Szakdolgozat

Miskolci Egyetem

A személyes információszervezés biztonságtechnikai aspektusai

Készítette:

Nemcsik Dániel

Programtervező informatikus

Témavezető:

Piller Imre

Miskolc, 2022

Miskolci Egyetem
Gépészmérnöki és Informatikai Kar
Alkalmazott Matematikai Intézeti Tanszék **Szám:**

Szakdolgozat Feladat

Nemcsik Dániel (I3I4BP) programtervező informatikus jelölt részére.

A szakdolgozat tárgyköre: kulcsszavak, hasonlók

A szakdolgozat címe: A személyes információszervezés biztonságtechnikai aspektusai

A feladat részletezése:

A személyes információszervezés (Personal Information Management) az egy személyhez köthető információk (például naptárbejegyzések, kontakt listák, jelszavak) kezelésével foglalkozik.
A dolgozat ennek biztonságtechnikai vonatkozásával foglalkozik. Megvizsgálja, hogy az aktuális, személyenként is több gépes környezetben milyen lehetőségek vannak az adatok szinkronizálására és védelmére.
Bemutatja egy Java asztali alkalmazás elkészítését, amely segíti a felhasználót, hogy az adataihoz az általa használt eszközökön biztonságosan hozzá tudjon férni.

Témavezető: Piller Imre (Tanársegéd)

A feladat kiadásának ideje:

. . . . . .
szakfelelős

Eredetiségi Nyilatkozat

Alulírott **Nemcsik Dániel**; Neptun-kód: `I3I4BP` a Miskolci Egyetem Gépészmérnöki és Informatikai Karának végzős Programtervező informatikus szakos hallgatója ezennel büntetőjogi és fegyelmi felelősségem tudatában nyilatkozom és aláírásommal igazolom, hogy *A személyes információszervezés biztonságtechnikai aspektusai* című szakdolgozatom saját, önálló munkám; az abban hivatkozott szakirodalom felhasználása a forráskezelés szabályai szerint történt.

Tudomásul veszem, hogy szakdolgozat esetén plágiumnak számít:

- szószerinti idézet közlése idézőjel és hivatkozás megjelölése nélkül;
- tartalmi idézet hivatkozás megjelölése nélkül;
- más publikált gondolatainak saját gondolatként való feltüntetése.

Alulírott kijelentem, hogy a plágium fogalmát megismertem, és tudomásul veszem, hogy plágium esetén szakdolgozatom visszautasításra kerül.

Miskolc, ...év ..... .hó ..... .nap

.....

Hallgató

1.

szükséges (módosítás külön lapon)

A szakdolgozat feladat módosítása

nem szükséges

....

dátum témavezető(k)

2. A feladat kidolgozását ellenőriztem:

témavezető (dátum, aláírás): konzulens (dátum, aláírás):

....

....

....

3. A szakdolgozat beadható:

....

dátum témavezető(k)

4. A szakdolgozat ..... szövegoldalt

..... program protokollt (listát, felhasználói leírást)

..... elektronikus adathordozót (részletezve)

.....

..... egyéb mellékletet (részletezve)

.....

tartalmaz.

....

dátum témavezető(k)

5.

bocsátható

A szakdolgozat bírálatra

nem bocsátható

A bíráló neve: .....

....

dátum szakfelelős

6. A szakdolgozat osztályzata

a témavezető javaslata: .....

a bíráló javaslata: .....

a szakdolgozat végleges eredménye: .....

Miskolc, ...

a Záróvizsga Bizottság Elnöke

Tartalomjegyzék

1. fejezet

Bevezetés

A dolgozat fő célja egy Java asztali alkalmazás elkészítése volt, ami képes többgépes környezetben is biztonságosan eltárolni a felhasználó adatait és visszaadni azokat szinkronizált módon az eszközök között.

A program készítése során különböző problémákat kellett megoldanom, mint például a megfelelő titkosítási algoritmus kiválasztása, adatszinkronizáció elérése különböző eszközök között, esztétikus Java grafikus felhasználói felület létrehozása, stb..

A dokumentum a felmerülő problémák potenciális megoldási lehetőségeit és elméleti hátterét mutatja be.
Megpróbálja részletesen ismertetni adattitkosítás során előfordulható problémákat.
Jellemzi, összeveti ezen problémák megoldási lehetőségeit. Ezek közé értendő az adatszerkezetek és adattárolási módok, adatbázis típusok, adatbázis titkosítási módszerek, titkosítási algoritmusok jellemzése.

2. fejezet

Koncepció

2.1. Személyes Információszervezés

A személyes információszervezés (PIM - Personal Information Management) olyan folyamat ami alatt egy ember megpróbálja az általa hasznosnak vélt információkat összegyűjteni, rendszerezni, tárolni [1]. A folyamat lehet fizikai vagy digitális.

Fontos, személyes információ lehet például:

- Személy hivatalos adatai, okmányai
- Telefonszámok, e-mail címek, elérési adatok
- Képek, videók, TV adások
- Időpontok, tapasztalatok, emlékek
- Költségek, kiadások
- Zenék
- Érdeklődési körökkel, hobbikkal kapcsolatos információk

Ez a lista személyről személyre változik, minden ember különbözik, más információkat talál fontosnak.

Nagyon előnyös, ha valamilyen formában megvalósítjuk a személyes információszervezést az életünkben.
Gondoljunk bele, hányszor volt már olyan, hogy egy időpontot elfelejtettünk, nem találtunk meg egy weboldalt, nem emlékeztünk egy születésnapra vagy egy leadási határidőre. Egy PIM rendszer lényege ilyen információknak a struktúrált, összeszedett tárolása, hogy megkönnyítse az adott személy mindennapjait, hiszen ha egy helyen tárolunk minden fontos információt, akkor a jövőben tudni fogjuk, hogy hol keressük őket.

2.2. Adattárolás

Az adatok tárolása megoldható adatbázisok segítségével és fájlokba.

2.2.1. XML és JSON

Adatátviteli formátumok közül a leggyakrabban használtak közé tartozik az XML és a JSON [2]. Céljuk, hogy platform-független, programozási nyelv független, alkalmazás független adatleíró formátumot adjanak, ami képes adatok nagy mértékű átvitelére és feldolgozására. A két formátum részletes összehasonlítását tartalmazza a 2.1-es táblázat.

XML – eXtensible Markup Language

Rugalmas, könnyen értelmezhető emberek és számítógép számára is.

HTML-hez hasonló szintaxisú nyelv. Felépítése tag-ekből áll. Nem előre definiáltak a tag-ek, ez azt jelenti, hogy mi hozzuk létre őket. Egy példa tag felépítése: <autó>....</autó>.

Első a nyitó tag, a második, ferde vonallal jelölt tag a záró tag.
Tag-ek között megadhatjuk a tartalmukat (pl. autó márka), vagy további tag-ekkel, amiket gyerekelemek (pl rendszám, szín) nevezünk, tovább konkretizálhatjuk a tartalmat.
Üres tag megadására is lehetséges, aminek nincs tartalma (pl.:<autó/>).

Egy XML dokumentum robosztus méreteket is elérhet a tag-ekből álló felépítése miatt.

JSON – JavaScript Object Notation

Rugalmas, könnyen értelmezhető emberek és számítógép számára is

Adatok név-érték párokban szerepelnek (pl. ”keresztnév” : ”Dániel”).
Több egymást követő adat vesszővel van elválasztva egymástól (pl.: ”márka” : ”Mercedes”, ”szín” : ”piros”).
{ } – objektumot tartalmaz (pl. {”keresztnév” : ”Dániel”, ”vezetéknév” : ”Nemcsik”})
[ ] –tömböt tartalmaz (pl. { ”diákok”:[
{”keresztnév” : ”Dániel”, ”vezetéknév” : ”Nemcsik”},
{”keresztnév” : ”Tibor”, ”vezetéknév” : ”Nagy”}]}

2.1. táblázat. JSON és XML összehasonlító táblázat

JSON	**XML**
JavaScript nyelven alapszik	SGML szabványból származtatott
Tábla(kulcs-érték) felépítés	Fa felépítés
Könnyedén értelmezhető	Tag felépítés miatt, aki nem tudja hogy kell értelmezni, annak nehezebb dolga lesz mint egy JSON file-lal lenne
Nem támogat kommenteket	Támogat kommenteket
Adatok szerver és böngésző oldal közötti hordozására preferált	Szerver oldali információ tárolásra preferált
Kevésbé terjedelmes és gyorsabb	Több szó szerepel benne, így nagyobb terjedelmű. Lassabb.
Kisebb file méret	Nagyobb file méret
String, számok, tömbök, objektumok és boolean értékeket támogat	Komplexebb adattípusokat is támogat például képek, nem-primitív adattípusok, stb..
Minden böngésző támogatja	Legtöbb böngésző támogatja
Adatcsere formátumtípusú	Jelölőnyelv formátumtípusú
Adat megjelenítésére nem alkalmas	Mivel jelölőnyelv (markup language), ezért adat megjelenítésére is alkalmas
Egy szabványos javascript függvénnyel használat előtt elemezni/felbontani/szeparálni (parse) kell a file-t.	Adott programozási nyelvnek megfelelően kell elemezni/felbontani/szeparálni (parse) a file-t.
Könnyedén elemezhető, kevés kód szükséges hozzá	Nehezebben elemezhető a tartalma
Adat-orientáltnak nevezik	Dokumentum-orientáltnak nevezik

Tegyük fel, hogy egy alkalmazás adatai vagy JSON vagy XML típusú fájlként, lokálisan tároljuk a felhasználó eszközén. **Lokális fájl** alapú tárolást az alábbi tulajdonságokkal jellemezhetjük:

- Gyors és egyszerű hozzáférés az adatokhoz.
- Könnyedén implementálható megoldás, évtizedek óta alkalmazott módszer az adatok lokális fájlként való tárolása.
- Könnyedén átmásolható fizikai adathordozóra, így potenciálisan növelve a védelmet.
- Adathordozó elhagyása / lopása, fájlok véletlen törlése biztonsági mentés nélkül adat teljes elvesztését eredményezheti.
- Több hasonló fájl esetén gyakori jelenség az adatduplikáció.

2.2.2. Adatbázis

Adatbázisok kapcsán minden bizonnyal hallottunk már az SQL-ről, legtöbb relációs adatbázis ezt a nyelvet használja, egy struktúrált lekérdező nyelv. Feltételezem az SQL ismeretét, ezért bővebb jellemzésére nem tér ki a dokumentum.
Adatbázisok viszont lehetnek különböző típusúak, különböző adatmodellűek. Két típust fogunk megnézni, a relációs adatbázist és NoSQL adatbázist. Ennek a két adatmodellnek egy rövid összehasonlítását tartalmazza a 2.2-es táblázat.

2.2. táblázat. Relációs és NoSQL adatbázis összehasonlító táblázat

Relációs adatbázis	**NoSQL adatbázis**
Adatok összefüggése táblák közötti kapcsolattal határozható meg. Struktúrált formátumú tárolási mód.	Dinamikus séma, nem struktúrált adatnak nevezzük. Nem relációs adatmodell. Adatok gyakran kulcs-érték párokban szerepelnek vagy JSON formátumban.
Felépítéséből adódóan adatmodell változásokat nehezebb implementálni.	Könnyebb implementálni adatmodell változásokat, mivel az adatok nem biztos, hogy kapcsolódnak egymáshoz.
Fix, meghatározott adat felépítés esetén a legjobb használni.	Nem struktúrált, komplex adatok esetén gyorsabb, kisebb erőforrás igényű lehet, mint egy relációs adatbázis használata.

Azonfelül, hogy egy adatbázis milyen adatmodellű, lehet lokális, a felhasználó eszközén tárolt és távoli, felhőalapú, cloud adatbázis.
Térjünk ki a lokális és felhőalapú adatbázisok közötti összehasonlításra:

Lokális adatbázis [3], hasonló előnyei vannak, mint egy helyileg tárolt fájlnak:

- Egyszerű hozzáférés az adatokhoz, teljes adat kontrollálás.
- Nincs szükség internetkapcsolatra.
- Gyorsabb, mint egy távoli adatbázis, mivel gyorsabb a helyi lemez elérése, mint egy távoli esetében a hálózaton keresztüli kommunikáció.

Hátrány:

- Nehéz az adatmegosztás külső résztvevővel (másik számítógép lokális adatbázisa).
- Ha az eszköz olyan állapotba kerül, hogy a fájlokhoz nem lehet hozzáférni, akkor elveszlik minden adat.

Felhőalapú adatbázis [4]:

- Bárhonnan hozzáférhető, internet elérés szükséges. Negatívuma ugyanebből származik, ha nincs internet kapcsolat, nem használható.
- Adatok nem helyileg a számítógépen tárolódnak, emiatt lassabb lehet a hozzáférés. Figyelembe kell venni, hogy más szerverek / alkalmazások is használhatják ugyanazt a hálózatot, ami szintén befolyásolhatja az elérési sebességet.
- Teknikai hibák ellenére (felhasználó eszköze meghibásodik) az információ sértetlen marad a távoli adatbázison.
- Cloud adatbázis használata esetén a szinkronizáció bonyolult problémája kikerülhető. Több eszköz használhatja ugyanazt az adatbázist, ugyanazokkal az adatokkal dolgozhatnak.

2.3. Adat állapot

Adat állapota alatt három lehetséges esetet értünk, data-at-rest, data-in-motion és data-in-use. Az alfejezet a data-at-rest és data-in-motion [5] adatállapot fogalmát, biztonságukkal kapcsolatos információkat írja le.

2.3.1. Data-at-rest

Nyugvó adatnak olyan adatokat nevezünk, amelyek nem mozognak eszközről eszközre, vagy hálózatról hálózatra. Általában merevlemezen vagy pendrive-on tárolódnak.

A nyugalmi adatvédelem célja a bármilyen eszközön vagy hálózaton tárolt inaktív adat védelme.

Nyugvó adatokat általában kevésbé sebezhetőnek tartják, mint a mozgásban lévő adatokat (data-in-motion), gyakran értékesebbnek is tekinthető tartalmuk. Nyugvó adatok esetében az ellopható információ mennyisége sokkal nagyobb lehet mint az éppen úton levő adatoké (data-in-motion).

