiin, etenkin AES:n käyttöön. (Delfs & Knebl, 2007, ss. 33–41)

2.1.1 AES-salausalgoritmin ja sen perusteet

AES tulee sanoista Advanced Encryption Standard ja se valittiin kansainväliseksi salausstandardiksi vuonna 2001 Yhdysvaltain kansallisen standardi- ja teknologiainstituutin (NIST) toimesta. Se perustuu belgialaisten kryptografien Joan Daemenin ja Vincent Rijmenin kehittämään Rijndael-algoritmiin. AES siis tunnettiin alun perin nimellä Rijndael, mutta standardisoinnin myötä nimeksi valikoitui AES. Se on laajalti käytössä erilaisissa turvallisuuskriittisissä sovelluksissa, kuten VPN-yhteyksissä, tiedostojen salauksessa ja turvallisessa verkkoliikenteessä. (Ferguson, Schneier, & Kohno, 2010, ss. 54-55)

AES perustuu tietojen salaukseen sekä purkuun annetun avaimen avulla. Se käyttää symmetristä 128-bittisiä, eli 16-tavuisia lohkoja ja tukee kolmea erilaista avain pituutta. Pidempi avain tarjoaa parempaa suojaa, mutta vaatii enemmän laskentatehoa. Avaimen pituus myös määrittää kierrosten ja kierrosavainten lukumäärän (taulukko 1). (Ferguson;Schneier;& Kohno, 2010, ss. 54–55)

Taulukko 1. AES:n avainpituudet, kierroksien ja kierrosavainten lukumäärä

Avaimen pituus	**Kierroksien lukumäärä**	**Kierrosavainten lukumäärä**
128-bittinen avain	10 kierrosta	11 kierrosavainta
192-bittinen avain	12 kierrosta	13 kierrosavainta
256-bittinen avain	14 kierrosta	15 kierrosavainta

AES toimintaperiaatteeltaan salaa 128-bittiä syötettä kerrallaan. Salattavan datan ollessa pidempi kuin 128-bittiä, se jaetaan tämän pituisiin lohkoihin ja jokainen lohko salataan erikseen. Lohkot muunnetaan saman pituisiksi salaviesteiksi avaimen avulla. Jos datan, eli syötteen pituus ei ole täsmälleen 128-bittiä, siihen lisätään täydennystä (padding), jotta viimeinen lohko saadaan tämän pituiseksi ennen salausta. On tärkeää huomata, että avaimen pituus ei vaikuta lohkojen kokoon, vaan turvallisuustasoon. 192-bittisen avaimen käyttö ei siis tarkoita, että AES toimisi 192-bittiä pitkillä lohkoilla. (Ferguson, Schneier;& Kohno, 2010, ss. 54–55)

2.1.2 Kierroksen rakenne

Kierroksen aluksi syöte XORartaan kierrosavaimella, joka johdetaan pääavaimesta käyttämällä vapaasti suomennettua ”avain ajoitusta” (key schedule). XOR tarkoittaa loogista porttia, jossa lähtö on tosi, jos täsmälleen yksi tulo on tosi. Tämä looginen portti on käytössä kryptografiassa, koska se mahdollistaa tietojen palautettavuuden. Kun suoritat XOR-operaation uudelleen saadun tuloksen ja avaimen avulla, palauttaa se alkuperäisen syötteen. Tämä ei olisi mahdollista muilla loogisilla porteilla. XOR-operaatiota suositaan myös sen turvallisuusominaisuuksien vuoksi. Toisin kuin AND- tai OR-operaatiot, XOR ei vuoda tietoja syötteestä tai avaimesta, mikä parantaa tietoturvaa ja lisäksi XOR-operaation tilastolliset ominaisuudet ovat kryptografian kannalta suotuisammat, sillä se tuottaa tasapainoisempia ja arvaamattomampia tuloksia.

Avain ajoitus laajentaa tässä yhteydessä alkuperäisen avaimen niin, että jokaisessa kierroksessa on käytettävissä oma uniikki avain. Laajennus siis johdattaa pääavaimesta, käyttäjän syöttämästä salasanasta useita eri kierrosavaimia. Koska AES käsittelee dataa 16-tavuisina lohkoina, jokainen lohko käsitellään 4x4 tavutaulukkona. Tässä tavutaulukossa jokainen tavu sijoitetaan taulukkoon vasemmasta yläkulmasta alkaen ylhäältä alas ja vasemmalta oikealle. Esimerkki tavutaulukosta kaavassa 1. (Stallings, 2017, ss. 174–177)

Kaava 1. AES 4x4 tavutaulukko

$$\begin{bmatrix} b_0 & b_4 & b_8 & b_{12} \\ b_1 & b_5 & b_9 & b_{13} \\ b_2 & b_6 & b_{10} & b_{14} \\ b_3 & b_7 & b_{11} & b_{15} \end{bmatrix}$$

Taulukolle suoritetaan SP-Network (substitution-permutation network), joka sisältää kaksi pääoperaatiota, substitution eli korvauksen ja permutation eli permutaation. Näitä operaatioita suoritetaan avainpituuden määrittelemän kierrosten lukumäärän verran. Nämä yhdessä tuottavat konfuusiota ja diffuusiota, missä konfuusiolla tarkoitetaan tavujen arvojen muuttamista epälineaarisiksi ja diffuusiolla tarkoitetaan sekoittamalla tavujen paikkoja. Korvaus tuottaa konfuusiota ja permutaatio tuottaa diffuusiota. Korvauksessa käytetään SubBytes-operaatiota, joka on AES spesifinen korvaus menetelmä. SubBytes-operaatiossa jokainen tavu korvataan toisella tavulla ennalta määritellyn korvaustaulukon, eli S-boxin avulla, mikä hajottaa alkuperäisen rakenteen ja tuottaa epälineaarisen lopputuloksen. S-boksi on yleensä kiinteä 256-alkiollinen taulukko, joka sisältää kaikki mahdolliset 8-bittiset arvot. Jokaiselle tavulle löytyy S-boksista vastaava korvausarvo. Korvauksen jälkeen suoritetaan permutaatio, joka sisältää kaksi vaihetta, ShiftRows ja MixColumns. Näissä vaiheessa tavut sekoitetaan uusille paikoilleen, mikä lisää diffuusiota. Tämä sekoitus tehdään siirtämällä rivejä vasemmalle eri määrillä. ShiftRows siirtymät taulukoitu taulukkoon 2 ja visuaalinen esimerkki tavujen siirrosta kaavassa 2. (Stallings, 2017, ss. 177–190)

Taulukko 2. ShiftRows siirtymät

Ensimmäinen rivi (0. rivi)	Ei siirry lainkaan
Toinen rivi (1. rivi)	Siirretään vasemmalle yhdellä tavulla
Kolmas rivi (2. rivi)	Siirretään vasemmalle kahdella tavulla
Neljäs rivi (3. rivi)	Siirretään vasemmalle kolmella tavulla

Kaava 2. SiftRows esimerkki

$$Syöte = \begin{bmatrix} b_0\ b_4\ b_8\ b_{12} \\ b_1\ b_5\ b_9\ b_{13} \\ b_2\ b_6\ b_{10}\ b_{14} \\ b_3\ b_7\ b_{11}\ b_{15} \end{bmatrix}, \qquad SiftRows\ suoritettu = \begin{bmatrix} b_0\ b_4\ b_8\ b_{12} \\ b_5\ b_9\ b_{13}\ b_1 \\ b_{10}\ b_{14}\ b_2\ b_6 \\ b_{15}\ b_3\ b_7\ b_{11} \end{bmatrix}$$

MixColumns on monimutkaisempi operaatio. Siinä jokainen sarake sekoitetaan keskenään ja käsitellään itsenäisenä. Jokaista saraketta sekoitetaan käsittelemällä jokainen sarake 8-bittisen Galois-kenttäaritmetiikan $GF(2^8)$ avulla. Sarakkeessa olevat tavut kerrotaan kiinteällä 4x4-matriisilla (kaava 3), mikä sekoittaa tavut keskenään ja luo monimutkaisen riippuvuuden alkuperäisten tavujen välille. Galois-kenttäaritmetiikka on matemaattinen rakenne, jossa laskutoimitukset suoritetaan modulo-operaatiolla, joka varmistaa, että tulokset pysyvät määritellyn bittikoon sisällä. Kaikki laskutoimitukset suoritetaan modulo $2^8 = 256$. MixColumns operaatio suoritetaan jokaisella kierroksella, paitsi viimeisellä. Tämä varmistaa, että salausprosessin lopussa tiedot eivät sekoitu enempää matemaattisesti. Tiedot ovat silti kryptattuja viimeisen kierrosavaimen avulla. (Stallings, 2017, ss. 177–190)

Kaava 3. MixColumns kiinteä matriisi ja esimerkkilasku

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix}$$

$$b_0' = (02 * b_0) \oplus (03 * b_5) \oplus (01 * b_{10}) \oplus (01 * b_{15})$$

$$b_5' = (01 * b_0) \oplus (02 * b_5) \oplus (03 * b_{10}) \oplus (01 * b_{15})$$

$$b_{10}' = (01 * b_0) \oplus (01 * b_5) \oplus (02 * b_{10}) \oplus (03 * b_{15})$$

$$b_{15}' = (03 * b_0) \oplus (01 * b_5) \oplus (01 * b_{10}) \oplus (02 * b_{15})$$

On huomioitavaa, että esimerkkilaskussa ei lasketa tavallisia kertolaskuja. $\oplus$, tarkoittaa XOR:a ja symboli * viittaa kertolaskuun aiemmin mainitussa Galois-kentässä $GF(2^8)$.

Kierroksen lopuksi lisätään uusi kierrosavain (AddRoundKey), joka XORataan nykyisen tilan kierrosavaimen kanssa. AddRoundKey suoritetaan myös ennen ensimmäistä pääkierrosta, koska se antaa alkuperäisen avaimen vaikutuksen jo ennen kuin salauskierrokset alkavat. Näillä menetelmillä kryptoanalyysi tehdään vaikeammaksi avalanche-efektin avulla, missä pienikin muutos syötteessä tai avaimessa vaikuttaa laajasti salattuun viestiin. Ja kuten edellä mainittiin, avaimen pituus määrittää sen, kuinka monta kierrosta suoritetaan. (Stallings, 2017, ss. 194–197)

AES kierrosrakenne lyhyesti:

- **Alustava kierros**: AddRoundKey suoritetaan ennen ensimmäistä pääkierrosta. Alkuperäinen avain XORataan heti syötteen kanssa, mikä muuttaa datan ennen varsinaisia kryptografisia operaatioita.

- **Pääkierrokset:** Jokaisessa pääkierroksessa suoritetaan neljä vaihetta – SubBytes, ShiftRows, MixColumns ja AddRoundKey.

- **Viimeinen kierros:** Suoritetaan päävaiheet ilman MixColumns-operaatiota.

2.1.3 AES moodit

AES:ää on mahdollista käyttää useilla eri moodeilla (block cipher mode of operation), kuten ECB, CBC, CTR jne. CBC (Cipher Block Chaining) on yleisesti suositeltu ja käytetty salausmoodi AES:n käytössä. Tässä moodissa jokainen lohko XORataan edellisen lohkon kanssa ennen salausta. Ensimmäinen lohko XORataan alustusvektorin (IV) kanssa, koska aiempaa lohkoa ei ole. Tämän moodin hyöty on se, että identtiset syöttölohkot eivät tuota identtisiä salauslohkoja, mikä parantaa tietoturvaa. Heikkouksia tässä moodissa on esimerkiksi se, että salauksen ja purkamisen täytyy tapahtua sarjassa, mikä voi hidastaa prosessointia sellaisiin moodeihin verrattuna, missä lohkoja voidaan käsitellä itsenäisesti. Tämä moodi myös vaatii alustusvektorin, joka täytyy olla satunnainen ja salainen. Sillä pyritään estämään hyökkääjän yritys ennakoida tietoja perustuen identtisiin salauslohkoihin tai tunnistaa samankaltaisuuksia. Tämä myös tarkoittaa sitä, että salattu tekstitiedosto ei paljasta toistuvia tai tunnistettavia kuvioita, vaikka siinä olisi toistuvia sanoja.