A nyugvó adatok biztonsága a szükséges óvintézkedések megtételétől függ.
Egy cég legtöbb esetben adatit saját hálózatán belül tárolja, ettől függetlenül veszélyben lehetnek rosszindulatú külső és belső fenyegetésekkel szemben. Egy betolakodó könnyedén elérheti az adott cég adatait, ha sikerül jogtalanul hozzáfernie egy számítógépükhöz vagy egy lopott eszközt feltörnie.

Data-at-rest típusú adatok védelme érdekében az egyik legjobb, legegyszerűbb és leggyakrabban alkalmazott módszer ezek titkosítása.

2.3.2. Data-in-motion / data-in-transit

Mozgásban lévő adatnak vagy tranzitadatnak olyan adatokat nevezünk, amelyek aktívan mozgásban vannak egyik helyről a másikra, vagy az interneten vagy magánhálózaton keresztül.

Mivel az adatok mozgásban vannak, ezért kevésbé biztonságosnak tekinthetők. Célja olyan adatok védelme amelyek például belső hálózaton belül mozognak, vagy helyi tárolóeszközről felhőtípusú tárolóeszközre.

Tranzitadat esetében is egy kiváló biztonsági intézkedés az adatok titkosítása. Védi az adatokat, ha két fél közötti kommunikációt 'lehallgatják'. Ez a védelem az adatok titkosításával biztosítható, még mozgatás előtt, vagy magának a kommunikációs csatornának titkosításával. Ez lehet talán a legfontosabb része a mozgásban lévő adatok védelmének, illetve a megfelelő kulcskezelés. Végpontok hitelesítése és adat érkezésekor való visszafejtése és ellenőrzése is tovább fokozhatja a védelmet.

2.4. Adattitkosítási módszerek

Ezt a szegmenst két részre bontható. Először az adatbázis titkosítási módszerek kerülnek bemutatásra, ezek olyan titkosítási eljárások, amik kizárólag adatbázisok és adataik titkosítására lettek kifejlesztve. Másik csoportba minden más olyan titkosítási módszer tartozik, amik nem adatbázishoz köthetők (fájlok, fájlrendszerek).

2.4.1. Adatbázis titkosítási módszerek

Olyan folyamatot nevezünk adatbázis titkosításnak, ami egy vagy több titkosítási algoritmus használatával az adatbázisban tárolt adatokat titkosított szöveggé (cipher text) alakítja. Ez a szöveg értelmezhetetlen a megfelelő kulcs ismerete nélkül [6].
Célja, hogy megvédje az adatainkat a potenciális fenyegetésektől. Ha egy hacker valahogy sikeresen feltöri az adatbázist, akkor számára értéktelen, értelmezhetetlen szöveggel fog találkozni.
Többféle titkosítási technika is létezik, melyek közül a legleterjedtebbek következnek.

Transparent Data Encryption (TDE) (Átlátható adat titkosítás):

- Teljes adatbázist, nyugvó adatokat (data-at-rest) titkosít merevlemezen és a biztonsági mentési adathordozón is. Használatban és szállításban lévő adatokat nem véd (data-in-use, data-in-transit).
- A módszer biztosítja, ha még el is lopják a fizikai adathordozót, akkor sem férnek hozzá a tolvajok az eszközön lévő adatokhoz.
- Mivel az összes adatot titkosítja, ezért nem szükséges speciális módon rendezni az adatokat.
- Adatok titkosítása tároláskor történik, visszafejtésük pedig rendszer memóriába való hívásakor történik.
- Szimmetrikus kulcsot használ a kódoláshoz.
- Microsoft, Oracle, IBM is alkalmazza ezt a módszert adatbázis fájlok titkosítása érdekében.

Column Level Encryption (Oszlop szintű titkosítás):

- Relációs adatmodell esetén egy adatbázis táblákból, oszlopokból, sorokból és cellákból/mezőkből áll. Ahogy a módszer nevéből is következik, relációs adatbázis egy sorát titkoítja.
- Lehetséges független oszlopok titkosítása. Egy oszlop összes adatát titkosítja kivétel nélkül.
- Akkor használatos, ha nincs szükség teljes adatbázis titkosításra, egyértelműen megkülönböztethető, hogy mely oszlopok tárolnak érzékeny adatot és melyek nem.

- Előnye, hogy könnyedén megkülönböztethető az érzékeny és a kevésbé érzékeny adat, illetve külön kulcs használható minden oszlop titkosításához, így növelve a biztonságot. Sokkal rugalmasabb, mint a teljes adatbázist titkosító TDE.
- Hátránya az előnyeiből fakad. Több oszlop több kulccsal való titkosítása az adatbázis teljesítményének csökkenéséhez vezethet. Lassabban lehet keresni és indexelni is.
- Microsoft, Oracle, IBM , MyDiamo és még sok más cég használja ezt a titkosítási módszert.

Field / Cell Level Encryption (Cella/mező szintű titkosítás):

- Relációs adatmodell esetén használható.
- Kiválasztható, hogy pontosan melyik mezőt szeretnénk titkosítani.
- Akkor használatos, ha nincs szükség teljes adatbázis titkosításra, hanem megkülönböztethető, hogy mely cellák tárolnak érzékeny adatot és melyek nem.
- Előnyei és hátrányai megegyeznek a Column Level Encryption-ével.
- Nem biztos, hogy minden esetben szükséges a mezők dekódolása, lehetőség van egyenlőség vizsgálatra.
- Microsoft, Oracle, IBM , MyDiamo és még sok más cég használja ezt a titkosítási módszert.

2.4.2. Egyéb titkosítási módszerek

Filesystem Encryption: [7]

- Fájlrendszer titkosítás, szokás még fájl / mappa titkosításnak, FBE-nek (file-based encryption) is nevezni.
- Célja fájl(ok) tartalmának titkosítása.
- Előnye, hogy minden fájlt külön kulccsal titkosítható, így növelve a biztonságot.
- Egy kriptográfiai kulcs addig van a memóriában, amíg az adott fájl meg van nyitva.
- Aki fizikailag hozzáfér a tároló számítógéphez, az láthatja, hogy milyen nevű fájlok találhatóak a rendszeren, holott a tartalmukat nem tudja megnézni, amíg nem ismeri a kulcsot.
- Olyan adatokat is képes titkosítani, amelyek nem részei egy adatbázis rendszernek.

- Csökkenti a teljesítményt és operációs rendszer hozzáférést is kíván a használatához.
- Teljesítmény problémák miatt nem igazán alkalmazzák, de ennek ellenére kis felhasználószámú rendszerek esetében ajánlott.

Full Disk Encryption: [8]

- Merevlemez teljes tartalma titkosításra kerül.
- Általában ugyanazt a kulcsot használja az egész meghajtó titkosításához, ezért futásidőben az összes adat visszafejthető.
- Nagy hátránya, ha a támadó futásidőben fér hozzá a számítógéphez, minden fájl elérhető számára.

Application Level Encryption: [9]

- Kódolás és dekódolás adatátvitel és tárolás előtt történik.
- Maga az alkalmazás végzi a titkosítási folyamatot.
- Az adat csak a megfelelő alklamazáson keresztül érhető el. Egy hacker-nek szüksége van az adatbázis és az adatokat használó alkalmazásra is az adatok visszafejtéséhez.
- Negatívuma lehet, ha ezt a titkosítási módszert szeretné alkalmazni egy cég, akkor maguknak kell implementálniuk, ami egy nem informatikai cég esetében bonyolult probléma lehet.
- Másik hátránya, a kulcsok kezelésének az összetettsége is megnövekedhet, ha több, különböző alkalmazásnak kell ugyanazon adatbázishoz hozzáférnie, írnia, olvasnia.

2.5. Szinkronizáció

Szóba került egy probléma többgépes alkalmazás készítése esetében, mégpedig az adatok szinkronizálása különböző eszközökön [10].

Az adatszinkronizálás egy olyan folyamat, ami alatt a különböző eszközökön eltárolt információkat megpróbáljuk össze egyeztetni, összhangba hozni.

Alkalmazások készítésekor szükséges a szinkronizáció problémájának megoldása, hiszen nem szeretnénk ugyanazt az információt bevinni a rendszerünkbe, amit már egy másik eszközön korábban megtettünk. Legtöbb esetben szeretnénk azonos adatokat elérni minden készüléken.

Programozási szempontból ez adatbázisok és/vagy fájlok szinkronizálását jelenti.

A probléma részletes felvezetése a következőképpen nézne ki:

Tegyük fel van két fájl, A és B, amelyek két lassú kommunikációs kapcsolattal (slow-communication link) összekötött gépen vannak. A-t úgy szeretnénk frissíteni, hogy tartalma megegyezzen B tartalmával. Ha A nagyméretű, akkor A másolása B-re lassú lesz. Gyorsaság érdekében A-t tömöríthetjük küldés előtt, de ez nem egy perfekt megoldás.

Tegyük fel, hogy A és B nagyon hasonló, mondjuk ugyanazon eredeti fájlból származnak. Felgyorsítás érdekében a hasonlóság valamilyen módon történő kihasználása lenne a logikus lépés. Gyakori módszer, hogy A és B közötti különbségeket küldik el a felek egymásnak, majd ebből a listából a fájl rekonstruálásra kerül.

A probléma az ilyen módszerekkel, hogy a különbségek kialakításához olvashatónak kell lennie mindkét fájlnak, ami csak úgy érhető el, ha a kapcsolat egyik végén mindkét fájl rendelkezésre áll. Ha nem található meg egy eszközön mindkét fájl, akkor nem használhatók ezek a módszerek.

Mondhatjuk, hogy ez a probléma felvetés lokális adatbázis fájlok szinkronizálása esetén is fennáll.

A szinkronizáció megoldása nem egy triviális probléma, nem is találtam rá specifikus 'how-to' jellegű leírást, lépésekre lebontott protokollt, ahogy két fájl szinkronizálási folyamata kéne történjen.
Megtaláltam viszont a szinkronizációnak két fajtáját [11]:

2.5.1. Egyirányú szinkronizáció (One-way synchronization)

Szokás még fájltükrözésnek (file mirroring), fájlreplikációnak (file replication) és fájlmentésnek (file backup) nevezni.

A fájlok várhatóan csak egy helyen változnak. A változtatások egyeztetése érdekében a szinkronizálási folyamat egy irányba másolja a fájlokat. A két tárolási helyszín nem tekinthető egyenértékűnek. Az egyik helyszín a forrás (source), a másik pedig a cél (target). Bármilyen változtatás a forrásba tükröződni fog a célba. A célon elvégzett változtatások nem fognak a forráson replikálódni.Ha ez a folyamat végigmegy, azt lehet mondani, hogy a forrás tükrözve van a célba.

Ez a módszer a forrás pontos másolatát hozza létre a célba. Hasznos és hatékony biz-

tonsági mentés szempontjából, mivel csak a változtatott / új fájlok másolódnak.

2.5.2. Kétirányú szinkronizáció (Two-way synchronization)

Gyors szinkronizálásnak is szokás nevezni (fast sync).

Ez a folyamat mindkét irányba másolja a fájlokat. A fájlok várhatóan mindkét helyen változnak, a két (vagy több) hely egyenértékűnek tekinthető.

Célja, hogy két vagy több hely azonos legyen egymással.

2.6. Kulcskezelés - Key management

Kulcs előállítását, cseréjét, tárolását és használatát jelenti [12].

Komplexitás szempontjából, minél több alkalmazás adata kerül titkosításra, úgy nő általában a tárolandó és kezelendő kulcsok száma is.

Kulcsok nem megfelelő tárolása és kezelése adatszivárgást eredményezhet. Ha a kulcskezelő rendszer valamilyen oknál fogva elveszti vagy törli a kulcsokat, akkor a titkosított adatok is elvesznek, feltéve ha nem készült a kulcsokról biztonsági másolat.

Vitatható, de úgy gondolom, hogy a megfelelő kulcskezelés a titkosítási folyamat legfontosabb része.

Kimondottan kulcskezelésre számos, úgynevezett kulcskezelő rendszereket (KMS) hoztak létre. Egy ilyen rendszer magába foglalja a kulcsok biztonságos generálását, cseréjét, tárolását.

Kulcskezeléshez tartozó fogalom a 'certificate' azaz tanúsítvány. A certificate egy szabványosított módszer egy adott felhasználó / applikáció / szerver hitelességének igazolására.

A maximum biztonság elérése érdekében általában egy harmadik-oldali ún. 'certificate authority'(CA) azaz tanúsítvány kiadó felel a certificate-ek kiosztásáért. Egy certificate a következő információkat tartalmazhatja:

- Szervezeti információk: egyértelmű vállalat / szervezet azonosítók pl név vagy cím.
- Certificate authority neve: a certificate előállítója ezzel az információval azonosítja magát.
- Digitális aláírás: az előállító ezzel az aláírással látja el a certificate-et, hogy ellenőrizhető legyen annak hitelessége. A megfelelő CA ellenőrzi a tanúsítványt, hogy egy hiteles szolgáltatótól származik-e.

3. fejezet

Titkosítási algoritmusok

A titkosítási algoritmusok fogalma eltérő az eljárásoktól, amiket adattitkosítási módszereknek neveztem. Egy titkosítási algoritmus olyan matematikai folyamat, amely egy szimpla szöveget (plain text) kódolt szöveggé transzformál (cipher text) [13].
Lehet szimmetrikus vagy aszimmetrikus egy algoritmus, ez a kódolt szöveg és a titkosítási kulcs(ok) közötti kapcsolatot írja le.

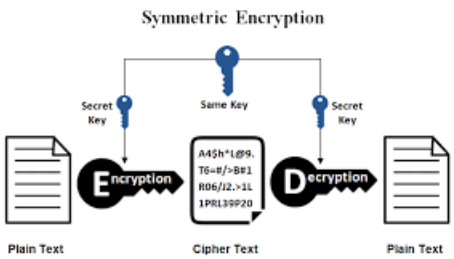
Mindegyik titkosítási módszer, legyen az adatbázis titkosítás, vagy fájlrendszer titkosítás, valamilyen titkosítási algoritmust alkalmaz.
A dolgozat méréseit és összehasonlításait ezeken az algoritmusokon végeztem.