ECB moodiin verrattuna, missä jokainen lohko salataan erikseen samalla avaimella, CBC moodi on huomattavasti turvallisempi. ECB:n heikkous on se, että samat syötteet tuottavat tunnistettavia kuvioita. Yleisesti sanotaan, että ECB moodia ei kannata edes harkita käytettäväksi käytännön puolella, edes harjoituksissa, vaan sen olemassaolon merkitys piilee varoituksena huonosta suunnittelusta heikkouksiensa takia.

CTR (Counter Mode) puolestaan on lohkosalausmoodi, joka muuntaa lohkosalaimen virtasalaimeksi. Toisin kuin perinteinen lohkosalaus, CTR käyttää yksinkertaista menetelmää, jossa jokainen 128-bittinen lohko lasketaan käyttämällä ns. ”noncea”, eli satunnaisnumeroa tai tunnistetta. Salaus ja salauksen purku voidaan tässä moodissa tehdä rinnakkain, koska jokainen avainlohko voidaan laskea itsenäisesti. Tämä tekee CTR moodista huomattavasti tehokkaamman ja nopeamman tavan suorittaa salaus tai salauksen purku. Toisaalta tässäkin

moodissa on omat heikkoutensa. Esimerkiksi CBC moodin verrattuna, CTR moodi on tietovuoto alttiimpi, jos nonce toistuu liian usein. Tästä syystä usein sanotaan, että CBC moodin ollessa hitaampi, on se myös turvallisempi. Mutta CTR moodille on myös omat käyttötarkoituksensa, kuten sovelluksissa, joissa korkea suorituskyky on ensiarvoisen tärkeää, esimerkiksi suurten datamäärien salauksessa tai salauksen purkamisessa. (Stallings, 2017, ss. 213–224)

2.2 Avainten johtaminen

Avaintenjohdatus on keskeinen osa modernia kryptografiaa, erityisesti kun puhutaan turvallisesta ja tehokkaasta tiedon suojaamisesta. Kun dataa salataan, avaimen täytyy olla riittävän vahva, jotta hyökkääjä ei voi helposti päätellä sitä tai murtaa salausta, esimerkiksi brute force-hyökkäyksillä. Avaintenjohdatus tarkoittaa siis prosessia, jossa yksinkertaisesta ja mahdollisesti heikosta salasanasta tuotetaan kryptografinen avain, joka on vahvempi ja turvallisempi käytettäväksi salauksen yhteydessä, kuten AES:n kanssa. Tähän liittyvät avaintenjohdatusfunktiot ovat suunniteltu lisäämään laskennallista vaikeutta ja satunnaisuutta, mikä tekee avaimen murtamisesta merkittävästi haastavampaa. Tämä siksi, koska usein käyttäjä voi käyttää hyvinkin yksinkertaista salasanaa, joka voi olla altis monille erilaisille hyökkäyksille, erityisesti sanakirjahyökkäyksille tai sateenkaaritaulukkohyökkäyksille. Näiden hyökkäyksien tarkoitus on kokeilla yleisiä tai aiemmin kerättyjä ja löydettyjä salasanoja nopeasti ja tehokkaasti. Tämän estämiseksi salasanoja voidaan ”parantaa” käyttämällä avaintenjohdatusfunktioita.

On olemassa muutamia mainitsemisen arvoisia avaintenjohdatusfunktioita, mutta niistä ehdottomasti suosituin ja tunnetuin on PBKDF2, eli password-based key derivation function 2. Tämä johtuu siitä, että turvallisuuden lisäksi se on yksi harvoista avaintenjohdatusfunktioista, joka on standardisoitu. Tämä on antanut PBKDF2:lle huomattavasti enemmän näkyvyyttä ja tunnettavuutta, sillä monet yritykset ja muut tahot vaativat standardien noudattamista tietoturvakäytännöissään. Tämänkin standardoinnin on suorittanut NIST. Myös PBKDF2:n helppo saatavuus monissa kirjastoissa, kuten .NET, Python ja Java kirjastoissa on tehnyt siitä suositun funktion ja helposti integroitavan. Tästä syystä myös tässä työssä tutustutaan tarkemmin sen toimintaan. Kuitenkin muita mainitsemisen arvoisia funktioita ovat bcrypt ja scrypt. Näitä funktioita tässä työssä ei tarkemmin tutkita, mutta lyhyesti sanottuna näillä funktioilla on sama päätavoite ja käyttötarkoitus, kuin PBKDF2:lla, mutta algoritmien rakenne on hiukan erilainen. Esimerkiksi bcrypt funktio perustuu Blowfish-salausalgoritmiin, mikä käyttää automaattista suolaa

varmistaakseen erilaiset hash-arvot. Puolestaan scrypt on suunniteltu erityisesti brute force-hyökkäyksiä vastaan, sillä se käyttää paljon muistia, mikä tekee siitä todella kalliin ja hitaan murtaa. Näistä scrypt on selkeästi turvallisempi vaihtoehto ja onkin käytössä erittäin turvallisissa ympäristöissä. (Percival & Josefsson, 2006; Moriarty, Kaliski, & Rusch, 2017)

2.2.1 PBKDF2-funktion toimintaperiaatteet

PBKDF2, eli password-based key derivation function 2, on salasanapohjainen avaintenjohdantoalgoritmi, joka on osa PKCS #5 v2.0-standardia. Tämän standardin on julkaissut RSA Laboratories. PBKDF2 on osa laajempaa kryptografista standardisointiprosessia ja se on aiemmin mainitun standardi- ja teknologiainstituutin (NIST) suosittelema SP 800-132 julkaisussa, jossa käsitellään turvallisia avaintenjohdantomenetelmiä.

PBKDF2 käyttää hajautusfunktiota osana avaimen derivointiprosessia, esim. HMAC-SHA-256 ja sen päätarkoitus on tuottaa turvallinen kryptografinen avain heikommasta, käyttäjän antamasta salasanasta, mikä lisää satunnaisuutta. Tämä tapahtuu käyttäjän salasanan ja muiden parametrien perustella ja sitä voidaan esimerkiksi käyttää tuottamaan turvallinen ja salainen pääavain AES salausta varten.

PBKDF2:n tarkoitus tässä yhteydessä olisi siis luoda uusi avain alkuperäisestä avaimesta siten, että alkuperäisen avaimen selvittäminen brute force-hyökkäyksellä olisi vaikeampaa. Tämä siksi, että PBKDF2 lisää laskennallista kuormaa jokaisella iteraatio kerralla, mikä hidastaa hyökkäystä. (Moriarty;Kaliski;& Rusch, 2017; Turan, Barker, Burr; & Chen, 2010)

2.2.2 Salasana, suola ja iteraatiokerrat

PBKDF2 käyttää kolmea pääparametria, joiden perusteella se muodostaa uuden avaimen heikommasta avaimesta. Pääparametrit ovat siis salasana (password), suola (salt value) ja iteraatioiden määrä (iteration count). Siinä missä salasana on käyttäjän itse antama tieto, suolana toimii satunnainen arvo, joka yhdistetään salasanaan. Suolan tarkoitus on tehdä jokaisesta prosessista uniikki ja estää hyökkäyksiä, joissa käytetään valmiiksi laskettuja taulukoita, esim. sateenkaaritaulukkohyökkäyksissä. Tämä johtuu siitä, että satunnaisella datalla kahdesta identtisestä salasanasta saadaan muodostettua uniikit. Iteraatioiden määrä puolestaan lisää laskennallista kuormaa, joka tekee algoritmista hitaamman ja kalliimman murtaa, sillä hyökkääjän on toistettava sama prosessi monia kertoja. PBKDF2 käyttää hajautusfunktiota (yleensä HMAC) toistuvasti, yleensä useita tuhansia kertoja. Esimerkiksi

10 000 iteraationmäärällä PBKDF2 suorittaa HMAC-hajautuksen 10 000 kertaa, jossa jokaisella kerralla se yhdistelee saatua tulosta uudelleen salasanan ja suolan kanssa. Tämä hidastaa hyökkäysyrityksiä. PBKDF2 prosessissa voidaan myös määritellä haluttu avainpituus. Vaikka PBKDF2:n käyttämän hajautusfunktion tuottama avaimen koko on yleensä vakio, esim. SHA-1: 160 bittiä, SHA-256: 256 bittiä, jne., PBKDF2 kykenee tuottamaan minkä tahansa pituisen avaimen iteratiivisesti yhdistelemällä hajautusfunktion tuloksia saaden halutun pituuden, mikäli haluttu avainpituus ylittää hajautusfunktion tuottaman pituuden. Lopputuloksena PBKDF2 antaa salauksen vaatiman avaimen sopivalla pituudella, joka perustuu annettuun salasanaan, suolaan ja iteraatiomäärään. Tämä lopputulos on huomattavasti turvallisempi kuin yksittäinen hajautus salasanasta ennen AES:n käyttöä. (Moriarty, Kaliski, & Rusch, 2017; Turan, Barker, Burr, & Chen, 2010)

2.2.3 PBKDF2-algoritmin tehokkuus

PBKDF2 on suunniteltu vaikeuttamaan kryptografista analyysiä ”avalanche-efektin” avulla, kuten aiemmin todettiin myös AES:n yhteydessä. Tämä tarkoitti sitä, että pienikin muutos salasanassa tai suolassa, eli hyödynnettävässä datassa, muuttaa merkittävästi lopputulosta. Iteraatiot lisäävät myös hyökkäysten kustannuksia huomattavasti. Tämän vuoksi PBKDF2 on yleisesti käytetty algoritmi salasanapohjaisissa sovelluksissa. Näiden perusteella voidaan myös todeta, että mahdollisimman satunnaisella ja vahvalla salasanalla, onnistuneen hyökkäyksen toteuttaminen on todella vaikeaa ja kallista. (Upadhyay, Gaikwad, Zaman, & Sampalli, 2022)

2.3 Hajautusalgoritmit

Hajautusalgoritmit ovat matemaattisia funktioita, jotka ottavat syötteen ja palauttavat siitä kiinteän pituisen merkkijonon, jota kutsutaan hash-arvoksi. Syötteenä toimii esimerkiksi tiedoston data tai salasana. Hash-arvo toimii alkuperäisen datan sormenjälkenä ja tästä syystä jokaisen syötteen tulisi tuottaa oma ainutlaatuinen hash-arvo. Nykyään laadukkaita hajautusalgoritmeja ei ole kovin montaa tarjolla. Turvallisimpina pidetään SHA-perheen algoritmeja, kuten SHA-1, SHA-256 ja SHA-512. Näiden lisäksi on olemassa uudempia hajautusalgoritmeja, jotka perustuvat SHA-3 standardiin, mutta niiden turvallisuutta tutkitaan ja analysoidaan yhä aktiivisesti. (Ferguson, Schneier, & Kohno, 2010)

Vaikka SHA-perheen algoritmit ovat standardoituja ja laajasti käytössä, myös niiden turvallisuus vaatii edelleen jatkuvaa analysointia. Standardoinnin on tehnyt Yhdysvaltain

NIST, joka on liittovaltion virasto ja niiden kehitystyössä on ollut mukana NSA, joka on Yhdysvaltain tiedusteluvirasto. Kuten aiemmin mainittiin tutkiessamme AES:n toimintaa ja historiaa, NIST:n tehtävänä on kehittää teknisiä standardeja monilla aloilla. NSA puolestaan keskittyy kansalliseen tietoturvaan ja on osallistunut kryptografisten standardien kehittämiseen ja analysointiin suojatakseen maan tietoliikennettä. (Ferguson, Schneier, & Kohno, 2010)

SHA-2 on laajalti käytössä monissa turvallisuuskriittisissä sovelluksissa, kuten SSL/TLS-protokollissa, joka suojaa verkkoliikennettä internetissä. Myös kryptovaluutat hyödyntävät SHA-2 algoritmia ja erityisesti maailman tunnetuin ja arvostetuin kryptovaluutta Bitcoin käyttää SHA-256 hajautusta keskeisenä osana lohkoketjunsa turvallisuutta. SHA-2 algoritmeja käytetään myös digitaalisissa allekirjoituksissa, erilaisissa tietoturvaprotokollissa, kuten SSH (Secure Shell) ja S/MIME (Secure/Multipurpose Internet Mail Extensions), jotka suojaavat tiedonsiirtoa tietokoneiden välillä ja sähköpostiliikennettä. Myös yksi tärkeimmistä käyttökohteista on salasanojen hajautus palvelimilla. Sen sijaan, että palvelimelle tallennettaisiin salasanojen selkokielinen versio, tallennetaan niistä saadut hash-arvot. Tämä parantaa tietoturvaa, sillä mahdollisessa tietomurrossa hyökkääjä saa haltuunsa ainoastaan salasanojen hash-arvot. Näiden purkaminen alkuperäisiksi salasanoiksi vaatii lisätyötä, mikä vaikeuttaa salasanojen väärinkäyttöä. (Encryption Consulting, n.d.)

Lähes kaikki käytössä olevat hajautusfunktiot ovat iteratiivisia hajautusfunktioita. Iteratiivinen rakenne jakaa syötteen tietyn kokoisiin lohkoihin, esim. 512-bitin kokoisiin lohkoihin ja täydentää viimeisen lohkon tarvittaessa aiemmin läpikäydyllä täydennysmekanismilla, jota englanniksi kutsutaan nimellä ”padding”. Yleisesti suositelluin ja suosituin hajautusalgoritmi on SHA-perheeseen kuuluva SHA-256 algoritmi. (Ferguson, Schneier, & Kohno, 2010, ss. 77–80)

2.3.1 SHA-perheen historia

SHA tulee sanoista “Secure Hash Algorithm” ja sen ensimmäinen versio oli nimeltään SHA ilman numerointia tai muita lisäliitteitä. Nykyään kuitenkin ensimmäistä versioita kutsutaan yleisesti nimellä SHA-0. SHA-0 julkaistiin vuonna 1993, mutta vain kaksi vuotta myöhemmin vuonna 1995 NSA ilmoitti löytäneensä heikkouden algoritmista. Tämä johti siihen, että NSA päivitti algoritmin, jossa heikkous korjattiin ja päivitetty versio julkaistiin nimellä SHA-1. Aluksi heikkouden tai korjauksen yksityiskohtia ei paljastettu, mutta vuonna 1998 tutkijat Florent Chabaud ja Antoine Joux julkaisivat tiedon SHA-0:n heikkoudesta. Sen heikkous liittyi sen sisäiseen rakenteeseen ja haavoittuvaisuuteen törmäyshyökkäyksiä vastaan, minkä tutkijat

olivat onnistuneet toteuttamaan tehokkaasti. Törmäyshyökkäys tarkoittaa tilannetta, jossa kaksi eri syötettä tuottavat saman hash-arvon. Korjauksessa paranneltiin tapaa, miten bittejä sekoitetaan ja yhdistellään hajautusprosessin aikana. (Ferguson, Schneier, & Kohno, 2010, s. 82)

Vuonna 2001 NIST julkaisi kolme uutta hajautusfunktiota, jotka kuuluvat SHA-2 perheeseen ja vuonna 2004 perhe päivitettiin myös neljännellä hajautusfunktiolla. Näiden funktioiden hash-arvojen pituudet ovat 224, 256, 384 ja 512-bittiä. Rakenteeltaan SHA-2 muistuttaa edeltäjäänsä, SHA-1 algoritmia ja se on suunniteltu käytettäväksi salausavainten kanssa, joiden pituus on 128, 192 ja 256-bittiä, esim. AES:n kanssa. (Ferguson, Schneier, & Kohno, 2010, ss. 82–83)

2.3.2 SHA-perheen toimintaperiaatteet

Kuten aiemmin todettiin, SHA-0:sta löydettiin iso heikkous ja se päivitettiin SHA-1:een. Keskeinen muutos näiden välillä oli yhden bitin siirto jokaisen lohkon laskennassa. SHA-1 perustuu MD4-algoritmiin ja sen tuottama hash-arvo on suuruudeltaan 160-bittiä. SHA-1 koostuu useista vaiheista, joiden tarkoituksena on varmistaa, että alkuperäisen viestin jokainen bitti vaikuttaa lopulliseen hash-arvoon. Myös SHA-1 on rakenteeltaan iteratiivinen algoritmi, jossa alkuperäinen viesti jaetaan 512-bittisiin lohkoihin ja tarvittaessa täydennetään ”padding”-mekanismilla. Jokainen lohko koostuu 16:sta ”sanasta”. Sana tässä yhteydessä tarkoittaa yksikköä, joka tarkoittaa 32-bittiä. Kun data on jaettu lohkoihin, alkuperäiset 16 sanaa venytetään 80 sanaksi käyttämällä laskennallista prosessia, jossa uusia sanoja muodostetaan alkuperäisistä sanoista XOR-operaatioiden ja muiden menetelmien avulla. SHA-1 itsessään käyttää viittä 32-bittistä rekisteriä, joita kutsutaan A, B, C, D ja E rekistereiksi. Nämä rekisterit toimivat väliaikaisina tallennuspaikkoina algoritmin suorituksen aikana ja rekistereiden tiedot päivitetään jokaisen laskentakierroksen aikana. SHA-1 käy läpi 80 laskentakierrosta, jotka ovat jaettu neljään päävaiheeseen. Usein näitä päävaiheita kutsutaan laskentakierroksiksi, joka voi aiheuttaa aluksi sekaannusta. Jokaisessa neljässä päävaiheessa käytetään eri loogisia operaatioita, kuten XOR- ja AND-operaatioita. Myös vakiot ja bitinsiirrot muuttuvat päävaiheiden aikana. Kaikkien laskentakierrosten jälkeen, viisi 32-bittistä rekisteriä yhdistetään, jolloin tuloksena saadaan 160-bittinen hash-arvo alkuperäisen viestin datasta. (Stallings, 2017, ss. 355-356; Ferguson, Schneier, & Kohno, 2010, ss. 82–83)

Nykyään SHA-1 käyttöä ei enää suositella varsinkaan uusissa projekteissa, sillä se ei täytä nykyaikaisen kryptografian vaatimuksia. Tämä johtuu tehokkaampien

törmäyshyökkäysmetodien kehityksestä. Tästä syystä monet yritykset ovat siirtyneet uudempien vaihtoehtojen käyttöön. Vaikka SHA-2 muistuttaa rakenteeltaan edeltäjäänsä, eroaa se kuitenkin siinä, että SHA-2 käyttää pidempiä rekistereitä ja erilaisia laajennustekniikoita, mikä parantaa turvallisuutta. Esimerkiksi SHA-256 käyttää kahdeksaa 32-bittistä rekisteriä ja sisältää enemmän laskennallista monimutkaisuutta. SHA-2 käytöllä siis saavutetaan korkeampi turvallisuustaso suuremman hash-arvon ja monimutkaisuutensa ansiosta. Vaikka SHA-2 on laskentatehossa hitaampi kuin SHA-1, on se kuitenkin turvallisempi vaihtoehto näistä kahdesta. (Stallings, 2017, s. 356; Ferguson;Schneier;& Kohno, 2010, ss. 82–83)

SHA-3 julkistettiin vuonna 2015, mutta se ei ole läheskään yhtä laajasti käytössä ja tunnettu kuin SHA-2 perheen algoritmit. Se on rakenteeltaan hyvin erilainen edeltäjiinsä verrattuna. Se perustuu Keccak-algoritmiin ja sen luojia ovat Guido Berton, Joan Daemen, Michaël Peeters ja Gilles Van Assche. Edelleen melkein kymmenen vuotta SHA-3:n standardoinnin jälkeen, SHA-2 on yhä suositumpi ja yleisimmin käytetty standardimalli. Tämä johtuu useista eri syistä, mutta suurimpina syinä on, että SHA-2 on edelleen turvallinen käytettäväksi useimmissa sovelluksissa, se on huomattavasti nopeampi kuin SHA-3 ja se on integroitunut osaksi useita järjestelmiä ja protokollia. SHA-3 on kuitenkin hyväksytty standardi ja sitä käytetään joissakin uusissa kryptografisissa sovelluksissa, joissa halutaan ylimääräistä varmuutta erilaisten hyökkäysten varalta. Vaikka SHA-3 tarjoaa teoreettisesti paremman suoja, SHA-2 on edelleen käytännössä yhtä turvallinen, sillä siitä ei ole löydetty merkittäviä heikkouksia. Toistaiseksi SHA-2 on siis säilynyt ensisijaisena valintana useimmissa tietoturvasovelluksissa. (Ferguson, Schneier, & Kohno, 2010, ss. 79–80; Stallings, 2017, ss. 365–375)

2.3.3 SHA-256 rakenne

Perehdytään tarkemmin, miten SHA-256 toimii, sillä se on yksi suosituimmista ja käytetyimmistä hajautusalgoritmeista. Kuvataan yksi täysi kierros SHA-256 algoritmista, joka toistetaan hajautuksen aikana 64 kertaa. Huomioitavaa on, että SHA-256 jakaa samankaltaisuuksia SHA-512 kanssa, mutta käyttää 32-bittisiä sanoja, vähemmän kierroksia ja eri vakioita, kuin SHA-512. Kierros alkaa syötteen esikäsittelyllä, mikä tarkoittaa sitä, että syöte jaetaan 512-bittisiin lohkoihin ja tarvittaessa suoritetaan täydennys. Täydennys alkaa aina bitillä ”1”, jota seuraa riittävä määrä ”0”-bittejä, jotta kokonaispituudeksi saadaan 448. Viimeiset 64-bittiä 512:sta säästetään säilyttämään tieto alkuperäisen viestin pituudesta bitteinä ennen täydennystä. Tämän jälkeen algoritmi alustaa kahdeksan 32-bittistä rekisteriä vakioarvoilla, jotka on nimetty H1–H8. Rekisterit toimivat väliaikaisina tallennuspaikkoina

laskentavaiheiden aikana ja alustetaan vakiarvoilla, jotka on listattu taulukkoon 3. (Criptografia, n.d.; Stallings, 2017, ss. 356–364)

Taulukko 3. Vakioarvot SHA-256 rekisterien alustuksessa

H1 := 0x6a09e667
H2 := 0xbb67ae85
H3 := 0x3c6ef372
H4 := 0xa54ff53a
H5 := 0x510e527f
H6 := 0x9b05688c
H7 := 0x1f83d9ab
H8 := 0x5be0cd19

Lohkojen käsittely vaiheessa jokainen 512-bittinen lohko jaetaan kuuteentoista 32-bittiseen sanaan, jotka merkitään W1–W16. Sanat W17–W64 muodostetaan kaavalla, joka on esitetty kaavassa 4. Algoritmi käyttää näitä sanoja ja laajentaa ne 64 sanaan. Jokainen lohko prosessoidaan käyttämällä kompressiofunktiota. (Criptografia, n.d.; Stallings, 2017, ss. 356–364)

Kaava 4. Kaava W17-W64 sanojen muodostamiseen

$W_t = \sigma_1(W_{t-2}) + W_{t-7} + \sigma_0(W_{t-15}) + W_{t-16}$,

missä σ_0 ja σ_1 määritellään bitin kierto- ja siirto-operaatiolla.