Az algoritmusok belső matematikai működéséről részletes leírást nem tartalmaz a dokumentum. Bonyolultak és hosszúak, sok-sok oldalt kellene írni ahhoz, hogy megismerjük minden egyes algoritmus pontosan milyen számításokat végez a végső kódölt szöveg eléréséhez.

3.1. Szimmetrikus titkosítási algoritmusok

Adat titkosítására és dekódolására egy privát kulcsot használ [14].
Az adatok megosztásához a címzettnek rendelkeznie kell a visszafejtési kulcs másolatával. Ez a titkosítás legegyszerűbb, legrégebbi és legismertebb fajtája.
Hátránya a kulcs megosztásából ered, ha a visszafejtési kulcs publikussá válik, az adatszivárgáshoz vezethet.
Előnye a sebesség.

Általános működési elve a szimmetrikus algoritmusoknak a 3.1 ábrán látható.



3.1. ábra. Szimmetrikus titkosítási algoritmusok működése [15].

3.1.1. AES (Advanced Encryption Standard)

Bevált, megbízható titkosítási módszer.

Komplex, több fázisból álló matematikai számításokat hajt végre [16].

Hátránya, hogy szoftveresen nehezen implementálható lehet. Komplexitása és nagysága miatt teljesítményi ideje is nagy lehet.

Úgynevezett 'block cipher', azaz blokk titkosítás. A titkosítandó adatokat blokkokra osztja, egy blokk mérete 128 bit.

3 verziója létezik:

- AES-128 – 128 bit nagyságú titkosítási kulcsot alklamaz.
- AES-192 – 192 bit nagyságú titkosítási kulcsot alkalmaz.
- AES-256 – 256 bit nagyságú titkosítási kulcsot alkalmaz.

Számítást bájtokon nem pedig biteken végez. 128 bites blokkot az algoritmus 16 bájtként kezel. Ezt a 16 bájtot 4 sorba és oszlopba rendezi a mátrixműveletekhez.

Kulcs méretétől függ, hogy hány kört fog az algoritmus elvégezni. 128 bites kulcs – 10 kör, 192 bites kulcs – 12 kör, 256 bites kulcs – 14 kör.

Mindegyik kör egy másik 128 bites kulcsot használ, amit az eredeti AES kulcsból számolnak ki.

3.1.2. DES (Data Encryption Standard)

Block cipher, egy blokk 64 bit nagyságú [17].

64 bit nagyságú sima szöveg titkosítás után 64bit nagyságú titkosított szöveget eredményez.

16 sorozatot véget el matematikai számításokból, mindegyikhez külön titkosítási kulcsot használ.

Kulcsok mérete 56 bit (64 bit, de 8-at közülük nem használ).

Hátránya lehet, hogy a titkosítás és visszafejtés ugyanazzal az algoritmussal és kulcsokkal történik.

Feistel kódon alapszik.

3.1.3. Triple DES

Működése megegyezik a DES működésével.

3x16 sorozatot végez, így biztonságosabbnak mondható [18].

Hátránya lehet, hogy a titkosítás és visszafejtés ugyanazzal az algoritmussal és kulcsokkal történik.

3.1.4. Blowfish

Block cipher, 64 bit méretű blokkok [19].

Kulcsok mérete 32 bittől 448 bit-ig terjed, változó méretűek, így lehetőséget nyújt személyes és ipari felhasználásra is. Úgynevezett alkulcsokat is használ (subkey), számszerint 18-at.

Sokkal gyorsabb mint a DES és Triple DES.

3.2. Aszimmetrikus titkosítási algoritmusok

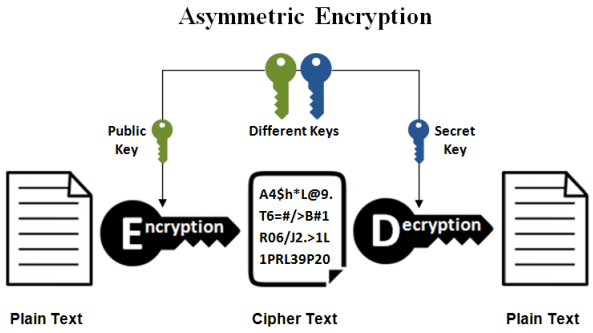
Egy privát és egy publikus kulcsot használ [20].

A publikus kulcs lehetővé teszi, hogy bárki titkosítsa az adatokat.

A privát kulcs szükséges az adatok dekódolásához.

Biztonságosabbnak vélhető, mivel a privát kulcsok nem kerülnek megosztásra, de ugyanakkor nagyobb a számítási költsége is.

Aszimmetrikus algoritmusok általános működési elve a 3.2 ábrán látható.



3.2. ábra. Aszimmetrikus titkosítási algoritmusok működése [15].

3.2.1. RSA (Rivest–Shamir–Adleman)

Azon az elven alapul, hogy a nagy számok szorzása könnyű, de a nagy számok tényezőkre bontása nehéz [21].

A publikus kulcs két számból áll, ami közül az egyik két nagy prímszám szorzata.

A privát kulcs egyik száma ugyanennek a két prímnek a szorzata.

3.2.2. DSA (Digital Signature Algorithm)

Digitális aláírások titkosítására használt szabvány. Digitális üzenet hitelesítésére is alkalmas [20].

Moduláris exponenciáláson (modular exponentiation) alapul, a diszkrét logaritmus (discrete logarithm) problémájával együtt.

Privát kulcsot egy üzenet digitális aláírásának generálására használják. Ellenőrizni az aláíró publikus kulcsával lehetséges.

3.2.3. ECC (Elliptic Curve Cryptography)

RSA-val vetélkedő aszimmetrikus titkosítási módszer [21].

Kulcs generálása matematikai elliptikus görbék segítségével történik.

Alapvető kulcs méret 256 bit, de görbétől függően változhat.

3.3. Hash, hashelés, hash algoritmusok

A titkosítási módszerek közül kimaradt, az úgynevezett 'hashing', vagy hashelés. Nem sorolható kimondottan sem az adatbázis sem a fájlrendszerek titkosításához.

Érzékeny adatok titkosítására használják, leggyakrabban jelszavakhoz [22].

Egyediek és ismételhetőek, ami azt jelenti, hogy egy szó ugyanazzal a hash algoritmussal transzformálva ugyanazt a titkosított szöveget fogja eredményezni.

Egy algoritmus mindig ugyanolyan méretű kimenetet állít elő.

Nagyon nehéz két olyan különböző szót találni, amelyek ugyanazon hash algoritmussal transzformálva ugyanazt a titkosított szöveget eredményeznék.

Egy hash algoritmussal transzformált szöveget visszaalakítani egyszerű, értelmes szöveggé (plain text) már nem lehet, éppen ezért használják jelszavak titkosítására.

Nem visszaalakíthatósága miatt egyezőség vizsgálatát lehet megnézni két ugyanazon hash algoritmussal transzformált , kódolt szöveg között. Ha két hash egyezik, akkor ugyan az volt a bemeneti érték, ha nem egyeznek, akkor különbözőek voltak.

Úgynevezett salt és pepper –rel szokás a hasheket ellátni, magasabb biztonság elérése érdekében.

Salt: A titkosítandó szövegrészhez extra szöveg csatolása bonyolultság növeléséhez. Regisztrációkor lehet például az email címet és jelszót kombinálni, majd a kombinált szövegen elvégezni a hash algoritmust. Ilyen adatokat 'salted hash'-nek nevezünk.

Pepper: Salted hash adathoz , tehát már titkosított adathoz fűz további értékeket. Egy adatbázis esetében ez általában megegyező értéket jelent, azaz minden hashelt adathoz ugyanaz a pepper érték adja hozzá.

Két leggyakrabban implementált hash algoritmus:

3.3.1. MD5 (Message Digest 5)

Régebben gyakran használt funkció, ami egy 128 bites kimenetet állít elő [23].

Kiderült róla, hogy nem ütközésálló (collision resistant), emiatt kriptográfusok más hash algoritmusokat ajánlanak. Egy hash függvényre akkor mondjuk, hogy ütközésálló, hogyha nehéz olyan két különböző bemenetet találni, amely ugyanazt a kimenetet eredményezi.

3.3.2. SHA (Secure Hashing Algorithm) Family

Hat különböző hash függvényből áll: SHA-0, SHA-1, SHA-224, SHA-256, SHA-384, SHA-512 [22].

Változó méretű bemenetet alakítanak fix méretű kimentté.

Az első négy 512 bites blokkokat használ 32 bites szavakra osztva, az utolsó kettő pedig 1024 bites blokkokat 64 bites szavakra bontva.

Kimenet mérete SHA-0 és 1 esetén 160 bit, SHA-224 esetén 224 bit, SHA-256 esetén 256 bit, SHA-384 esetén 384 bit, SHA-512 esetén 512 bit nagyságú.

Mindegyik algoritmus hasonlóképpen működik.

Eredmény mindig 160 bit hosszú. Eredeti üzenet hosszának kevesebbnek kell lennie, mint 2^{64}-en bit.

A SHA-2 családot, aminek a 256, 384, 512 is a tagja, szélés körben implementálták biztonsági alkalmazásokban és protokollokban, mint a TLS, SSL, PGP, stb.

A SHA-256-ot a Debian szoftvercsomag hitelesítésére használják és DKIM üzenetaláírási standard. Linux és Unix gyártók 256 és 512 bites SHA-2 használatára tértek át a biztonságos jelszótároláshoz.

Számos kriptovaluta, köztük a Bitcoin is használja a SHA-256-ot.

3.4. Mérések, összehasonlítások

A mérések végzéséhez az alkalmazásomból elmentett adatokat fogom tesztadatként használni. Az adatokat struktúrált xml dokumentum formátumban kerülnek tárolásra az összehasonlítások elvégzése érdekében. Három különböző méretű fájlt vizsgálunk. Reális méretű fájlok használatára törekedtem (500KB, 2MB 4MB). Már a 4 MB nagyság (nem titkosított) elérését a program rendeltetésszerű használata esetén is elég valószínűtlen tartom, mivel nagyon sok random adatot kellett bemásoljak ahhoz, hogy elérjem ezt a fájlméretet.

Feltehető a mérési eredmények arányos növekedése nagyobb fájlméret esetén.

3.4.1. Tesztprogram

A mérések összeállítása előtti próbálgatásaimból megtudtam, hogy a titkosítási és visszafejtési idő (a program kód alapján, a System.currentTimeMillis() parancs használatával) nem mindig egyezik meg ugyanazon fájl többszöri titkosítása esetén. Ebből következik, hogy egy adott fájl titkosítását mindegyik algoritmus használatakor többször is el kell végeznem, hogy egy korrekt közelítő értéket kapjak.

A tesztprogram elkészítésére elsősorban a javax.crypto és java.security könyvtárakat használtam. A program szempontjából ezen könyvtárak által szolgáltatott fontosabb interface-ek:

Key: Az összes kulcs legfelső szintű interface-e. Meghatározza az összes kulcs objektum által használt funkciókat.

PrivateKey: Célja, hogy csoportosítsa az összes privát kulcs interface-t. Nem tartalmaz metódusokat vagy konstansokat. Speciális privát kulcs interface-ek kiterjesztik ezt az interface-t.

PublicKey: Célja, hogy csoportosítsa az összes nyilvános kulcs interface-t. Nem tartal-

maz metódusokat vagy konstansokat. Speciális nyilvános kulcs interface-ek kiterjesztik ezt az interface-t.

Osztályok:

-SecretKeySpec: Használható titkos kulcs létrehozására byte tömbből anélkül, hogy SecretKeyFactory-t kellene hozzá használni. 'Nyers' titkosítási kulcsok számára különösen hasznos, amelyek ábrázolhatók byte tömbként és nincs hozzájuk kulcsparaméter társítva.

KeyPair: Egy kulcspár (egy nyilvános és egy privát kulcs) egyszerű tárolója.

KeyGenerator: Egy titkos (szimmetrikus) kulcsgenerátor funkcionalitását biztosítja.

KeyPairGenerator: Nyilvános és privát kulcspárok létrehozására szolgál.

Cipher: Kriptográfiai titkosítási funkciókat biztosít titkosításhoz és visszafejtéshez. A Java Cryptographic Extension (JCE) keretrendszer magját képzi.

A tesztprogram felépítése hasonló szimmetrikus és aszimmetrikus titkosítás esetén is, néhány eltéréssel.
A tesztprogram teljesen reprodukálható a felsorolt osztályok és interface-ek használatával.

Lényegi különbség a két alkalmazás között (szimmetrikus és aszimmetrikus) a kulcsok méretében, létrehozásában / generálásában rejlik. A kódrészletekbe nem megengedett az ékezetes betűk használata, ezért ékezet nélkül kerültek bele magyar szavak.

- Fájl elérési útvonal, jelen esetben a vizsgált xml fájlok útvonala, amik a tesztek készítésekor az asztalomon voltak.

```
String filePath = "C:\\Users\\AMD\\Desktop\2MB.xml";
File inputFile = new File(filePath);
```

- Az objektum és metódus, ami a titkosítást és dekódolást végzi. Az implementálás RSA esetén így nézett ki:

```
//Titkositas
Cipher cipher = Cipher.getInstance("RSA");
cipher.init(Cipher.ENCRYPT_MODE, publicKey);
byte[] encryptedFileBytes = cipher.doFinal(inputBytes);

//Visszafejtes
cipher.init(Cipher.DECRYPT_MODE, privateKey);
byte[] decryptedFileBytes = cipher.doFinal(inputBytes);
```

Szimmetrikus algoritmus használata esetén a publicKey és privateKey helyére a létrehozott szimmetrikus kulcsot kell elhelyezni.

- Kódolt byte-ok fájlba írása, majd a dekódolt byte-ok másik fájlba írása. Azért dekódoltam, hogy meggyőződjek róla, hogy semmi hiba nem történt titkosítás és visszafejtés alatt.

```
//Titkositott byte-ok irasa
```

```
FileOutputStream outputStream =
                            new FileOutputStream(encryptedFile);
outputStream.write(encryptedFileBytes);

//Dekodolt byte-ok irasa
outputStream = new FileOutputStream(decryptedFile);
outputStream.write(decryptedFileBytes);

//Ezek lezarasa a megfelelo helyen
outputStream.close();
inputStream.close();
```

- Az eltelt idő mérésére az említett System.currentTimeMillis() funkció:

```
//Kezdo ertek
long start = System.currentTimeMillis();
//Vegso ertek
long end = System.currentTimeMillis();
//Korrekt idotartam megkapasa
long finalTime = end2-start2;
```

A kezdő és végérték inicializálását a megfelelő helyekre kell elhelyezni. Titkosítás és visszafejtés esetén is a kezdőértéket a Cipher objektum létrehozása elé raktam, a kulcs létrehozását nem számítottam bele az időbe. A végértéket a titkosított/dekódolt byte-ok fájlba történő írása után.

Ahol a legnagyobb a kód különbsége, a kulcsok létrehozása. Először is meg kell jegyezzem, hogy két módon lehet megfelelő kulcsokat létrehozni:

A megfelelő generátor objektummal (szimmetrikus - KeyGenerator, aszimmetrikus - KeyPairGenerator).

Létrehozni egy String-et, majd abból egy Key objektumot a SecretKeySpec osztály segítségével.

Az aszimmetrikus algoritmusok méréséhez az első verziót alkalmaztam, a szimmetrikus algoritmusokhoz pedig a másodikat.

-Szimmetrikus algoritmus méréseknél a kulcs létrehozása a következő módon nézett ki DES használata esetén:

```
String key = "T@stK#Y1";
//Titkos kulcs letrehozasa
Key secretKey = new SecretKeySpec(key.getBytes(), "DES");
```

Az alkalmazás többi részében a secretKey objektumot kellett használni.

- Aszimmetrikus algoritmusok mérésekor a következő volt a kód:

```
KeyPairGenerator generator =
                            KeyPairGenerator.getInstance("RSA");
generator.initialize(2048);
KeyPair pair = generator.generateKeyPair();
```

```
PrivateKey privateKey = pair.getPrivate();
PublicKey publicKey = pair.getPublic();
```

Titkosításhoz a publicKey-t kell használni, visszafejtéshez pedig a privateKey-t.

3.4.2. Eredmények, következtetések

A számítógépem paraméterei:
Processzor - AMD Ryzen 5 3600X 6-Core 3.79GHz
RAM - 16 GB
OS - Windows 10 Home 64 - bit

Az előzőleg ismertetett szimmetrikus titkosítási algoritmusok tulajdonságainak összehasonlítását tartalmazza a 3.1-es táblázat.

3.1. táblázat. Ismertetett szimmetrikus titkosítási algoritmusok összehasonlítása

	AES	**DES**	**Triple DES**	**Blowfish**
Lehetséges kulcs méret(ek)	128/192/256 bit	64 bit (56-ot használ)	Összesen 168 bit	Mérete 32 bittől 448 bitig terjedhet
Mekkora egy block mérete?	128 bit	64 bit	64 bit	64 bit
Hány kört végez az algoritmus?	Kulcs méretétől függ 128 bit - 10 kör 192 bit - 12 kör 256 bit - 14 kör	16 kör	3 x 16 kör	16 kör
Külön kulcsot használ-e minden körben?	Igen, mindegyik kör más kulcsot használ.	Igen, különböző 48 bites kulcsokat.	Három különböző DES kulcsot.	Igen, 32 bit nagyságúakat.
Sikerült-e már feltörni?	Nem.	Igen, úgynevezett brute force támadással.	Vegyes válaszokat találtam, de mivel már nem igazán használják csak régebbi szoftverek, és a DES is fel lett törve, ezért valószínűleg **igen**.	Nem.

Úgy gondolom, hogy az algoritmusok biztonságával kapcsolatosan megfelelő méréseket nem tudok végezni, ezért nem szerepelnek a dokumentumban. A táblázat 'Sikerült-e már feltörni' sorában lévő információk is az interneten talált adatok, ezekből lehet valamilyen szinten az algoritmus biztonságára vonatkoztatni.

Az előző alfejezetben bemutatott tesztprogramot használva megvizsgáltam 500kb, 1mb, 2mb, 3mb és 4mb nagyságú fájl titkosításának sebességét a szimmetrikus algoritmusokat használva. A 3.2-es, 3.3-as, és A.1-től A.8-ig tartó táblázatok a mérések eredményeit tartalmazzák.
Az AES algoritmusok 128 bit nagyságú kulcsot használtak.
A táblázatokban szereplő értékek milliszekundumban (ms) értendők.

3.2. táblázat. 500KB-hoz tartozó titkosítási mérések

Titkosítás 500KB	**AES**	**DES**	**Triple DES**	**Blowfish**
1.mérés	49	51	73	48
2.mérés	44	50	69	44
3.mérés	46	50	70	49
4.mérés	43	50	74	49
5.mérés	48	56	71	46
6.mérés	46	57	74	50
7.mérés	44	53	74	52
8.mérés	44	57	74	52
9.mérés	47	58	70	50
10.mérés	61	58	70	46
Átlag	**46,2**	**54**	**71,9**	**48,6**

3.3. táblázat. 500KB-hoz tartozó visszafejtési mérések

Visszafejtés 500KB	**AES**	**DES**	**Triple DES**	**Blowfish**
1.mérés	7	17	28	14
2.mérés	7	21	28	13
3.mérés	6	17	29	12
4.mérés	7	16	35	13
5.mérés	7	17	28	13
6.mérés	7	18	35	12
7.mérés	8	16	33	14
8.mérés	6	20	29	15
9.mérés	8	17	29	14
10.mérés	8	19	30	14
Átlag	**7,2**	**17,8**	**30,4**	**13,4**

A fájl mérete titkosítás előtt és után minden esetben megegyezett, kivéve pár bájt különbséget.

Az átlagértékeket a 3.4-es és 3.5-ös táblázatokba gyűjtöttem össze jobb átláthatóság érdekében.

3.4. táblázat. Titkosítási átlagértékek

Titkosítás	**AES**	**DES**	**Triple DES**	**Blowfish**
500KB	46,2 ms	54 ms	71,9 ms	48,6 ms
1MB	54,2 ms	67,7 ms	104,4 ms	59,2 ms
2MB	56,7 ms	88,6 ms	153,9 ms	69,8 ms
3MB	61,1 ms	110,5 ms	217,4 ms	82,7 ms
4MB	63,2 ms	132 ms	269,2 ms	93,7 ms

3.5. táblázat. Visszafejtési átlagértékek

Visszafejtés	**AES**	**DES**	**Triple DES**	**Blowfish**
500KB	7,2 ms	17,8 ms	30,4 ms	13,4 ms
1MB	12 ms	29,9 ms	59,3 ms	19,2 ms
2MB	18 ms	48,5 ms	115,7 ms	33 ms
3MB	21,2 ms	67,9 ms	174 ms	42,4 ms
4MB	23 ms	85 ms	229,1 ms	50,7 ms

Néhány esetben viszonylag nagy eltérés volt az értékek között, például a Triple DES esetében, a 4MB méret 5.mérése (lásd A.7 táblázat) 50 milliszekundummal nagyobb mint a többi mérés. Pontosabb átlagértékeket kaphatnánk, ha egy módszerrel X méretű fájlt százszor titkosítanánk, nem pedig csak tízszer, és az így kapott eredményeket átlagolnánk.

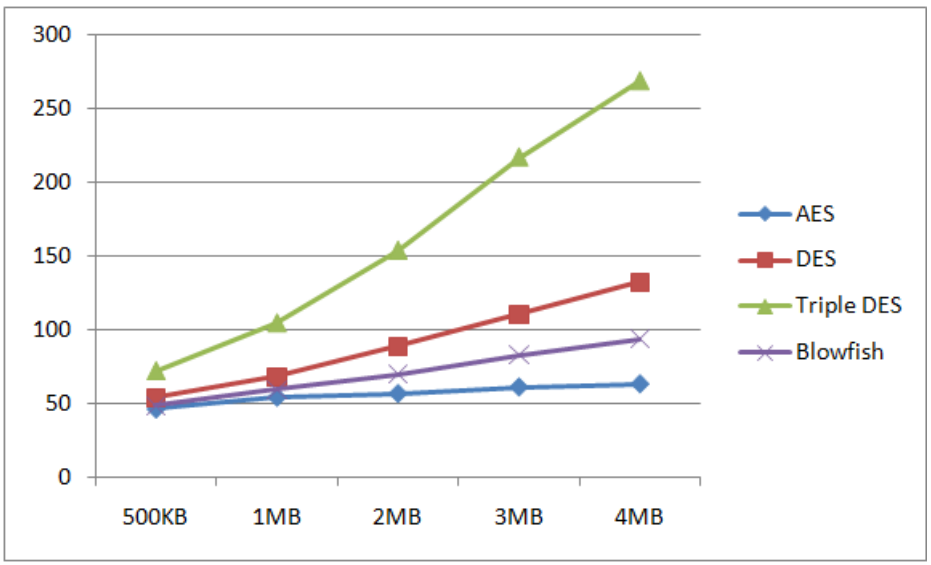
Ettől függetlenül az így kapott értékekről lehet következtetéseket vonni:

Mindkét téren (titkosítás és visszafejtés) az AES használata bizonyult a leggyorsabbnak (lásd 3.4 és 3.5 táblázat). Ahogy a fájl mérete nőtt, úgy az AES titkosítási ideje nem nőtt olyan látványosan, mint a többi algoritmus esetében. Visszafejtési időre szintén igaz ez az állítás.

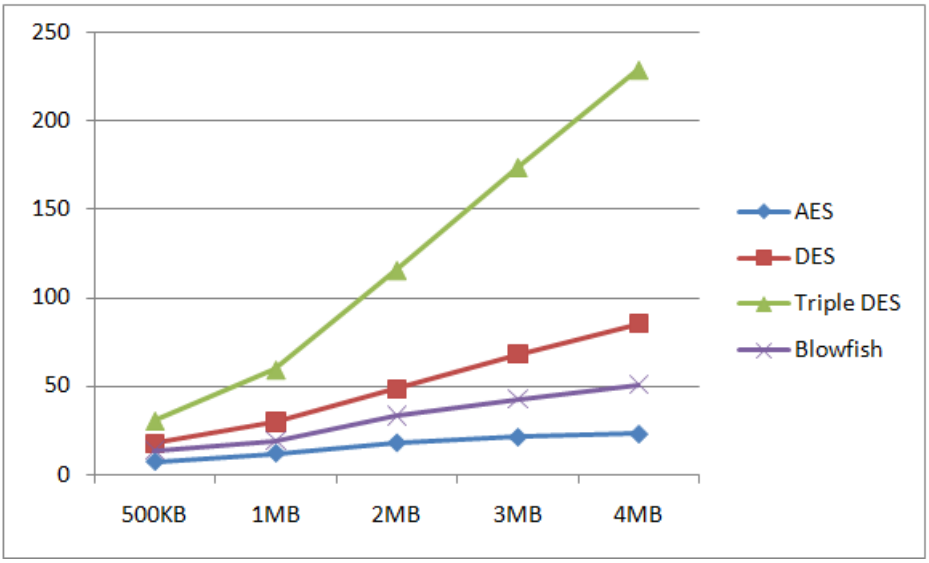
Gyorsaságot tekintve legközelebb az AES-hez a Blowfish állt, mind visszafejtés és titkosítás terén is.

A DES és a Triple DES közötti egyre inkább növekvő különbség várható volt, hiszen a Triple DES háromszor végzi el azt, amit a DES csak egyszer. Ennek ellenére titkosítási időben nem érte el a Triple DES az elődje háromszorosát, habár nagyobb fájlméret esetén a növekedésüket nézve valószínűleg el fogja, sőt ezt a tendenciát követve valószínűleg a különbség több, mint a háromszorosára is nőhet. Visszafejtési időben sokkal inkább látszik ez a különbség, 4 MB-os fájl titkosítása átlagosan kétszer addig tart a Triple DES-nek, mint a DES-nek, visszafejtést nézve 4 MB esetén ez az érték majdnem háromszorosára nőtt (lásd 3.4 és 3.5 táblázat).

Az átlagértékek szemléltetésére grafikonok is készültek, amik a 3.3 és 3.4-es ábrán láthatók. A függőleges tengely a titkosítási időket tartalmazza milliszekundumban mérve, a vízszintes tengely pedig a fájlok méretét.



3.3. ábra. Szimmetrikus titkosítási mérések



3.4. ábra. Szimmetrikus visszafejtési mérések

A program szempontjából valószínűtlennek tartom a 4MB-nál nagyobb fájl lehetőségét, de a mérések kedvéért 64MB nagyságú fájl titkosítását is elvégeztem ugyanilyen módon. A kapott értékek a 3.6-as és 3.7-es táblázatokban láthatók, amik szintén milliszekundumban értendők.

3.6. táblázat. 64MB-hoz tartozó titkosítási mérések

Titkosítás 64MB	**AES**	**DES**	**Triple DES**	**Blowfish**
1.mérés	392	1678	3505	956
2.mérés	369	1314	3524	950
3.mérés	359	1703	3522	968
4.mérés	411	1688	3512	993
5.mérés	364	1676	3515	979
6.mérés	377	1682	3514	974
7.mérés	354	1311	3514	974
8.mérés	374	1328	4351	974
9.mérés	377	1693	3518	1089
10.mérés	366	1694	3501	997
Átlag	**374,3**	**1576,6**	**3597,6**	**985,4**

3.7. táblázat. 64MB-hoz tartozó visszafejtési mérések

Visszafejtés 64MB	**AES**	**DES**	**Triple DES**	**Blowfish**
1.mérés	298	1301	3451	735
2.mérés	314	1272	3464	775
3.mérés	300	1323	3516	755
4.mérés	297	1297	3452	762
5.mérés	289	1305	3447	747
6.mérés	289	1309	3438	755
7.mérés	285	1560	3441	767
8.mérés	401	1297	4269	763
9.mérés	308	1299	3435	764
10.mérés	291	1298	3453	750
Átlag	**307,2**	**1326,1**	**3536,6**	**757,3**

A 4MB-os táblázatok értékeivel összehasonlítva ezeket az értékeket a következőt kapjuk:

Az AES titkosítási ideje 5,92-szeresére nőtt, amíg a visszafejtési ideje 13,35-szörösére.