Jokaiselle 512-bittiselle lohkolle algoritmi käy läpi 64 kierrosta, joissa jokainen kierros päivittää kahdeksan muuttujaa, a, b, c, d, e, f, g ja h. Kierroksen aikana suoritetaan laskut, jotka on esitetty kaavassa 5. Ch (Choose) funktio valitsee kahden arvon väliltä yhden arvon kolmannen arvon perusteella. Eli se valitsee f tai g arvon riippuen arvosta e. Se valitsee arvon f, jos e on 1 ja arvon g, jos e on 0. Maj (Majority) funktio puolestaan palauttaa bitin, joka esiintyy useimmin kolmen syötebitin a, b ja c joukossa. Sigma (∑) funktiot 0 ja 1 suorittavat bitinkiertoja ja siirto-operaatioita. (Criptografia, n.d.; Stallings, 2017, ss. 356–364)

Kaava 5. T1 ja T2 väliarvojen laskenta

$T_1 = h + \Sigma_1(e) + Ch(e, f, g) + K_t + W_t$

$T_2 = \Sigma_0(a) + Maj(a, b, c)$

Operaatioiden jälkeen muuttujat päivitetään taulukon 4 mukaisella tavalla. Muuttujien päivityksen perustana on, että väliaikaisten arvojen avulla lasketaan uusia arvoja jokaisella kierroksella. Tämä sekoittaa dataa monimutkaisesti, mikä tekee tuloksena saadusta hash-

arvosta turvallisen ja vaikeammin ennustettavan. (Criptografia, n.d.; Stallings, 2017, ss. 356–364)

Taulukko 4. SHA-256 rekisterien päivityskaava

h = g
g = f
f = e
e = d + T1
d = c
c = b
b = a
a = T1 + T2

Kun kaikki 64 kierrosta on suoritettu, päivitetään a-h muuttujien arvot yhdistämällä ne H1-H8 arvoihin. Lopullinen hash-arvo muodostetaan liittämällä muuttuja H1-H8 yhteen, mikä tuottaa 256-bittisen hajautuksen (kuvat 1 ja 2). (Criptografia, n.d.; Stallings, 2017, ss. 356–364)

Kuva 1. SHA-256 H1–H8 muuttujien päivitys

$$
\begin{aligned}
H_1^{(t)} &= H_1^{(t-1)} + a \\
H_2^{(t)} &= H_2^{(t-1)} + b \\
H_3^{(t)} &= H_3^{(t-1)} + c \\
H_4^{(t)} &= H_4^{(t-1)} + d \\
H_5^{(t)} &= H_5^{(t-1)} + e \\
H_6^{(t)} &= H_6^{(t-1)} + f \\
H_7^{(t)} &= H_7^{(t-1)} + g \\
H_8^{(t)} &= H_8^{(t-1)} + h
\end{aligned}
$$

Kuva 2. SHA-256 lopullinen hajautus

$$H = H_1^{(N)} \| H_2^{(N)} \| H_3^{(N)} \| H_4^{(N)} \| H_5^{(N)} \| H_6^{(N)} \| H_7^{(N)} \| H_8^{(N)}.$$

3 Ohjelman suunnittelu ja toteutus

Ohjelman rakennuksessa tullaan keskittymään rakenteen selkeyteen, turvallisuusnäkökulmien huomioimiseen ja ohjelman päätehtävä tulee olemaan tiedostojen salaaminen ja salauksen purkaminen turvallisesti käyttäen teoriaosuudessa mainittuja algoritmeja ja funktioita. Työ tullaan toteuttamaan C#-ohjelmointikielellä ja työssä tullaan hyödyntämään erilaisia .NET-kirjastoja. Ohjelman suunnittelussa ja toteutuksessa otetaan

myös huomioon käyttäjäystävällisyys, jotta siihen voisi tutustua kuka tahansa riippumatta käyttäjän teknillisestä osaamisesta.

Toteutus tullaan tekemään modulaarisella rakenteella, jossa päätoiminnallisuudet, kuten tiedoston käsittely, salaus ja käyttäjän syötteiden validointi jaetaan omiin selkeisiin metodeihin. Näitä metodeja kutsutaan päämetodissa silloin, kun niitä tarvitaan. Ohjelman suunnittelussa tullaan myös ottamaan huomioon erilaisia vaatimuksia ja käyttörajoituksia ja nämä asiat käydään myös sanallisesti läpi.

3.1 Ohjelman vaatimukset ja käyttörajoitukset

Ohjelman tavoitteena on tarjota käyttäjälle turvallinen ja tehokas tapa tiedostojen salaamiseen ja purkamiseen käyttäen AES-salausalgoritmia. Ohjelma suunnitellaan erityisesti tiedostojen suojaamiseen salasanalla ennen niiden tallentamista tai jakamista. Käyttökohteita ovat esimerkiksi henkilökohtaiset asiakirjat, kuten salasanoja sisältävät tekstitiedostot, kuvat tai muut arkaluontoiset tiedostot. Ohjelman tarkoituksena on palvella niin yksityishenkilöitä kuin yrityksiä ja samalla rohkaista käyttäjiä käyttämään vahvoja salasanoja. Turvallisuuden lisäämiseksi käytämme PBKDF2:sta ja SHA-256 hajautusta AES:n kanssa. Ohjelma on itsenäinen ja toimii yleisimmillä käyttöjärjestelmillä, kuten Windows, macOS ja Linux.

3.1.1 Toiminnalliset ja ei-toiminnalliset vaatimukset

Ohjelma toteutetaan C#-kielellä ja se toimii konsolipohjaisena sovelluksena. Salaus ja purku pyritään suorittamaan mahdollisimman tehokkaasti, kuitenkin täyttäen nykyaikaiset salausstandardit. Sisäisesti ohjelma toteutetaan käyttäen AES algoritmia ja 256-bittistä avainta. Salasanat johdetaan PBKFD2 algoritmilla käyttäen SHA-256 hajautusta, satunnaista suolaa ja 100 000 iteraatiota. Näillä menetelmillä varmistetaan tiedostojen turvallisuus sekä suojataan tiedostot mahdollisilta tietomurroilta.

Ohjelma soveltuu minkä tahansa tiedostotyypin salaamiseen ja purkamiseen koosta riippumatta. Lisäksi ohjelmassa on ominaisuus, joka antaa käyttäjälle tietoa salaus- tai purkuprosessin etenemisestä reaaliajassa. Käyttäjän antamat syötteet validoidaan ja ohjelma ilmoittaa esimerkiksi väärästä salasanasta tai virheellisestä tiedostopolusta. Salaus- ja purkuprosessit suunnitellaan siten, että tiedostojen alkuperäinen tiedostomuoto säilyy muuttumattomana ja tuotetut tiedostot tallennetaan automaattisesti alkuperäisen tiedoston

sijaintiin. Tiedostot nimetään salauksen tai purkamisen jälkeen joko ”encrypted” tai ”decrypted” etuliitteellä, millä pyritään helpottamaan salatun tai puretun tiedoston tunnistamista. Käyttäjä voi salauksen tai purkamisen jälkeen uudelleen nimetä tiedostot haluamallaan tavalla. Ohjelma myös käsittelee yleisimmät virhetilanteet, kuten väärät tiedostopolut tai salasanat turvallisesti ilman, että alkuperäiset tiedostot vaurioituvat. Sisäisesti otetaan myös huomioon tiedostojen eheys ja koodiversion varmistus, jolla voidaan varmistaa, että tiedostot eivät ole vioittuneet tai niitä ei ole manipuloitu.

Koska ohjelma on etenkin suunniteltu paikallisten tiedostojen salaus- ja purkuprosesseihin, on ohjelmassa myös käyttörajoituksia ja muita huomiotta jätettyjä asioita.

3.1.2 Käyttörajoitukset

Ohjelma sisältää joitakin käyttörajoituksia, jotka ovat jätetty huomiotta sovelluksen käyttötarkoituksellisista tai muista syistä, esim. virheen minimoimiseksi. Ohjelma ei mm. tarkista tiedoston sen hetkistä tilaa, kun se otetaan salattavaksi tai purettavaksi. Eli ohjelma ei varmista, että onko tiedosto toisen sovelluksen käytössä vai ei. Tästä syystä ohjelmaa ei suositella käytettäväksi jatkuvasti käytössä olevien, eli reaaliaikaisten tiedostojen salaukseen tai purkamiseen. Tämä siksi, että tiedostojen tiedot saattavat muuttua salaus- tai purkuprosessien aikana. Samasta syystä ohjelmaa ei myöskään suositella käytettäväksi suoraan pilvipalveluiden tai muiden synkronointiin perustuvien ohjelmien kanssa. Tiedostojen salaus ja purku suositellaan tekemään paikallisesti, jonka jälkeen tiedostoja voidaan tallentaa muihin palveluihin.

Ohjelmaa ei myöskään suositella käytettäväksi sellaisten tiedostojen salaukseen tai purkamiseen, jotka sisältävät upotettuja polkuja tai digitaalisia allekirjoituksia. Salaus- tai purkuprosessit voi rikkoa eheystarkistuksen, mikä voi johtaa tiedoston tietojen vioittumiseen siten, että ne eivät läpäise validointitestejä. Lisäksi ohjelmassa ei ole sisäänrakennettua virheenkorjausmekanismia, joka yrittäisi korjata vioittuneen tiedoston virheen sattuessa. Tästä syystä ohjelma myös suunnitellaan niin, että syötetiedostoja ei muuteta ja ohjelma osaa kuitenkin tunnistaa yleisimmät viat ja informoida käyttäjää niistä.

Ohjelma ei myöskään sisällä salasanojen tallennusta, mikä tarkoittaa, että käyttäjällä on täysi vastuu salasanojen turvallisesta tallentamisesta tai muistamisesta. Tämä rajoitus estää salasanan palauttamisen, mikä parantaa osittain ohjelman tietoturvaa, mutta tekee myös tiedostojen purkamisesta mahdotonta, jos salasana hukkuu.

Vaikka ohjelmassa on tiettyjä käyttörajoituksia, se soveltuu mainiosti paikallisten tiedostojen salaukseen. Kaikki käyttörajoitukset, ohjeet ja muut dokumentoinnit kirjoitetaan selkeästi käyttäjän luettavaksi.

3.2 Ohjelman arkkitehtuuri

Ohjelma on rakennettu Visual Studio 2022-kehitysympäristössä käyttäen .NET Framework -kirjastoja. Työssä on käytetty mm. ”System.Security.Cryptography”, ”System.IO” ja ”System.Text”-kirjastoja (kuva 3). Ohjelma on suunniteltu modulaariseksi siten, että jokainen toiminto on jaettu omiin erillisiin metodeihin. Tärkeimpiin komponentteihin kuuluvat päämetodi, kryptografiset metodit, tiedostokäsittely ja salasanan validointi. Päämetodi (main) ohjaa ohjelman kulkua ja kutsuu muita metodeja, kun niitä tarvitaan. Kryptografiset metodit (EncryptFile ja DecryptFile) vastaavat nimiensä mukaisesti tiedoston salauksesta ja purkamisesta. Näissä metodeissa käytetään kryptografisia algoritmeja ja funktioita tuottamaan turvallinen salaus ja salauksen purku. Purkamisvaiheessa myös varmistetaan yhteensopivuus versionumeron ja suolan avulla. Tiedostonkäsittelymetodeissa validoidaan tiedostopolku sopivaksi, jotta voidaan varmistaa ennen salausta tai purkua, että tiedosto löytyy ja on yhteensopiva. Mikäli tiedostoa ei löydy, käsitellään tämä virheenä ja käyttäjä voi yrittää uudelleen syöttää tiedostopolun. Salasanan validointimetodi varmistaa, että käyttäjän syöttämä salasana täyttää vaaditut vahvuuskriteerit salausvaiheessa. Tämän avulla pyritään rohkaisemaan käyttäjää käyttämään vahvoja salasanoja, jotta AES:n, PBKDF2:n ja SHA-256:n käytöstä saadaan paras mahdollinen hyöty. Salaus- ja purkuprosessit on pyritty tekemään virheenkestäviksi ja käyttäjä saa visuaalista palautetta salausprosessin etenemisestä tai yleisimmistä virheistä.

Kuva 3. Käytetyt .NET-kirjastot

```
using System;
using System.IO;     //for file input/output operations
using System.IO.Pipes;  //for file input/output operations
using System.Numerics; //provides numeric types
using System.Security.Cryptography; //cryptographic functions
using System.Text;  //encoding operations
using System.Linq;  //Includes methods like .Any()
```

3.2.1 Salausmetodi

”EncryptFile”-metodi toteuttaa tiedostojen salauksen hyödyntäen AES-algoritmia 256-bittisellä avaimella. Metodia tehdessä on pyritty hyödyntämään nykyaikaisten

salauskäytäntöjen menetelmää turvallisen salauksen luomiseksi luomalla satunnainen suola ja käyttämällä salasanapohjaista avainta. Metodi ottaa kolme parametriä, ”inputFile”, joka on polku salattavalle tiedostolle, ”outputFile”, joka on polku salatun tiedoston tallennukselle ja ”password”, joka on käyttäjän antama salasana (kuva 4).

Kuva 4. EncryptFile-metodin määrittely

```
1 reference | Jere Järvinen, 89 days ago | 1 author, 3 changes
public static void EncryptFile(string inputFile, string outputFile, string password)
{
```

Tarkastellaan tarkemmin metodin jokainen vaihe:

1. **Version hallinta:**
   Muuttuja ”revikka” alustetaan arvolla 1 (kuva 5). Tämä edustaa salaukseen käytetyn koodin versiota, joka tallennetaan ulostulotiedostoon yhteensopivuuden varmistamiseksi purkuvaiheessa. Tämä mahdollistaa tulevaisuudessa koodin päivittämisen ilman, että vanhojen tiedostojen purku estyy.

Kuva 5. Versionumeron alustus

```
int revikka = 1; //Code version
```

2. **Suolan luominen:**
   Metodissa luodaan satunnainen 16-tavua, eli 128-bittiä pitkä suola hyödyntämällä ”RandomNumberGenerator”-luokkaa (kuva 6). Suola on keskeinen komponentti salasanapohjaisessa avaimen johtamisessa, sillä se varmistaa, että sama salasana tuottaa erilaisen salatun datan eri tiedostoille. Suola tallennetaan ulostulotiedoston alkuun.

Kuva 6. Satunnaisen suolan luominen

```
//Define the salt (random data) value with random number generator
byte[] salt = new byte[16]; //Declares 16 byte (128 bits) array for the salt
using (var rng = RandomNumberGenerator.Create())
{
    rng.GetBytes(salt); //Fills the salt array with cryptographically secure random bytes
}
```

3. **Salasanapohjainen avaimen johdannaisfunktio:**
   Metodi käyttää PBKDF2 algoritmia ”Rfc2898DeriveBytes”-luokan avulla, jolla johdetaan 256-bittinen avain ja 128-bittinen alustusvektori AES:n käytettäväksi (kuva 7). ”Rfc2898DeriveBytes” käyttää SHA-256 hajautusta, satunnaista suolaa ja 100 000 iteraatiota, jotka on määritetty koodissa. Tämän avulla salaukseen saadaan lisättyä satunnaisuutta.

Kuva 7. Avaimen ja alustusvektorin johdatus

```
int iterations = 100000; //Number of iterations fo the key derivation function

//Perform AES encryption
using (Aes aes = Aes.Create())
{
    //Instantiates Rfc2898DeriveBytes with the password, salt, iteration count, and SHA-256 as the hash algorithm for deriving the encryption key and IV (initialization vector)
    //Rfc2898DeriveBytes class implements PBKDF2 (Password-Based Key Derivation Function 2), a method to derive keys from the password given a salt and a number of iterations
    using (var rfc2898DeriveBytes = new Rfc2898DeriveBytes(password, salt, iterations, HashAlgorithmName.SHA256))
    {
        aes.Key = rfc2898DeriveBytes.GetBytes(32);  //Sets the AES key (secret key) by deriving 32 bytes (256 bits) from rfc2898DeriveBytes
        aes.IV = rfc2898DeriveBytes.GetBytes(16);   //Sets the AES IV (block of bits to randomize the encryption) by deriving 16 bytes (128 bits)
```

4. **AES-salauksen asetukset:**
   Ennen varsinaista tiedostojen käsittelyä, luodaan metodissa salausobjekti ja määritellään se käyttämään PBKDF2 avulla johdettua avainta ja alustusvektoria. Tämän lisäksi määritellään kaksi muuttujaa, ”totalBytes” ja ”bytesProcessed”, joita käytetään tiedostokäsittelyn aikana päivittämään edistyspalkin tiedot (kuva 8).

Kuva 8. Salausobjektin ja asetuksien luominen

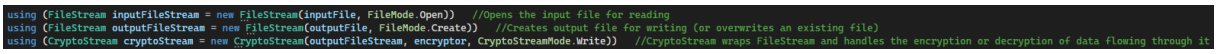
```
ICryptoTransform encryptor = aes.CreateEncryptor(aes.Key, aes.IV); //Creates encryptor object with derived key and IV

long totalBytes = new FileInfo(inputFile).Length;   //Calculates the total size of the input file in bytes
long bytesProcessed = 0;    //Counter to keep track of the number of bytes processed so far
```

5. **Tiedoston käsittely:**
   Tiedostot käsitellään osissa ja syötetiedosto luetaan ja kirjoitetaan ”FileStream-” ja ”CryptoStream”-luokkien avulla (kuva 9). ”inputFileStream” lukee syötetiedoston sisältöä ja ”outputFileStream” käytetään luomaan uusi tiedosto mihin salatut tiedot halutaan kirjoittaa, jonka jälkeen ”CryptoStream" toimii ns. ”kääreenä”, joka käyttää ”outputFileStream” apuna salatun tiedon kirjoituksessa.

   Aluksi uuteen tiedostoon tallennetaan salauksessa käytetty suola ja koodiversio, jotta purkuvaiheessa voidaan suorittaa salaus oikein ja varmistaa yhteensopivuus. Tämän jälkeen luodaan puskuri muuttuja, jonka avulla tiedosto käsitellään 4096-tavun paloissa. Tämä optimoi muistin käyttöä etenkin silloin, kun suuria tiedostoja käsitellään. Jokainen pala luetaan, salataan AES-salauksella ja salatut tiedot kirjoitetaan samaan ulostulotiedostoon, mihin tallennettiin myös suola ja koodin

versionumero. Samaan aikaan metodi päivittää tietoja salausprosessin etenemisestä ja tulostaa etenemisen reaaliajassa ”UpdateProgressBar”-toiminnon avulla (kuva 10).

Kuva 9. FileStream ja CryptoStream käyttö



Kuva 10. Salatun datan kirjoitus uuteen tiedostoon

```
outputFileStream.Write(salt, 0, salt.Length);   //Writes the salt to the start of the output file

outputFileStream.WriteByte((byte)revikka);  //Writes version number next

byte[] buffer = new byte[4096]; //Create a buffer array to gold data read from the file temporarily. Adjust buffer size as needed
int bytesRead;
//Continuously read from the input file until there are no more bytes to read
Console.WriteLine("\nEncrypting...");
while ((bytesRead = inputFileStream.Read(buffer, 0, buffer.Length)) > 0)
{
    cryptoStream.Write(buffer, 0, bytesRead);   //Write the bytes via cryptoStream automatically encrypting them
    bytesProcessed += bytesRead;    //Update the count of processes bytes
    UpdateProgressBar((double)bytesProcessed / totalBytes); //Update the progress bar display
}
```

3.2.2 Purkumetodi

”DecryptFile”-metodi toteuttaa salatun tiedoston purkamisen alkuperäiseen muotoonsa AES-salauksella. Se käyttää samaa avaintenjohdannaismenetelmää ja algoritmia kuin salausmetodi ”EncryptFile”. Näin varmistetaan yhteensopivuus ja tiedostojen eheys. Tarkastellaan tarkemmin salauksen purkamisen eri vaiheet metodissa:

1. **Alustus ja syötetiedoston avaaminen:**
   Aluksi määritellään suola 16-tavuiseksi, joka on salausvaiheessa määritelty vakiopituus. Tätä suolaa käytetään avaimen johtamisessa ja se on satunnainen. Suola luetaan tiedoston alusta avaimen johtamista varten. Tämän jälkeen tarkistetaan myös vielä versionumero, joka on tallennettu suolan perään. Jos versionumero ei vastaa odotettua arvoa, eli numeroa 1, ohjelma lopettaa purkamisen ja ilmoittaa käyttäjälle virheestä. Tässä kerrotaan, että syynä voi olla esim. korruptoitunut tai ei tuettu tiedosto tai versionumero. Myös iteraatioiden määrä määritellään ennen purkuoperaatioita (kuva 11). Tässä on tärkeää käyttää samaa määrää iteraatioita mitä salauksessa on käytetty, että purkaminen ei epäonnistu väärän avaimen tai alustusvektorin takia.

Kuva 11. Purkuoperaation alustus ja tiedoston avaus

```
byte[] salt = new byte[16]; //Declares 16 byte (128 bits) array for the salt

using (FileStream inputFileStream = new FileStream(inputFile, FileMode.Open))   //Opens the input file for reading
{
    inputFileStream.Read(salt, 0, salt.Length); //Reads the salt from the beginning of the file into the "salt" array

    int iterations = 100000; //Number of iterations used in the key derivation (matching the encrypting process)

    //Reads the version byte and then checks if the version number is what was expected
    int revikka = inputFileStream.ReadByte();
    if (revikka != 1)
    {
        Console.WriteLine("\nUnsupported file, version number or corrupted file.\nClosing app...");
        System.Threading.Thread.Sleep(8000);    //8 second delay
        System.Environment.Exit(1); //Exit with error
    }
```

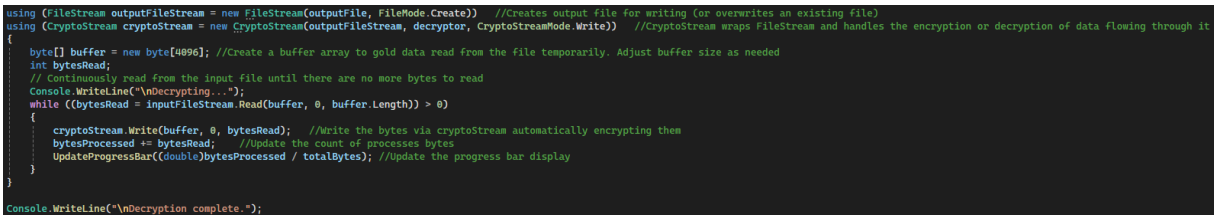
2. **Avaimen ja alustusvektorin generointi:**
   Salasana yhdistetään luetun suolan kanssa, jolla saadaan johdettua oikea 256-bitten avain ja 128-bittinen alustusvektori purkamista varten (kuva 12). Avaimen ja alustusvektorin johdatus toimii samalla periaatteella, kuin salausvaiheessa. Ennen purkuoperaatiota luodaan purkuobjekti ja määritellään se käyttämään tiedostosta luettuja ja johdettua avainta ja alustusvektoria. Tässä on hyvä muistaa XOR-operaation avulla tehdyn johtamisen perusperiaate. Kun samalla salasanalla ja suolalla käännetään dataa uudelleen, kääntyy salattu data alkuperäiseen muotoonsa. Ennen purkuoperaatioita myös määritellään taas ”totalBytes” ja ”bytesProcessed”, mutta ”totalBytes” kohdalla otetaan huomioon suolan ja versionumeron pituus. Nämä vähennetään ”totalBytes” arvosta, jotta saadaan purkamisen etenemisestä reaaliajassa pelkästään käännettävästä datasta.