A DES titkosítási ideje 11,94-szeresére nőtt, amíg a visszafejtési ideje 15,6-szorosára.

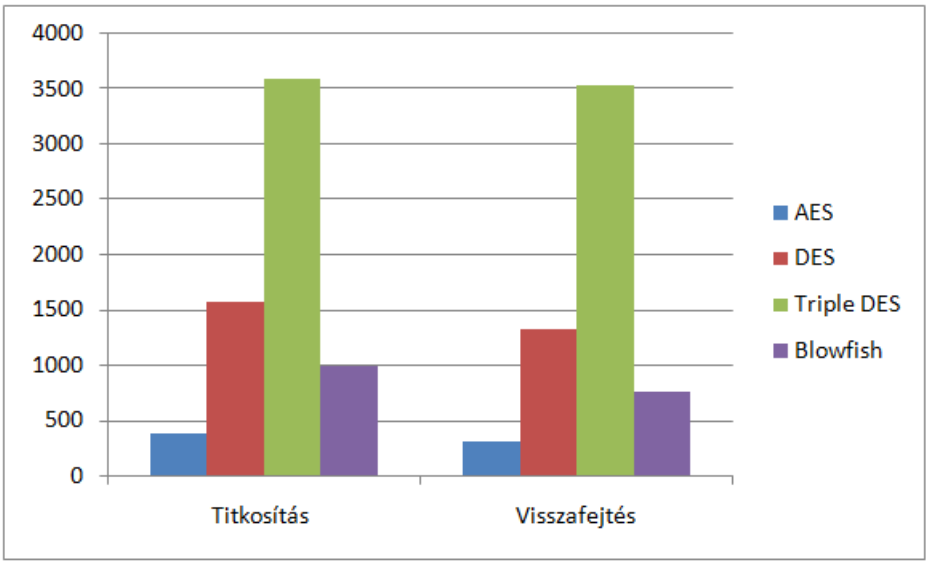
A TripleDES titkosítási ideje 13,36 -szorosára nőtt, amíg a visszafejtési ideje 15,44-szeresére.

A Blowfish titkosítási ideje 10,51-szeresére nőtt, amíg a visszafejtési ideje 14,94- szeresére.

Ezek az adatok azért lehetnek érdekesek, mivel ha a 4MB és 64MB-os fájlméretet nézzük, akkor igaz, hogy 64 / 4 = 12, így ezt a logikát követve 12-szeres időnövekedést várhatunk titkosítás és visszafejtés esetén is. Ennek ellenére nem ezt kapjuk, hol többet, hol kevesebbet, tehát az egyenes arányosság logikája nem alkalmazható az algoritmusok idejének becslésére.

Ezt az állítást igazolhatjuk az 500KB-1MB-2MB-4MB-os táblázatokat nézve is, mivel ezekben a táblázatokban sem duplázódtak meg pontosan az idők, annak ellenére, hogy a fájl mérete dupla akkora lett.

A 64 MB-os esethez külön diagramot készítettem, különben a skálázás miatt a kisebb méretű fájlok értékei alig látszottak volna (lásd 3.5 ábra).



3.5. ábra. 64MB-os fájl titkosítás, visszafejtés. Függőleges tengely - idő (ms)

Aszimmetrikus titkosítási algoritmusok:

RSA

Szerettem volna a szimmetrikus algoritmusokkal összehasonlítani, de nem lenne korrekt, mivel az adatmennyiség felsőhatára, amit a javax.crypto könyvtár használatával titkosítani tudtam 245 byte volt. Ha nagyobb mérettel próbálkoztam hibaüzenetet kaptam „Data must not be longer than 245 bytes".

Ezután az interneten jobban utána olvastam a problémának és megtudtam, hogy az RSA algoritmus maximum akkora mennyiségű adatot képes titkosítani, mint az RSA kulcs mérete (mínusz az ún. header data, azaz fejlécadat).

Kettő hatványait használtam kulcsméret megadására, először 2048 bit nagyságú kulccsal kezdtem.
Az eredmények milliszekundumban értendők és a 3.8-as és 3.9-es táblázatban találhatók.

3.8. táblázat. RSA titkosítási mérések különböző méretű fájlokkal

Titkosítás RSA	**64 byte**	**128 byte**	**245 byte**
1.mérés	28	24	28
2.mérés	28	24	30
3.mérés	25	26	30
4.mérés	23	26	28
5.mérés	24	25	28
6.mérés	26	21	25
7.mérés	27	23	22
8.mérés	30	28	22
9.mérés	29	26	21
10.mérés	25	28	25
Átlag	**26,5**	**25,1**	**25,9**

3.9. táblázat. RSA visszafejtési mérések különböző méretű fájlokkal

Visszafejtés RSA	**64 byte**	**128 byte**	**245 byte**
1.mérés	7	6	7
2.mérés	6	5	4
3.mérés	6	7	5
4.mérés	4	7	7
5.mérés	6	7	6
6.mérés	7	4	4
7.mérés	5	7	9
8.mérés	7	7	5
9.mérés	8	5	7
10.mérés	7	8	6
Átlag	**6,3**	**6,3**	**6**

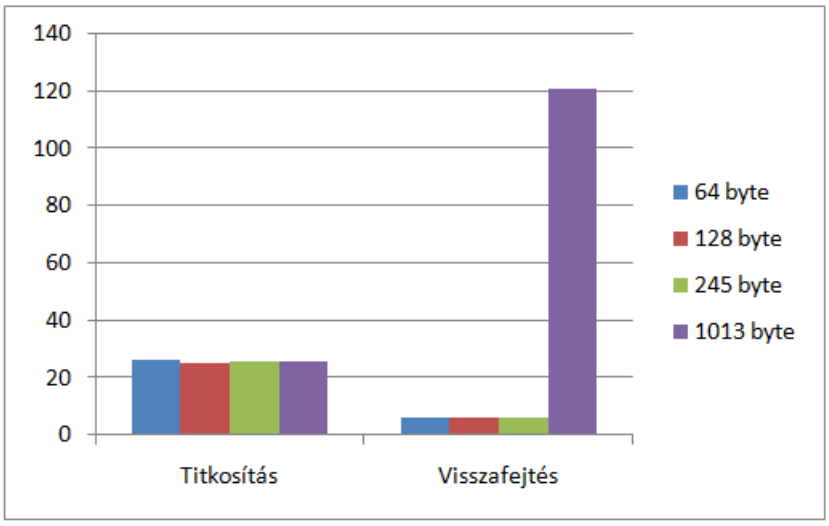
Ezután megpróbáltam nagyobb fájlokat titkosítani, így növelnem kellett a kulcs nagyságát is.

A kulcs nagysága: 8192 bit. Titkosítandó fájl nagysága: 1013 byte (a fejlécadat nagysága miatt ez lett a limit, de a kettőt összeadva egyébként 2^{10}-en, azaz 1024 byte). Titkosított fájl mérete: 1024 byte. Az eredmények a 3.10 táblázatban találhatók. A 4 különböző fájl átlagértékei grafikonos formában a 3.6-as ábrán láthatók.

3.10. táblázat. RSA titkosítás és visszafejtés 1024 byte nagyságú fájl esetén

1024 byte	**Titkosítás**	**Visszafejtés**
1.mérés	28	117
2.mérés	21	123
3.mérés	22	121
4.mérés	31	121
5.mérés	29	125
6.mérés	23	119
7.mérés	24	122
8.mérés	27	121
9.mérés	28	119
10.mérés	23	124
Átlag	**25,6**	**121,1**

Annak ellenére, hogy ezek az értékek nem tűnnek nagynak, a számítógépemnek elég sokáig tartott minden mérés elvégzése, volt néhol 10 másodperc is, mire az eredményt megkaptam. Mivel már az 1 kb-os fájl titkosítása is eddig tartott, nagyobb méretű fájllal nem próbálkoztam.



3.6. ábra. RSA titkosítás, visszafejtés. Függőleges tengely - idő (ms)

DSA mérését és összehasonlítását nem tartom korrektnek, mivel nem nagy méretű adat titkosítására, hanem digitális aláírásokra lett kifejlesztve.

ECC mérést szintén nem készítettem, mivel nem tudtam. Utánanéztem az interneten és ECC titkosítást elméletileg a javax.crypto könyvtár segítségével nem lehet végrehajtani. Más függvény könyvtárat azért nem használtam, mert úgy gondolom fennáll a lehetősége, hogy eltérnek az algoritmusok implementálásai, így olyan könyvtár használata lenne optimális, amely az összes vizsgált algoritmust tartalmazza.

Röviden összefoglalva, a tesztekkel és mérésekkel olyan eredményeket kaptam, amikre számítottam, annak ellenére, hogy a sebességük az előző alfejezetek leírásaiban nem lett számszerűsítve.
Az AES-nél nem értek egyet a talált állítással, miszerint "Hátránya, hogy szoftveresen nehezen implementálható lehet, illetve komplexitása és nagysága miatt teljesítményi ideje is nagy lehet".
A szoftveres implementálás függvény könyvtár használata miatt volt egyszerű, a matematikai részét nem nekem kellett megoldanom.
Nagy teljesítményi ideje pedig relatív, hiszen a többi vizsgált szimmetrikus algoritmushoz képest az AES-é volt a legkisebb.

4. fejezet

Tervezés

Ez a fejezet a program tervezésével kapcsolatos információkat, specifikációkat tartalmazza. Egy látványterv leírása a felhasználói felületről és a szükséges funkciókról.

4.1. Grafikus Felhasználói Felület (GUI) - megjelenés, funkció

Az alkalmazás két menüpontből álljon, jegyzetek és kereső menü. A két menü működése egyedi objektumok használatával történjen.

4.1.1. Szükséges adattípusok

Szükséges egy Note és egy Page objektum konstruktor létrehozása.
A Note objektumnak legyenek a következő változói:

- String noteName - A jegyzet neve.
- ArrayList<Page> pageList - Egy Page típusú ArrayList a jegyzet oldalainak tárolására. Egyszerűen lehet majd törölni és hozzáadni új Page objektumokat.
- int Id - Egy ID az egyszerű azonosítás érdekében.
- static int counter - Egy számláló, ami majd az ID-t növeli, maga a számláló értékét pedig a konstruktorban kell növelni.
- Minden adathoz a megfelelő getter-setter metódusok.

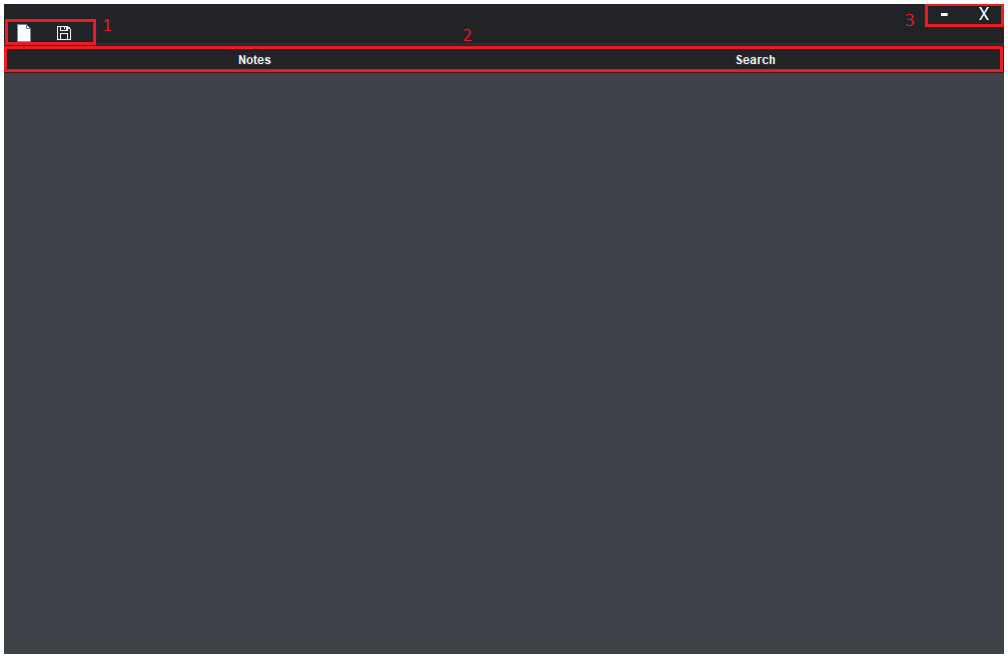
A Page objektumnak legyenek a következő változói:

- String pageName - Az oldal neve.
- String pageText - Az oldal tartalma, a szövegdobozba írt adatok.
- int noteId - A lapot tartalmazó jegyzet ID-je ahhoz, hogy egyszerűen meghatározható legyen a kapcsolat egy Note és egy Page objektum között.
- int Id - Egy ID az egyszerű azonosítás érdekében.

- static int counter - Egy számláló, ami majd az ID-t növeli, maga a számláló értékét pedig a konstruktorban kell növelni.
- Minden adathoz a megfelelő getter-setter metódusok.

4.1.2. Közös elemek

A következő elemek olyanok, amelyeknek mindkét menüpontban látszaniuk kell és használhatóak kell legyenek. A 4.1. ábrán pirossal bekeretezett és számozott részek ezeket az elemeket jelzik, leírásuk számozva, a kép alatt található.



4.1. ábra. Program 'körvonala', alapja, a felület amire a többi menü épül.

1. Gombok helye. Fájl megnyitása és mentése gombra gondoltam. Ha további gombok hozzáadása szükséges lenne, azok is ide kerülnének, feltéve ha mindegyik menüpontban használhatóak.

Fájl megnyitása gomb (4.1 ábrán az 1.ikon): lokális adatbázis használata esetén lehet fontos. Szükségessége attól függ, hogy lokális vagy felhőalapú adatbázis kerül felhasználásra.

Fájl mentése gomb (4.1 ábrán a 2. ikon): elmenti az adatok változásait, amiket a későbbiekben az adatbázisba kell töltsön.

2. Két menüpont váltására alkalmas gombok helye, ezekre kattintva válthatunk a menük között.

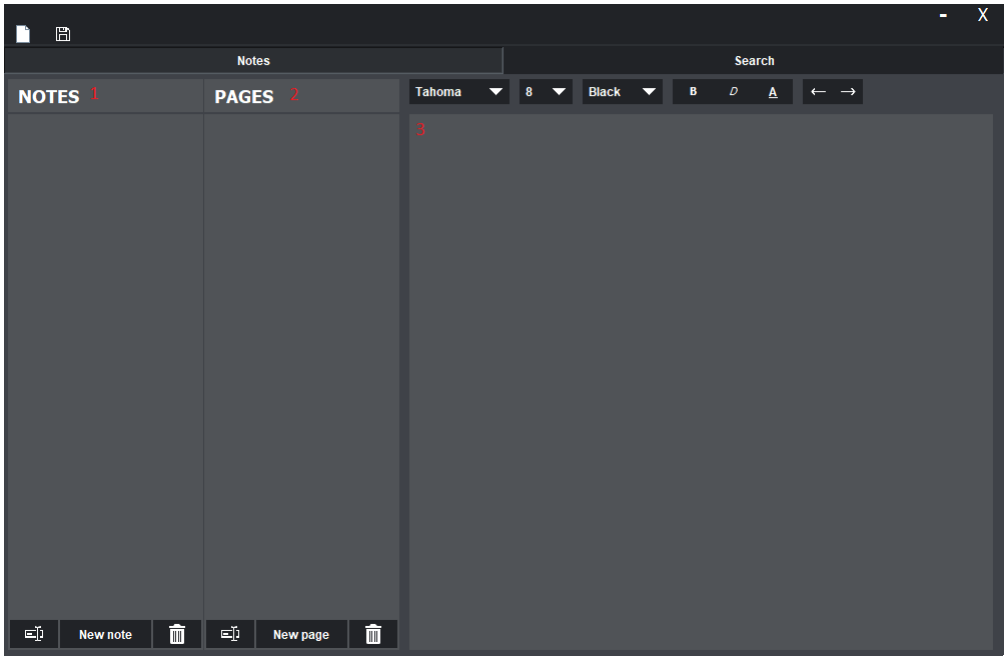
3. Bezár és tálcáz gombok, funkcióik egyértelműeknek tartom.

Az alkalmazás ablak bárhová legyen elhelyezhető a képernyőn (tehát lehessen húzgálni), amíg a felső sötétebb színű sávon tartjuk lenyomva a bal egérgombot, viszont átméretezni ne lehessen.

Belépéskor ez a kép fogadja a felhasználót, egyik menüpontban se legyen benne.

4.1.3. Jegyzetek menü

A 4.2. ábrán látható a jegyzetek menü elképzelt felépítése A pirossal számozott elemek leírása található a kép alatt, elmondja, hogy milyen funkciókat kell majd, hogy az egyes elemek betöltsenek.



4.2. ábra. Jegyzetek menü

1. Jegyzetek (Note objektumok) listája, ahol a jegyzetek neve jelenik meg egymás alatt. Ha betelne a jegyzetek listája, akkor a görgő segítségével lehet köztük navigálni. Átnevezni, létrehozni és törölni jegyzeteket a lista alján található gombokkal lehessen. A gombok csak a kijelölt jegyzetet szerkesszék!

2. Oldalak (Page objektumok) listája. Ha nincs jegyzet kiválasztva, akkor a lista alapvetően üres legyen. Az éppen kijelölt (rákattintott) jegyzet oldalainak nevei legyenek itt felsorolva.
Egy jegyzethez több oldal is létrehozható, de egy adott oldal csak egy meghatározott jegyzethez tartozhat (feltéve ha nem másoltuk be egy másik jegyzetbe ugyan azt a szöveget és adtunk a lapunknak hasonló címet).
Ha betelne az oldalak listája, akkor a görgő segítségével lehessen köztük navigálni.
Átnevezni, létrehozni és törölni oldalakat a lista alján található gombokkal lehetséges.
A gombok csak a kijelölt oldalakat szerkesszék!
Ha nincs kijelölve jegyzet, akkor a lapok listája is legyen üres.

3. Szövegszerkesztő, az éppen kijelölt oldal tartalma legyen itt látható és szerkeszthető. Ha nincs oldal kijelölve, akkor ne legyen használható a szerkesztő és ne jelenjen meg benne szöveg.

Legyenek elérhetők szöveg szerkesztésére alkalmas gombok, amelyek funckiói a következők:

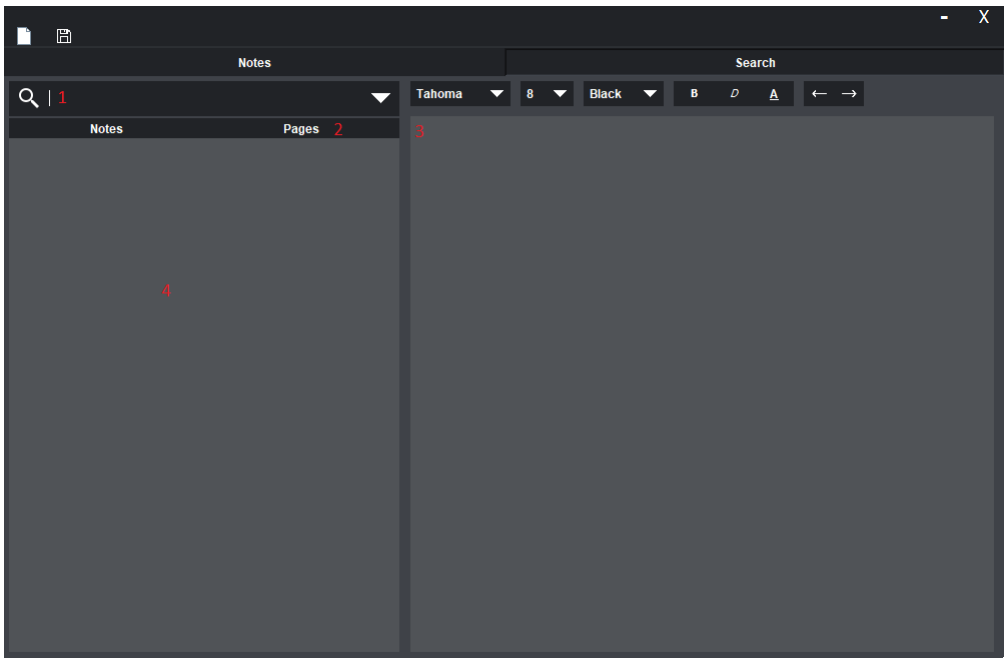
- Betűstílus, betűméret és szín megadására alkalmas legördülő választási menü.

- Szöveg dőltté, vastaggá és aláhúzottá alakító gombok. Ki-bekapcsolóként működjenek.
- Undo gomb, ami az előző szerkesztést vonja vissza.
- Redo gomb, ami az előzőleg visszavont szerkesztést csinálja újra.

Az első és második pontban leírt funkcióknak csak a kijelölt szövegen legyen hatása. A gombok legyenek gyorsbillentyűkkel is használhatók.

4.1.4. Kereső menü

A 4.3. ábrán látható a kereső menü elképzelt felépítése A pirossal számozott elemek leírása található a kép alatt, elmondja, hogy milyen funkciókat kell majd, hogy az egyes elemek betöltsenek.



4.3. ábra. Kereső menü

1. Kereső sáv. Ne legyen case-sensitive, tehát ne tegyen kis- és nagybetű között különbséget.
Jegyzet- és oldalnevek, és az oldalak tartalma között is keressen egyező adatokat.
Jelenjen meg egy legördülő menü ahogy írjuk a szöveget, ami a találatok nevét tartalmazza. Tehát ha egyezést talál a beírt szöveg és

- egy jegyzet neve között, akkor a jegyzet neve jelenjen meg.
- egy lap neve között, akkor a lap neve jelenjen meg.
- egy lap tartalma között, akkor a tartalmazó lap neve jelenjen meg.

Beírt szövegre a nagyító gombra való kattintással tudjunk keresni, vagy a legördülő menüben megjelenő találatokra kattintva.

2. Talált jegyzetek és lapok között a két adott gombbal lehessen váltani. Jelenjen meg

a kiválasztott találatok listája.
Gombra való kattintással lehessen kiválasztani hogy melyik lista látszódjon.

3. A jegyzetek menünél bemutatott szövegszerkesztő, melynek működése legyen megegyező. A kijelölt lap szövege jelenjen meg itt, legyen szerkeszthető.

4. A 2. pontban jellemzett gombok esetén két lehetőség legyen: vagy a jegyzetek listája látszódjon, vagy a lapok listája.
A listák az érvényes találatokat tartalmazzák.
Lehessen a listák elemei között (úgy mint a jegyzetek menüben) kattintással választani. Ilyenkor további két lehetőség legyen:

- Ha egy jegyzet listaelemet választunk, akkor a program tegyen át a jegyzetek menüpontba, és legyen a kiválasztott jegyzet kijelölve. Ha a jegyzetet görgetéssel lehet elérni, akkor a görgő is legyen a megfelelő pozícióba úgy, hogy látszódjon a listán a kiválasztott elem. Lapja ne legyen kijelölve, szövegdoboz legyen üres.
- Ha egy lap listaelemet választunk, akkor a szövegdobozban jelenjen meg a tartalma. Legyen itt is szerkeszthető a tartalom.

4.2. Adatbázis

Az adatok adatbázisba legyenek eltárolva, vagy felhőalapú adatbázisba, vagy lokális adatbázisba.
Ha lokális adatbázisra esik a választás, az adatok szinkronizálása több eszköz között legyen megoldva.
Adatbázishoz való csatlakozás a programkódban legyen megfelelően implementálva.

4.3. Titkosítás

A titkosítási algoritmus tesztek alapján a legjobbnak vélt algoritmussal kerüljenek az adatok kódolásra.
A titkosítás vagy alkalmazás szinten, saját implementációval legyen megoldva vagy a tárolástól függően az ismertetett adatbázis titkosítási módszerek egyikének alkalmazásával.
Mivel az alkalmazás minden esetben az adathalmaz egészével dolgozik, ezért elegendő lehet, ha egy xml vagy json fájl kerül titkosításra, amely az összes adatot tartalmazza, nem pedig minden adat egyesével.
A titkosítási kulcsok legyenek biztonságosan tárolva, több eszköz között biztonságosan megosztva.
Az alkalmazás bezárása előtt kerüljenek át az adatok az adatbázisba. Alkalmazás megnyitása után kerüljenek beolvasásra majd megjelenítésre.

5. fejezet

Implementáció

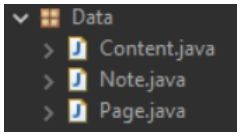
A fejezet a program készítése során felmerülő problémák leírásával foglalkozik. Bemutatja hogyan sikerült az alkalmazás egyes részeit implementálnom. Nem tartalmaz a program használatával kapcsolatos információkat.
A fejezet felépítése a program csomag-szintű felépítésével egyezik.
Nem LaTeX-Java stílusú kódrészleteket tartalmaz, hanem beszúrt képeket a kész kódról.

5.1. Data

Az előző fejezetben leírt szükséges adattípusokat egy Content nevű osztályba gyűjtöttem össze. A csomag felépítése a 5.1. ábrán látható.

Két tároló objektumot tartalmaz, amelyek DefaultListModel típusúak, ezen belül Note és Page típusú adatokat tartalmaznak. DefaultListModel objektumot egy lista elemeinek létrehozására és megjelenítésére használjuk Java-ban. Mivel az adatok csak listákban kerülnek megjelenítésre, ezért ez az objektum tökéletes volt az adatok tárolására.

Az alkalmazás azon részei, amelyek az adatokkal dolgoznak a Content osztályon keresztül érik el őket.



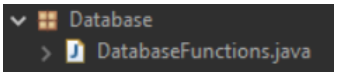
5.1. ábra. Data csomag és osztályai

5.2. Database

Egy osztályt tartalmaz, a DatabaseFunctions-t, amiben az adatbázissal kapcsolatos funkciókat gyűjtöttem össze, mint az adatbázishoz való csatlakozás, fájlok fel- és letöltése. A csomag felépítése a 5.2. ábrán látható.

Az adatok Firebase nevű felhőalapú adatbázisban kerülnek tárolásra, így az alkalmazás megfelelő működéséhez szükséges van internetkapcsolatra.
Az alkalmazást át kellett alakítanom Maven project-re, hogy a Firebase adatbázis

driver-éhez hozzá tudjak férni, majd az API segítségével kapcsolatba tudjak lépni az adatbázissal.
Lehetségünk van Firebase-en belül Blob (Binary large object/Nagy bináris objektum) fájlok tárolására. Az alkalmazás adatai titkosított XML fájl formájában kerülnek az adatbázisba. A Firebase ezen felül data-at-rest titkosítást is alkalmaz.



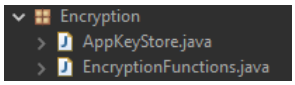
5.2. ábra. Database csomag és osztálya

5.3. Encryption

Titkosítással kapcsolatos osztályokat tartalmaz. A csomag felépítése a 5.3. ábrán látható.