Kuva 12. Avaimen ja alustusvektorin generointi

```
else
{
    //Creates AES instance for decryption
    using (Aes aes = Aes.Create())
    {
        //Instantiates Rfc2898DeriveBytes with the password, salt, iteration count, and SHA-256 as the hash algorithm for deriving the encryption key and IV (initialization vector)
        //Rfc2898DeriveBytes class implements PBKDF2 (Password-Based Key Derivation Function 2), a method to derive keys from the password given a salt and a number of iterations
        using (var rfc2898DeriveBytes = new Rfc2898DeriveBytes(password, salt, iterations, HashAlgorithmName.SHA256))
        {
            aes.Key = rfc2898DeriveBytes.GetBytes(32);  //Sets the AES key (secret key) by deriving 32 bytes (256 bits) from rfc2898DeriveBytes
            aes.IV = rfc2898DeriveBytes.GetBytes(16);   //Sets the AES IV (block of bits to randomize the encryption) by deriving 16 bytes (128 bits)

            ICryptoTransform decryptor = aes.CreateDecryptor(aes.Key, aes.IV);  //Creates decryptor object with derived key and IV

            long totalBytes = new FileInfo(inputFile).Length;   //Calculates the total size of the input file in bytes
            totalBytes -= (salt.Length + sizeof(byte));   //Adjust totalBytes to exclude the bytes used for salt and version number
            long bytesProcessed = 0;    //Counter to keep track of the number of bytes processed so far
```

3. **Purkuprosessi:**
   Myös purkuprosessissa määritellään 4096-tavun puskurimuuttuja, joka auttaa prosessin optimointia luku ja purkuvaiheessa. Purkuprosessissa käytetään salausprosessin tavoin ”FileStream”- ja ”CryptoStream”-luokkia tiedon lukemiseen syötetiedostosta ja puretun datan kirjoitukseen ulostulotiedostoon. Samalla ”bytesProcessed” ja ”totalBytes” vastaavat visuaalisen edistymispalkin päivityksestä käyttäjälle (kuva 13).

Kuva 13. Purkuprosessi



3.2.3 Virheenkäsittely

Ohjelma käsittelee yleisimmät virhetilanteet turvallisesti käyttäen ”try/catch”-lohkoja, validointimenetelmiä ja ohjaavia konsoliviestejä. Tutkitaan virheenkäsittelyä käymällä läpi tärkeimpiä virheenkäsittelykohtia.

1. **Syötetiedoston virheet:**
   Jos käyttäjän antamaa tiedostopolkua ei löydy, ohjelma ilmoittaa asiasta ja pyytää käyttäjää syöttämään tiedostopolun uudelleen. Tätä toistetaan niin pitkään, että tiedostopolku sopivalle tiedostolle löytyy. Tämä käsittely tapahtuu omassa ”GetValidFilePath()”-metodissa (kuva 14).

Kuva 14. Tiedostopolun virhe

```
//For getting valid file path from the user
2 references | 0 changes | 0 authors, 0 changes
private static string GetValidFilePath()
{
    //Promts the user to paste the file path for input file
    Console.WriteLine("\nDrag and drop file here or enter the file path manually:");
    do
    {
        filePath = Console.ReadLine().Trim('"');

        //Checks if the file at the provided path exists
        if (!File.Exists(filePath))
        {
            Console.WriteLine($"\nFile not found: {filePath}. Try again.\n");
            System.Threading.Thread.Sleep(1000);    //1 second delay before the next attempt can be made
        }
        else
        {
            break;
        }
    } while (true);
    return filePath;
}
```

2. **Virheellinen salasana:**

   Jos käyttäjä antaa väärän salasanan purkuprosessissa, ohjelma tunnistaa tämän, kun AES-purku epäonnistuu. Virheestä ilmoitetaan käyttäjälle, ja osittain tai viallisesti purettu tiedosto poistetaan automaattisesti. Tämä käsittely tapahtuu ”main()”-metodissa ”try/catch”-lohkolla (kuva 15).

Kuva 15. Virheellisen salasanan tunnistaminen

```
//Perform decryption with try function for catching possible decryption errors.
try
{
    DecryptFile(filePath, outputFilePath, password); //Perform decryption
}
catch (CryptographicException) //Handle possible decryption errors
{
    Console.WriteLine("\nWrong password or file corrupted.\nClosing app...");
    File.Delete(outputFilePath); //Delete partially decrypted file that contains error
    System.Threading.Thread.Sleep(8000); //8 second delay
    System.Environment.Exit(1); //Exit with error
}
```

3. **Tiedostojen korruptio:**

   Jos tiedosto on vioittunut, ohjelma ilmoittaa, että tiedosto ei ole tuettu tai se on vioittunut. Tämä tapahtuu tarkastamalla metatiedot, esimerkiksi suola tai versionumero tiedoston alusta ja käyttäjää informoidaan mahdollisesti korruptoituneesta tiedostosta, mikäli purku epäonnistuu.

4. **Käyttäjän kirjoitusvirheet:**
   Käyttäjä voi syöttää virheellisiä tietoja, kuten kirjoittaa väärin toiminnot ”encrypt” tai ”decrypt”, antamalla tyhjiä valintoja tai antamalla epäsopivan salasanan salausvaiheessa. Näitä virheitä käsitellään ”main()”-metodissa ”switch”-funktiolla ja omissa ”GetPassword()”- ja ”IsPasswordStrong()”-metodeissa. Virheen sattuessa, kerrotaan käyttäjälle mikä meni vikaan. Kerrotaan, että täytyy kirjoittamalla valita ”encrypt” tai ”decrypt” ja salasanan puutteet listataan, eli käyttäjää ohjataan koko prosessin läpi mahdollisimman selkeästi.

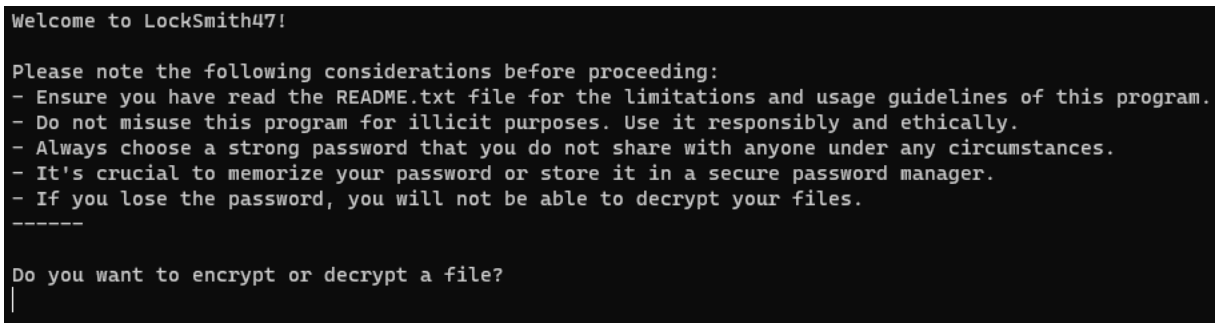
5. **Ohjelman luvattoman käytön esto:**
   Ohjelma on suunniteltu toimimaan turvallisesti ja tämä voidaan suorittaa valvomalla sen eheyttä ja estää luvaton käyttö. Tämä on toteutettu sisäänrakennetulla mekanismilla, joka tarkistaa, että ohjelman mukana toimitetut keskeiset tiedostot, kuten lisenssiehdot, ovat paikallaan ja sisältävät alkuperäisen sisällön ”ValidateCopyright()”-metodissa. Mikäli ohjelman tiedostojen eheys on vaarantunut tai niitä on muokattu, ohjelma tulkitsee tilanteen tekijänoikeusrikkomukseksi ja sulkeutuu automaattisesti.

3.2.4 Käyttöliittymä

Ohjelman käyttöliittymä on suunniteltu yksinkertaiseksi ja suoraviivaiseksi, jotta se palvelee niin teknisiä kuin vähemmän teknisiä käyttäjiä tehokkaasti. Käyttöliittymä on toteutettu komentorivipohjaisena CMD-ikkunassa, mikä mahdollistaa ohjelman keveyden ja yhteensopivuuden useiden eri käyttöjärjestelmien kanssa, kuten Windowsin, macOS:n ja Linuxin. Käyttöliittymässä käyttäjä kommunikoi ohjelman kanssa tekstipohjaisten komentojen ja valintojen avulla. Ohjelma esittää selkeitä kysymyksiä ja antaa käyttäjälle ohjeita vaiheittain, jotka ohjaavat tiedoston salauksen ja purkamisen. Etenkin käyttöliittymää on yritetty ohjelmaa tehdessä kehittää, jotta ohjelma olisi mahdollisimman käyttäjäystävällinen. Vuorovaikutus käyttäjän ja ohjelman välillä tapahtuu seuraavasti.

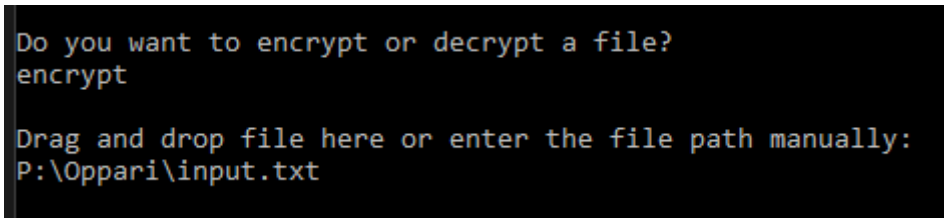
1. **Toiminnon valinta:**
   Käyttäjää pyydetään valitsemaan, haluavatko he salata tai purkaa tiedoston (kuva 16). Vastaus kirjoitetaan näppäimistöllä ja ohjelma validoi syötteen.

Kuva 16. Toiminnon valinta



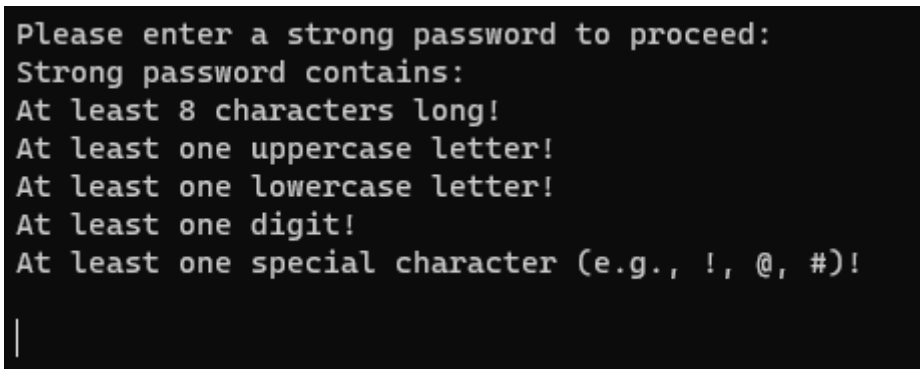
2. **Tiedoston valinta:**
   Käyttäjä antaa salattavan tai purettavan tiedoston polun ja ohjelma tarkistaa tiedostonolemassaolon ennen jatkamista (kuva 17).

Kuva 17. Tiedostopolun syöttäminen



3. **Salasanan syöttö:**
   Käyttäjä syöttää salasanan joko tiedoston salaamiseen tai purkamiseen (kuva 18). Ohjelma opastaa käyttäjää käyttämään vahvoja salasanoja ja validoi salasanan salausvaiheessa. Opastusta ja kriteerien kertomista ei tapahdu purkuvaiheessa.

Kuva 18. Salasanan syöttäminen



Nykyisessä toteutuksessa käyttöliittymä on täysin komentorivipohjainen, mikä saattaa rajoittaa ohjelman houkuttelevuutta visuallista käyttöliittymää suosiville käyttäjille. Tarkoituksena onkin tulevaisuudessa päivittää koodi ja ohjelma käyttäen graafista

käyttöliittymää, esim. Formsin avulla. Tämä mahdollistaa lisää ominaisuuksia ja houkuttelevan ulkonäön. Toistaiseksi kuitenkin ohjelman ensimmäinen versio haluttiin suorittaa komentorivipohjaisena sen keveyden ja helppouden takia.