Az AppKeyStore osztálynak két funkciója van.
Egyik egy random generált 256 bit-es AES titkosítási kulcs létrehozására szolgál, amit egy KeyStore objektumban tárol, ezt .jks kiterjesztésű 256 karakter hosszú jelszóval védett fájlként tárolja. A fájl a progarm működése alatt feltöltődik az adatbázisba.
A jelszó karakterek random sorozatából áll, de minden .jks fájl ugyanazzal a jelszóval van védve. Vitatható, hogy ez a megoldás biztonságos-e, úgy gondolom, hogy igen, a jelszó hosszából és karaktereiből ítélve, ugyanis nem csak számokat és betűket tartalmaz, hanem szimbólumokat is.
Adatbázisba töltését szükségszerűnek tartom, hogy különböző eszközökön is el tudjuk érni ugyanazt a tartalmat.
Másik metódusa segítségével letölti az adatbázisból a .jks fájlt, amit ideiglenesen eltárol az eszközön, majd ebből az ideiglenes fájlból éri el a korábban létrehozott titkosítási kulcsot.

Az EncryptionFunctions osztály a titkosítást és visszafejtést végző funkciókat tartalmazza. Ezek a műveletek a .jks fájlba elmentett random generált kulcsot alkalmazzák az XML fájl titkosításához és visszafejtéséhez.
A funkciók AES algoritmust használnak, működésük megegyezik az algoritmusok összehasonlításához használt tesztalkalmazás működésével, a javax.crypto csomag osztályait alkalmaztam.

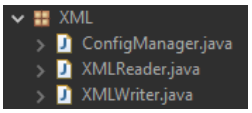


5.3. ábra. Encryption csomag és osztályai

5.4. XML

A csomag XML-lel foglalkozó osztályokat tartalmaz, felépítése a 5.4. ábrán látható. Ilyen az XMLReader, ami olvassa és 'parse'-olja az adott XML fájlt. Az XMLWriter XML dokumentum írását és formázását végzi.

A program a szükséges fájlokat egy config.xml-nek nevezett dokumentumból éri el, aminek írását és olvasását a ConfigManager osztály végzi. Ennek az osztálynak a részletes működését a következő fejezet tartalmazza.

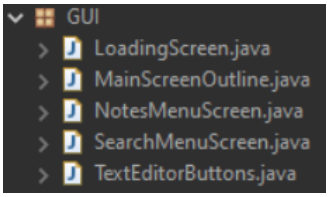


5.4. ábra. XML csomag és osztályai

5.5. GUI

Ez a csomag a felhasználói felület főbb osztályait tartalmazza (lásd 5.5. ábra).

Legnehezebb dolgom a TextEditorButtons nevű osztállyal volt, ami a 4.1.3-mas alfejezetben leírt szövegszerkesztő megfelelő működését biztosítja. A felhasználói felület készítése során ennek az osztálynak létrehozása volt a legidőigényesebb feladat.
Alapértelmezett beállítás a 12-es nagyság, fehér szín és Dialog betűstílus. A szövegszerkesztőbe beírt adatok és a szövegszerkesztő gombjai között nincs valós-idejű kapcsolat. Ezt egy példával tudom legjobban elmagyarázni: ha a szöveg egy részének (a gombok ugye csak a kijelölt részeket szerkesztik) stílusát megváltoztatjuk, tegyük fel 18-as betűméret, piros szín és Times New Roman betűstílusra, a gombokon feltüntetett szöveg megváltozik. Ha a kurzorral egy nem szerkesztett szövegrészre kattintunk, akkor a gombok állapota nem áll vissza az alapértelmezett beállításokra, hanem az utoljára beállított opción marad (tehát nem lesz Dialog-12-fehér, hanem Times New Roman-18-piros marad).

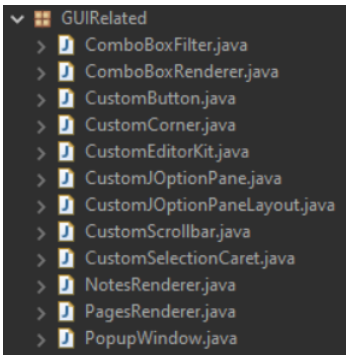


5.5. ábra. GUI csomag és osztályai

5.6. GUIRelated

A csomag olyan osztályokat tartalmaz, amelyek a grafikus felhasználói felület személyre szabásában segítettek. A csomag felépítése a 5.6. ábrán látható.
A felhasználói felületet Java Swing használatával készítettem. Ahhoz, hogy megértsük milyen nehézségekkel szembesültem, egy rövid leírásban be kell mutassam a függvénykönyvtárat:

A Swing egy GUI widget toolkit. Widget-ek alatt olyan grafikus elemeket értünk, mint gombok, szövegdobozok, listák, legördülő menük, görgők, stb...
A Swing egy régi technológia, viszont rengeteg alkalmazás használja. A Java GUI készítésre létrehoztak egy modernebb függvénykönyvtárat, ami a Swing-re épül, a JavaFX-et.



5.6. ábra. GUIRelated csomag és osztályai

Kevésbé használt, mint a Swing, valószínűleg azért, mivel az alkalmazások nagy része korábban, Swing-ben készült, és nincs szükség JavaFX-szé való konvertálásukra. Új Java GUI-k készítésére valószínűnek tartom, hogy inkább JavaFX-et használnak.

Azért választottam a Swing-et a JavaFX helyett, mert már dolgoztam a könyvtárral, míg a JavaFX-et egyáltalán nem ismertem, így azt gondoltam, hogy egyszerűbb dolgom lesz.
Hamar rá kellett jönnöm, hogy nem lesz olyan egyszerű dolgom, mint amire számítottam, de ettől függetlenül maradtam a Swing használata mellett. Ahhoz, hogy a widget-eknek egy modern megjelenést tudjunk adni, rengeteg szülőosztály-beli metódust kell megvizsgálnunk és felülírnunk. Sok esetben nem érhető el olyan funkció, amit logikusnak tartanánk, hogy létezik és közvetlenül szerkeszthető. Pár példa, hogy mire is gondolok:

A JButton widget egy gomb, amire lehet szöveget írni. Ha megnyomjuk a gombot, megváltozik a háttérszíne, amíg el nem engedjük az egérgombot. Ezt a színt közvetlenül egy metódussal nem lehet megváltoztatni. A felülírt kódról egy részlet a 5.7. ábrán látható

```
1  package GUIRelated;
2
3  import javax.swing.JButton;
7  public class CustomButton extends JButton{
8
9      /**
10     * Randomly generated serial version ID
11     */
12     private static final long serialVersionUID = 6531371007240993732L;
13     private Color hoverBackgroundColor;
14     private Color pressedBackgroundColor;
15
16     public CustomButton() {
17         this(null);
18     }
19
20     public CustomButton(String text) {
21         super(text);
22         super.setContentAreaFilled(false);
23     }
24
25     @Override
26     protected void paintComponent(Graphics g) {
27         if (getModel().isPressed()) {
28             g.setColor(pressedBackgroundColor);
29         } else if (getModel().isRollover()) {
30             g.setColor(hoverBackgroundColor);
31         } else {
32             g.setColor(hoverBackgroundColor);
33         }
34         g.fillRect(0, 0, getWidth(), getHeight());
35         super.paintComponent(g);
36     }
37
38     @Override
39     public void setContentAreaFilled(boolean b) {
40     }
41
42     public Color getHoverBackgroundColor() {
43         return hoverBackgroundColor;
44     }
45
```

5.7. ábra. JButton megjelenését felülíró osztály

A JList listaelemek háttérszínét, legyen-e az elemeknek kerete vagy sem, elemek szövegének színét sem lehet közvetlen metódussal megváltoztatni. A felülírt kódról egy részlet a 5.8. ábrán látható

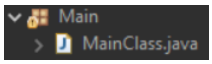
```java
package GUIRelated;

import java.awt.Component;

public class PagesRenderer extends JPanel implements ListCellRenderer<Page>{

    /**
     * Randomly generated serial version ID
     */
    private static final long serialVersionUID = 1645790349661751579L;

    private JLabel pagesNameLabel = new JLabel();

    private Color seperatorColor = new Color(63, 66, 72);

    public PagesRenderer() {
        setLayout(new BorderLayout(0, 0));

        JPanel panelText = new JPanel(new GridLayout(0, 1));
        panelText.add(pagesNameLabel);
        add(new JLabel("  "), BorderLayout.WEST);
        add(panelText, BorderLayout.CENTER);
    }

    @Override
    public Component getListCellRendererComponent(JList<? extends Page> list, Page page,
        int index, boolean isSelected, boolean cellHasFocus) {

            pagesNameLabel.setText(page.getPageName());
            pagesNameLabel.setOpaque(true);
            pagesNameLabel.setForeground(Color.WHITE);


            if(isSelected) {
                pagesNameLabel.setBackground(list.getSelectionBackground());
                setBorder(BorderFactory.createLineBorder(Color.WHITE));
                setBackground(list.getSelectionBackground());

            } else {
                pagesNameLabel.setBackground(list.getBackground());
                setBorder(BorderFactory.createMatteBorder(0, 0, 1, 0,  seperatorColor));
```

5.8. ábra. JList listaelemeit renderelő osztály

A csomag összes osztálya ilyen jellegű problémákat old meg, szükségesek az elképzelt, modern felhasználói felület létrehozásához.

5.7. Main

A csomag a fő futtatható osztályt tartalmazza. Internetkapcsolat ellenőrző függvényt és az alkalmazás adatait betöltő algoritmusokat tartalmazza. A csomag felépítése a 5.9. ábrán látható.



5.9. ábra. Main csomag és osztálya

5.8. További fejlesztési lehetőségek

Egyik opció sem kulcsfontosságú az alkalmazás működésének szempontjából, azért nem kerültek bele a program végső verziójába. Hasznos funkciók, de némelyik inkább csak a kényelmet szolgálja.
Az ötletek nagy részének későbbi megvalósítása ettől függetlenül tervbe van.

5.8.1. Automatikus elindulás

A program automatikusan induljon el az eszköz indításakor.

5.8.2. Naptár és emlékeztető

Az első megvalósítandó ötlet egy naptár elhelyezése az alkalmazás kettő meglévő menüje mellé.
Ez a képernyő jelenne meg alkalmazás indulásakor, főmenü szerepet töltene be.

A felületen lenne egy naptár, amit lehetne időben előre-hátra lapozni, akárcsak egy

rendes naptárat.

Emlékeztető írására alkalmas funkciókat is kéne tartalmazzon, ez lenne a menüpont lényege. A felhasználó írhatna egy emlékeztetőt egy jövőbeli eseményre (múltbeli eseményre hibát dobna) és be lehetne állítani, hogy mikor emlékeztessen (aznap, egy nappal előtte, két nappal előtte, stb..).

Emlékeztető alatt egy felugró ablakra gondolok, ami attól függetlenül, hogy milyen ablakok vannak még megnyitva (legyen az böngésző, vagy másik alkalmazás), azok felett lenne (on top).
Felépítése hasonló lenne a jegyzet/lap törlése modális ablakéhoz. Tartalmazna egy 'OK' gombot, ami bezárja ezt az ablakot, és egy 'Remind me later' gombot, ami ideiglenesen bezárja az ablakot, majd X perc múlva ismét megjeleníti. A gombok felett jelenne meg az emlékeztető szövege.

5.8.3. Beállítások

Egy felugró ablak, tele opciókkal.
Kimondottan a felhasználói felület megváltoztatására alkalmas beállításokra gondolok (pl.: színek variálása), de akár működés módosítására alkalmas beállításokat is tartalmazhatna ez az ablak.
Két titkosítási opciót is el lehetne itt helyezni. Ezalatt arra gondolok, hogy alkalmazás szintű titkosítási megoldást alkalmazzon az applikáció vagy elegendőnek tartja a felhasználó az adatbázis at-rest titkosítását, amit a szolgáltató végez.

6. fejezet

Működés, tesztelés

A fejezet az elkészített alkalmazás rendeltetésszerű működését mutatja be.

6.1. Config fájl

Az alkalmazáshoz szükséges fájlok elérési útvonalát a config.xml fájlban kell megadni. Egy fájl elérési útvonalát az alábbi módon kell megadni:
C:\Users\AMD\Desktop\firebase_service_key_example.json.

Ha az alkalmazást először futattjuk az eszközön, vagy egy új mappában tettük a futtatható .jar fájlt, amiben nem található config.xml, akkor automatikusan létrejön az adott mappában egy fájl, példa adatokkal. Ezeket meg kell változtatnunk, hogy használni tudjuk az alkalmazást.

A 6.1. ábra egy automatikusan létrehozott config.xml fájl felépítését mutatja. A pirossal számozott sorok tartalmát kell szerkesszük, a további alfejezetekben ezt fejtem ki.

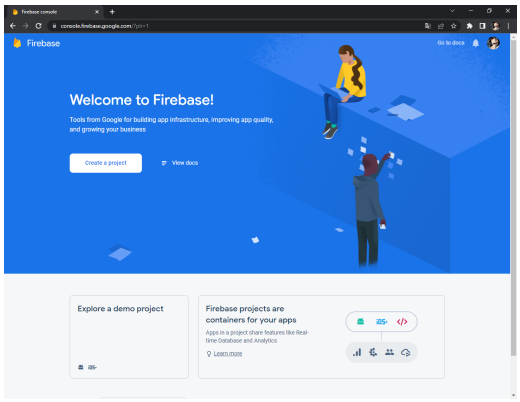
```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<config>
    <editable>
      1 <dbJsonPath>C:\Users\Example\Downloads\firebase_service_key_path.json</dbJsonPath>
      2 <dbStoragePath>firebase_example-15f47.appspot.com</dbStoragePath>
    </editable>
</config>
```

6.1. ábra. Config.xml tartalma első futtatás után

6.1.1. Firebase

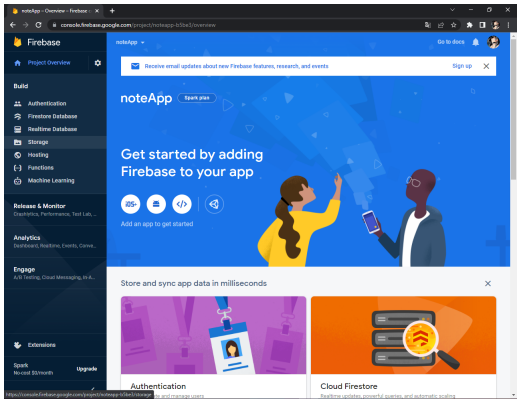
A 6.1. ábrán piros 1 és 2-essel jelölt sorok megfelelő kitöltéséhez létre kell hoznunk egy Firebase adatbázist.