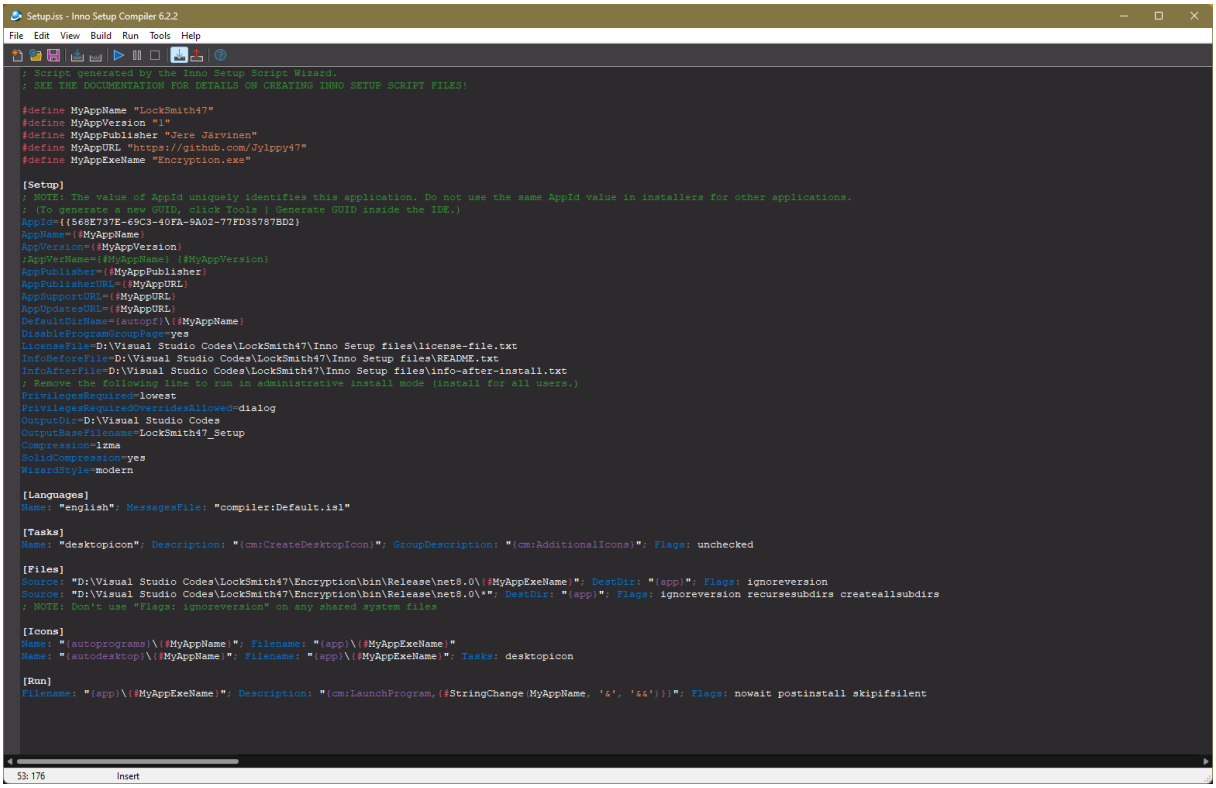
3.3 Sovelluksen ja asennustiedoston luominen

Projektin kääntäminen sovellukseksi on olennainen osa ohjelmistokehitysprosessia, jossa lähdekoodi paketoidaan asennettavaksi ja toimivaksi kokonaisuudeksi. Tämä mahdollistaa sovelluksen käytön käyttäjän laitteella ilman Visual Studio 2022 sovellusta. Tässä työssä asennustiedoston luomiseen hyödynnettiin Inno Setup-ohjelmaa.

Inno Setup on avoimen lähdekoodin asennusohjelmien luontityökalu, jonka avulla voidaan luoda mukautettavia asennustiedostoja. Työkalu käyttää ohjelmointikielenään Pascal Script-kieltä, joka pohjautuu Pascal-kieleen ja tarjoaa monipuoliset mahdollisuudet räätälöitävien asennuspakettien luomiseen. Työkalun avulla projektin tiedostot voidaan paketoida yhteen .exe-asennustiedostoon ja mahdollistaa asennusikkunan muokkaamisen. Tämä tarkoittaa sitä, että asennustiedoston muotoa ja kontekstia voi muokata haluamallaan tavalla, kuitenkin skriptin mahdollisuuksien mukaan.

Tässä projektissa useimmat Inno Setupin perustoiminnot jätettiin oletusasetuksille. Skriptissä kuitenkin personoitiin sovelluksen nimi, julkaisija ja muut tarvittavat tiedot sovellukselle sopiviksi. Skriptissä myös määriteltiin, että mitkä tiedostot sisällytetään asennuspakettiin, sekä useita erilaisia tekstipaketteja kuten lisenssitiedot, käyttörajoitukset ja muuta infoa, jota käyttäjä voi lukea asennuksen aikana. Asennuspakettiin lisättiin myös mahdollisuus luoda työpöytäkuvake sovellukselle sekä suorittaa ohjelma automaattisesti heti asennuksen jälkeen. Inno Setup-skriptin sisältöä voi tarkastella kuvasta 19.

Kuva 19. Inno Setupin sisältö



3.4 Dokumentointi

Ohjelmiston dokumentointi on tärkeä osa projektia, sillä se varmistaa ohjelman ymmärrettävyyden niin koodin kirjoittamisen kuin sovelluksen käytön aikana. Tässä työssä dokumentointia on tehty useassa eri vaiheessa ja niihin sisältyvät koodin kommentointi, sovelluksen käyttöohjeet ja käyttörajoitukset sekä tietosuojakäytäntö.

Koodin kommentointi on kirjoitettu niin, että koodin ymmärtäminen olisi helppoa myös pidemmän ajan kuluttua. Tämä parantaa koodin ylläpidettävyyttä ja varmistaa, että jatkokehitys olisi helpompaa. Kommentoinnissa on keskitytty erityisesti ohjelman tärkeimpiin toiminnallisuuksiin ja loogisiin osiin, jotta ohjelman toiminta olisi mahdollisimman helppo ymmärtää myös sellaisten ihmisten toimesta, jotka eivät ole perehtyneet kryptografisten algoritmien teoriaan.

Käyttöohjeet ja käyttörajoitukset ovat käyttäjän luettavissa sovelluksen asennuksen aikana. Nämä myös lisätään asennuskansioon tekstitiedostona asennuksen yhteydessä, jotta niitä voi tarkastella myöhemmin asennuksen jälkeen. Käyttöohjeet tarjoavat lyhyesti ja yksinkertaisesti selitetyt ohjeet ohjelman käytöstä ja rohkaisevat lukemaan käyttöliittymää

tarkasti, sillä käyttöliittymä osaa ohjata käyttäjää salauksen ja purkamisen aikana. Samaan tiedostoon on myös kirjoitettu lyhyesti ohjelman käyttörajoituksista.

Tietosuojakäytäntö esitetään käyttäjälle asennuksen yhteydessä, mikä käyttäjän täytyy hyväksyä. Käyttäjälle kerrotaan kuka sovelluksen omistaa ja että omistaja ei vastaa mahdollisista sovelluksen aiheuttamista ongelmista sen aikana tai jälkeen, koska sovellus on tehty opinnäytetyönä. Tietosuojakäytännössä mainitaan erityisesti, että sovelluksen omistaja ei ole vastuussa, mikäli esimerkiksi käyttäjän salasana hukkuu tai jos sovellusta väärinkäytetään. Väärinkäytöllä tarkoitetaan esimerkiksi tilannetta, jossa sovellusta on käytetty laittomiin toimiin missään muodossa.

4 Testaus ja tulokset

Myös testaus on tärkeä osa ohjelmistokehitystä, sillä sen avulla voidaan varmistaa sovelluksen suorituskyky, turvallisuus ja käyttäjäkokemus. Tässä työssä ohjelman testaus pyrittiin suorittamaan mahdollisimman kattavasti osaamisen ja mahdollisuuksien mukaan. Eri testausvaiheissa keskityttiin erityisesti käyttäjätestaukseen, suorituskyvyn mittaamiseen ja turvallisuuden arviointiin. Tuloksia analysoitiin vahvuuksien ja heikkouksien tunnistamiseksi ja niiden pohjalta laadittiin parannusehdotuksia, miten ohjelmaa voisi jatkokehittää.

4.1 Testausmenetelmät ja tulokset

Ohjelman testauksessa käytettiin seuraavia menetelmiä:

1. **Käyttäjätestaus**
   Käyttäjätestaus suoritettiin kahden henkilön toimesta, sellaisen, joka osaa käyttää sulavasti tietokoneita ja Windows ympäristöä sekä sellaisen toimesta, joka ei ole yhtä teknillinen osaaja. Testauksissa huomattiin selkeästi, että vaikka komentorivi-ikkuna on loistava käyttöliittymä keveyden ja helppolukuisuuden ansiosta sellaiselle, joka osaa Windows-ympäristöä käyttää, on vähemmän teknillisellä osaajalla selkeitä vaikeuksia seurata komentorivipohjaista käyttöliittymää. Etenkin käyttäjän oma huolimattomuus korostui ohjeista huolimatta. Käyttöliittymään kirjoitettuja asioita ei luettu kunnolla loppuun asti, esimerkiksi kun ohjelma salauksen tai purkamisen jälkeen kertoo tarkasti mihin kansioon uusi tiedosto tallennettiin ja millä nimellä. Vaikka tämä on käyttäjän omaa huolimattomuutta, johtuu tämä myös komentorivipohjaisen käyttöliittymän monotonisuudesta. Tärkeät tekstit tai asiat eivät

korostu tarpeeksi mustaharmaasta käyttöliittymästä ja käyttäjä ei välttämättä tiedä, mikä tiedostopolku edes on. Teknillisellä osaajalla ei ollut suurempia vaikeuksia ohjelman käytön aikana.

2. **Suorituskyky**

   Ohjelman suorituskykyä testattiin kolmella erikokoisella ja tyyppisellä tiedostolla. Näitä tiedostoja salattiin ja purettiin sovelluksen avulla ja tavoitteena oli analysoida salaus- ja purkuprosessin nopeutta, sekä vaikutusta järjestelmän resursseihin. Ajastuksessa esiintyi pientä vaihtelua operaation mukaan. Salauksen purkaminen osoittautui nopeammaksi prosessiksi kuin salaus, todennäköisesti satunnaisen suolan ja muun datan luomisen takia salaus vaiheessa. Sovellus suoriutui tiedostojen salauksesta ja salauksen purkamisesta tehokkaasti. Pienten, n. 10 MB olevien tiedostojen käsittelyaika oli lähes huomaamaton, n. yhden sekunnin luokkaa. Mitä pienempää tiedostoa salattiin tai purettiin, sitä huomaamattomampi käsittelyaika oli. Tiedostokoon kasvaessa myös käsittelyaika kasvoi. 152 MB tiedoston käsittelyaika oli n. 11 sekunnin luokkaa, joka on vielä hyväksyttävällä tasolla verrattuna salattavan tiedoston kokoon. Kuitenkin suurempien tiedostojen salauksessa prosessin käsittelyaika kasvoi huomattavasti. 667 MB kokoisen videotiedoston salaukseen kului n. 54 sekuntia ja salauksen purkamisessa kului n. 47 sekuntia. Tämä on jo huomattavasti pidempi aika odottaa salauksen tai purkamisen suorittamista ja vaatimuksista riippuen voi olla liian pitkä käsittelyaika. Kuitenkin tässä työssä ja käyttötarkoituksessa n. minuutin pituinen käsittelyaika on vielä hyväksyttävällä tasolla hyvin suurissa tiedostoissa, ottaen huomioon, että sovellus on lähtökohtaisesti tehty pienten tiedostojen salaukseen.

3. **Turvallisuus**

   Sovelluksen turvallisuutta arvioitiin analysoimalla käytettyjä salausmenetelmiä ja suolan käyttöä. Tarkastelussa pyrittiin varmistamaan, että salattu data ei ole avattavissa ilman oikeaa salasanaa. Lisäksi tarkasteltiin, miten sovellus käsittelee virhetilanteita kuten väärää salasanaa tai vioittuneita tiedostoja. Käytetyt salausalgoritmit ja suolan generointi todettiin turvallisiksi ja salausmenetelmät noudattavat alan parhaita käytäntöjä. Virhetilanteissa sovellus palautti selkeitä virheilmoituksia, esimerkiksi käyttäjälle ilmoitetaan väärästä salasanasta ilman, että se vaarantaa tietoturvaa.

4.2 Tulosten analysointi

Ohjelma sisältää selkeitä vahvuusalueita tämänhetkisessä versiossa, mutta ohjelmasta löytyy myös heikkouksia, mitkä eivät ole välttämättä heti huomattavissa. Ohjelmaa on testattu perinpohjaisesti kaikilla mahdollisilla tavoilla ja jokaisen toiminnon toimivuus on testattu kattavasti. Ohjelman selkeiksi vahvuuksiksi on osoittautunut sen turvallinen tiedostojen käsittelytapa ja käyttäjäystävällisyys. Vaikka käyttöliittymä on monotoninen ja vähemmän teknisellä käyttäjällä voi aluksi olla haasteita seurata käyttöliittymän ohjeistusta, on kuitenkin ohjeet, yleiset virhetilanteet ja muut tärkeät tekstit kirjoitettu selkeästi. Ohjelma osaa ohjata sen käyttöä melkein tilanteesta riippumatta ja testausten aikana ei saatu aiheutettua virhetilannetta, joka olisi sekoittanut ohjelman.

Sovelluksen vahvuuksiin voi myös lukea sen suorituskyvyn. Vaikka ison tiedoston salaus ja salauksen purkaminen voi kestää pienen hetken, tekee se prosessit kuitenkin tehokkaasti käyttäen sopivasti tietokoneen resursseja. Testausten aikana ei huomattu, että sovellus olisi käyttänyt liikaa tietokoneen resursseja, mikä tekee siitä käytännöllisen päivittäisessä käytössä. Kuitenkin heikommissa tietokoneissa, kuten vanhoissa läppäreissä suurten tiedostojen käsittely voi olla todella hidasta, mikä voi osoittautua heikkoudeksi. Tästä syystä ohjelmaa onkin suositeltu yksinkertaisten ja pienten tiedostojen salaukseen, kuten dokumenttien tai salasanatiedostojen salaukseen, eikä esim. ison kuvatarkkuuden omaavien elokuvien salaukseen.

Testausten aikana huomattiin myös yhden ominaisuuden puute, joka vaikuttaa suoraan käyttäjäystävällisyyteen ja suorituskykyyn. Ohjelma ei kykene ilmoittamaan virheellisestä salasanasta salauksen purkamisen aikana, vaan ilmoittaa virheestä vasta, kun koko tiedosto on yritetty purkaa. Tämä tarkoittaa sitä, että jos salauksen purkaminen kestää kauan, käyttäjä joutuu odottamaan koko tämän ajan ennen kuin hän saa ilmoituksen väärästä salasanasta. Tämä vaikeuttaa huomattavasti salasanan ”testaamista” tilanteessa, missä käyttäjä ei täydellä varmuudella muista salaukseen käyttämäänsä salasanaa. Tämä puute yritettiin ohjelman kirjoituksen aikana korjata kirjoittamalla testi tyyppinen salauksen purku, jossa pieni pala tiedoston alusta yritetään purkaa ja tätä kautta löytää kryptografinen poikkeama datassa. Tätä ominaisuutta ei kuitenkaan saatu lukuisten yritysten jälkeen toimimaan oikein ja ominaisuudesta luovuttiin toistaiseksi. Yritysten aikana ohjelma ei joko osannut tunnistaa väärän salasanan aiheuttamaa kryptografista poikkeamaa, tai jos osasi, niin se ei osannut palauttaa oikealla salasanalla tiedostoa alkuperäiseen muotoonsa, vaan data vioittui. Tämä ominaisuus jätettiin tietoisesti parannusehdotukseksi tulevaisuudessa, kun ohjelmaa päivitetään uudenlaiseksi.

Muihin parannusehdotuksiin lukeutui myös käyttöliittymän päivittäminen uuteen, jossa komentorivi-ikkunasta luovutaan. Tämä mahdollistaa painikkeiden ja edistyneempien ominaisuuksien luomisen käyttöliittymään, joka tekee ohjelmasta helppokäyttöisemmän ja selkeämmän. Samalla voisi lisätä myös ominaisuuden, jolla tiedosto voitaisiin avata ”Open With” toiminnolla suoraan kansiosta ilman, että sovellusta itsessään tarvitsee erikseen avata. Tällöin ohjelma saa tiedostopolun automaattisesti ja käyttäjän tarvitsee syöttää vain salasana. Myös suorituskykyä voitaisiin optimoida käyttämällä rinnakkaislaskentaa suurten tiedostojen käsittelyyn. Kuitenkin nämä ovat toistaiseksi vain tämänhetkisiä ideoita ja ohjelmaa päivitettäessä parannusehdotusten tärkeys punnitaan ja päivitetään ohjelmaa yksi ehdotus kerrallaan tärkeysjärjestyksessä.

5 Pohdinta ja johtopäätökset

Tässä osiossa arvioidaan työn onnistumista, omaa teoreettisen ymmärryksen kehittymistä sekä esitetään jatkokehitysmahdollisuuksia. Lisäksi tarkastellaan työn merkitystä ja sen käytännöllisyyttä arkisessa käytössä.

Työn tavoitteena oli kehittää tiedostojen salaus- ja purkusovellus, joka on turvallinen käyttää ja käyttäjäystävällinen. Projekti onnistui hyvin, sillä lopputuloksena syntyi toimiva tietokonesovellus, joka saatiin toteuttamaan salaus turvallisesti käyttäen AES-salausta ja PBKDF2-avainjohdannaisfunktiota SHA-256 hajautuksen avulla. Ohjelman käyttöliittymä ja asennusprosessi suunniteltiin selkeiksi ja testauksen perusteella sovellus vastaa sille annettua vaatimuksia. Toisin sanoen siis turvallisuuteen ja käyttäjäystävällisyyteen liittyvissä osa-alueissa pääosin onnistuttiin.

Lyhyesti onnistumisista voidaan kertoa salausmekanismin noudattaminen alan standardeja ja virhetilanteiden turvallinen ja huolellinen hallinta. Sovelluksen käytettävyys ja dokumentaatio on toteutettu selkeästi, mikä helpottaa käyttäjää sovelluksen käytössä. Asennusprosessi saatiin toteutettua myös helpoksi Inno Setupin avulla. Selkeys ja yksinkertaisuus olivat tärkeässä roolissa asennuspakettia tehdessä. Vaikka työssä saavutettiin suurilta osin asetetut tavoitteet, jäi muutama asia jatkokehitykseen tulevaisuudessa, joita käytiin läpi tulosten analysointi osiossa.

5.1 Teoreettisen ymmärryksen kehittyminen

Projektin aikana syvensin ymmärrystäni useista ohjelmistokehitykseen ja tietoturvaan liittyvistä aiheista ja niiden tärkeydestä nykyaikaisessa maailmassa. Työn aikana pääsin tutustumaan eri kryptografisiin algoritmeihin ja standardeihin, sekä myös yleisesti tietoturvaan. Etenkin työn aikana päästiin tutustumaan AES-salauksen ja PBKDF2-avainjohdannan toimintaperiaatteisiin sekä niiden toteutukseen C#-kielellä. Opin työn aikana lukuisia asioita, mutta etenkin esim. kuinka suola ja iteraatiot parantavat salauksen turvallisuutta, miksi pitempi salasana on parempi kuin lyhyt ja miksi tietyt algoritmit ja funktiot ovat suositumpia kuin toiset. Sovellusta tehdessä opin kuinka tiedostoja voidaan lukea, kirjoittaa ja käsitellä tehokkaasti käyttäen .NET-kirjastoja. Myös käyttöohjeiden ja muiden dokumenttien tekeminen kehitti viestinnällisiä taitoja ohjelmistokehityksen aikana ja lukuisten yritysten ja palautteiden avulla sain laadittua vaatimukset täyttävät dokumentaation ohjelmalle sen toiminnasta ja käytöstä.

5.2 Opinnäytetyön merkitys ja hyödyntäminen

Tämän opinnäytetyön myötä syntyi konkreettinen sovellus, jolla tiedostojen salaus ja salauksen purkaminen voidaan suorittaa turvallisesti. Sovellusta voidaan hyödyntää henkilökohtaisessa elämässä tai organisaatiotasolla tietoturvan parantamiseksi ja ainakin itse tulen käyttämään sovellusta salasanatiedostojen turvalliseen varmuuskopiointiin pilvipalvelulle, mikäli toinen salasanojen tallennukseen tarkoitettu kaupallinen ohjelma joskus lakkaa toimimasta. Työssä saavutettu ratkaisu havainnollistaa turvallisten kryptografisten menetelmien toteutusta, mikä voi tarjota esimerkin aloittelevalle ohjelmistokehittäjälle. Myös itse opin hyvän ohjelmistokehityksen periaatteita ja menetelmiä.

Siinä missä sovellus tarjoaa myös aiemmin todetusti luotettavan ratkaisun arkaluontoisten tiedostojen salaamiseen, toimii työ myös yksinkertaisena oppaana kryptografisten menetelmien käytöstä, ohjelmistokehityksen työvaiheista ja asennustiedoston luomisesta. Työtä voidaan myös hyödyntää monipuolisemmille ja edistyneemmille ohjelmille pohjana, kuten pilvipohjaisille salausratkaisuille tai käyttöliittymällisille sovelluksille.

Lähdeluettelo

Alexander, S. (2004). *password protection for modern operating systems.* https://www.usenix.org/system/files/login/articles/1103-alexander.pdf

Criptografia. (n.d.). *The cryptographic hash function SHA-256*. https://helix.stormhub.org/papers/SHA-256.pdf

Delfs, H. & Knebl, H. (2007). *Introduction to Cryptography.* Springer. https://books.google.fi/books?id=Nnvhz_VqAS4C&pg=PA11&redir_esc=y#v=onepage&q&f=false

Encryption Consulting. (n.d.). *What is SHA? What is SHA used for?* https://www.encryptionconsulting.com/education-center/what-is-sha/

Ferguson, N. Schneier, B. & Kohno, T. (2010). *Cryptography engineering.* Wiley Publishing.

F-Secure. (28. 10 2022). *What is phishing?* https://www.f-secure.com/en/articles/what-is-phishing

Moriarty, K. Kaliski, B. & Rusch, A. (2017). *RFC 8018 - PKCS #5: Password-Based Cryptography Specification Version 2.1.* IETF. https://datatracker.ietf.org/doc/rfc8018/

Percival, C. & Josefsson, S. (2006). *RFC 7914 - The scrypt Password-Based Key Derivation Function.* IETF. https://datatracker.ietf.org/doc/html/rfc7914

Safestate. (n.d.). *Mitä tietoturvalla tarkoitetaan?* https://www.safestate.com/fi/artikkelit/mita-tietoturvalla-tarkoitetaan/

Stallings, W. (2017). *Cryptography and network security - principles and practice.* Pearson.

Turan, M. S. Barker, E. Burr, W. & Chen, L. (2010). *NIST Special Publication 800-132: Recommendation for Password-Based Key.* NIST. https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-132.pdf

Upadhyay, D. Gaikwad, N. Zaman, M. & Sampalli, S. (2022). *Investigating the Avalanche Effect of Various Cryptographically Secure Hash Functions and Hash-Based Applications.* IEEE Access. https://ieeexplore.ieee.org/document/9923931