Egy böngészőben a console.firebase.google.com url beírása, majd egy google fiókba való belépés után ez a 6.2. ábrán látható kép fogad minket.



6.2. ábra. Firebase regisztráció

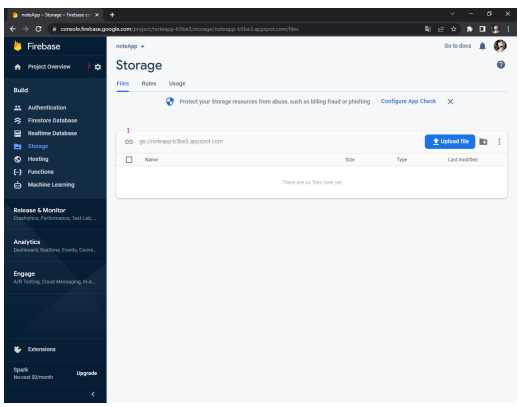
Kattintsunk a 'Create a project' gombra, majd végezzük el a szükséges lépéseket. Ha megtettük, a 6.3. ábra képét láthatjuk. Kattintsunk a 'Storage' gombra a 'Build' menün belül, a kép bal szélén található sávban.



6.3. ábra. Firebase kezdő képernyő

Kattintsunk a 'Get started' gombra, majd válasszuk ki a 'Start in production mode' lehetőséget, majd 'Next'. A legördülő menüben válasszuk a megfelelő Storage location-t, ami valószínűleg eur3 (europe-west) lesz. Fontos, hogy ezt nem lehet később megváltoztatni, így győződjünk meg választásunkról! Végül a 'Done' gombra nyomjunk.

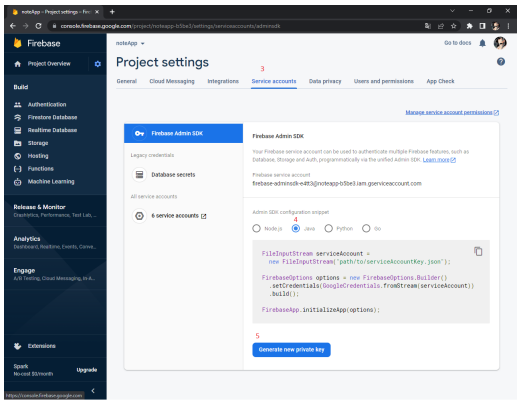
Ha ezt mind megtettük a 6.4. ábrához hasonlót látunk.



6.4. ábra. Firebase storage alapértelmezett kép

A config.xml fájlban a tag-ek (<dbJsonPath>...</dbJsonPath>) közé kell a megfelelő adatokat beírnunk. A 6.1. ábrán piros 2-essel jelölt <dbStoragePath>...</dbStoragePath> tag-ek közé a 6.4. ábrán 1-essel jelölt linket kell bemásoljuk a 'gs://' nélkül.

Ha ez megvan kattintsunk a 6.4. ábrán piros 2-essel jelölt fogaskerékre, majd 'Project settings', az újonnan megjelent felső sávban pedig a service accounts-ra (lásd 6.5. ábra 3-mas lehetőség).



6.5. ábra. Firebase service key generálás

Nyomjuk meg sorrendben a 6.5. ábrán piros számokkal jelölt gombokat, majd a felugró ablaknál a 'Generate key' gombot. Sikeresen letöltöttük a szükséges Firebase Service Key json fájlt!

A config.xml fájlban a <dbJsonPath>...</dbJsonPath> tag-ek közé másoljuk be a letöltött json fájl elérési útvonalát. Ezt a leírás elején található példa elérési úthoz hasonlóan tegyük.

Ha lépésről lépésre mindent megtettünk, a config.xml fájl tartalma hasonlóan kell, hogy kinézzen, mint a 6.6. ábrán látható.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<config>
    <editable>
        <dbJsonPath>C:\Users\AMD\Desktop\noteapp-b5be3-firebase-adminsdk-e4tt3-bb0109c371.json</dbJsonPath>
        <dbStoragePath>noteapp-b5be3.appspot.com</dbStoragePath>
    </editable>
</config>
```

6.6. ábra. Korrekt config.xml felépítés

Végezetül mentsük el, majd zárjuk be a config.xml fájlt, ezután futtassuk a .jar fájlt. Ha hibaüzenetet kapunk vizsgáljuk meg a config.xml fájlt. Nézzük meg, hogy helyesen írtuk-e be az elérési útvonalakat, '\' jeleket alkalmaztunk-e a megadásukhoz.
Ha továbbra is hibaüzenetet kapunk ismételjük el elejéről a lépéseket.

Ha mindent megfelelően adtunk meg a program elindul.

6.1.2. Fontos információk

Nagyon fontos, hogy a .jks fájlt ne töröljük. Ha mégis letöröljük hibákba fogunk ütközni, az alkalmazás adatai elvesznek és nem tudjuk őket sehogy visszaszerezni.

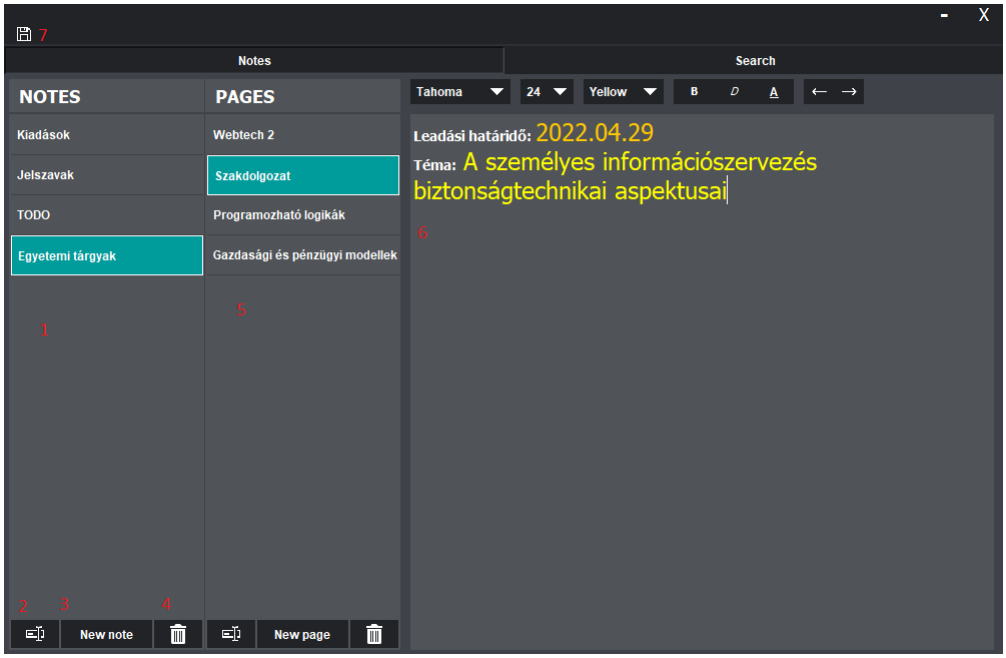
Ha töröltük a KeyStore fájlt, akkor 6.4. ábrán lévő felületen töröljük a Storage-ben

lévő, programhoz tartozó összes fájlt (appKeyStore.jks, applicationData.xml). Így az előzőleg elmentett titkosított adatok is eltűnnek, és a .jks fájl is. A program ebben az esetben következő futtatáskor új fájlokat generál és üres tartalommal indul.

A program futtatásához java futtató környezet szükséges, ezt JRE-nek hívják (Java Runtime Environment). Telepíthetjük az Oracle-féle implementációt is, ami a https://java.com/en/download/manual.jsp linken érhető el.

6.2. Jegyzetek menü

Az 6.7. ábrán látható a jegyzetek menü felépítése, a pirossal számozott elemek leírása található a kép alatt.



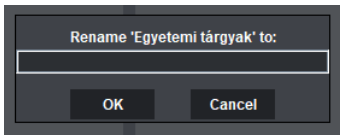
6.7. ábra. Jegyzetek menü használat közben.

1-es panel a jegyzetek panelje, itt találhatóak egymás alá beszúrva a különböző jegyzetek. Az éppen kijelölt jegyzet kék színű, kattintással tudunk új jegyzetet kiválasztani, valamint ha a panelen belül az üres részre kattintunk, akkor mindig az utolsó jegyzetet fogja kijelölni.

Található a panelen 3 gomb.

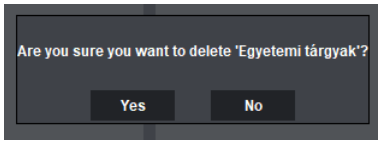
2-es gomb az éppen kijelölt jegyzet átnevezésére szolgál, egy felugró modális ablakba a beírt szövegre nevezhetjük át a kijelölt elemet, vagy megszakíthatjuk az egész folyamatot. (lásd 6.8. ábra)

3-mas gombbal új jegyzetet adunk a meglévő jegyzet listához. Alapértelmezett neve a jegyzeteknek 'New Note [sorszám]'. A sorszám a jegyzet id-jével egyezik, nem a listabeli pozíciójával.



6.8. ábra. Átnevezésre szolgáló modális ablak.

4-es gombbal az éppen kijelölt jegyzetet törölhetjük, vagy szakíthatjuk meg ezt a folyamatot egy felugró modális ablak segítségével. (lásd 6.9. ábra)



6.9. ábra. Törlésre szolgáló modális ablak.

5-ös panel a lapok panelje. Minden jegyzetnek külön lapjai lehetnek. A panel egyéb funkciói és kezelése teljesen megegyezik az előbb bemutatott jegyzetek menüével.

Ahogy kattintással váltogatunk a jegyzetek között, úgy frissül a lapok menü is, tehát ahogy a beszúrt képen látszik, az 'Egyetemi tárgyak' jegyzetnek 4 lapja van. Ha átkattintanánk a 'TODO'-ra, akkor ebben a jegyzetben található lapok listája jelenne meg. Alapvetően jegyzet váltáskor nem jelölődik ki egy lap sem.

6-os szövegdoboz felületen az éppen kijelölt lap tartalma jelenik meg. A felület mindig frissül, ha új lapra kattintunk, valamint ha nincs egy lap sem kijelölve, akkor üres lesz. A szövegdoboz felett gombok találhatók, amik a szöveg szerkesztésére használhatóak. Balról jobbra haladva a gombok:

-Szöveg stílus váltás: alapértelmezett Tahoma. Csak a kijelölt szövegrész stílusa fog megváltozni!

-Szöveg méret váltás: alapértelmezett 14-as méret. Csak a kijelölt szövegrész mérete fog megváltozni!

-Szöveg szín váltás: alapértelmezett fehér szín. Csak a kijelölt szövegrész színe fog megváltozni!

-Félkövér betű: kijelölt szövegrész félkövérré alakítása.
Gyorsbillentyű: Ctrl + B

-Dőlt betű: kijelölt szövegrész dőltté változtatása.
Gyorsbillentyű: Ctrl + I

-Aláhúzás: kijelölt szövegrész aláhúzása.
Gyorsbillentyű: Ctrl + U

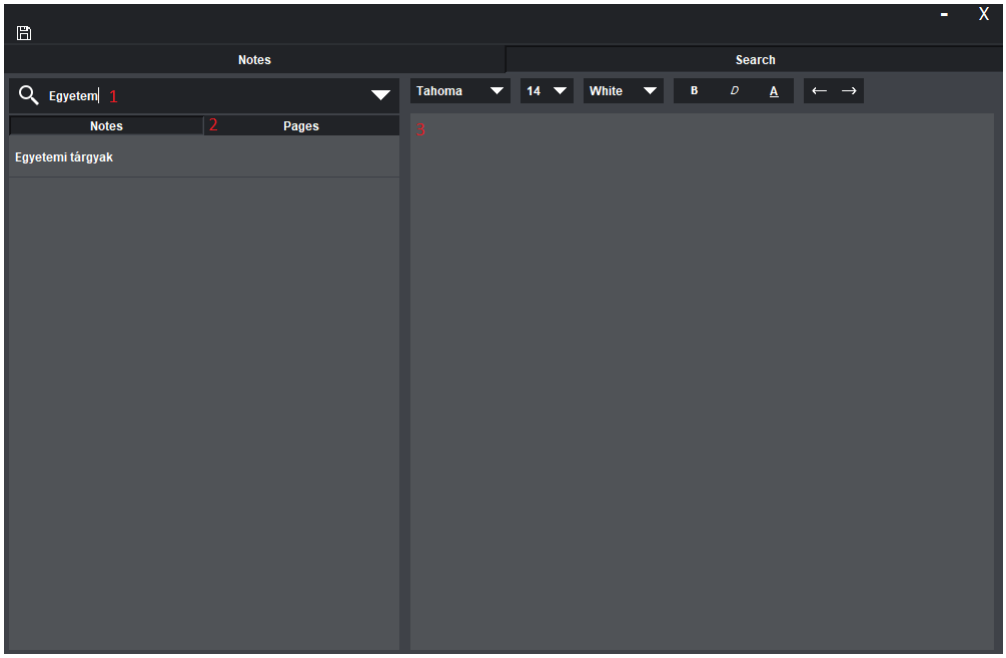
-Undo gomb: A szövegdoboz utolsó változtatását visszavonja. Fontos, hogy csak a szövegdobozon működik, note vagy page törlésre, átnevezésre, stb... nem!
Gyorsbillentyű: Ctrl + Z

-Redo gomb: Az undo gomb ellentéte. Fontos, hogy csak a szövegdobozon működik, note vagy page törlésre, átnevezésre, stb... nem!
Gyorsbillentyű: Ctrl + Y

7-essel jelzett gomb a mentés gomb. Gyorsbillentyű: Ctrl + S. Az alkalmazás tartalmát titkosítja majd elmenti az adatbázisba.
A gomb megnyomásakor vagy gyorsbillentyű használatakor az alkalmazás kis időre megfagyhat.

6.3. Kereső menü

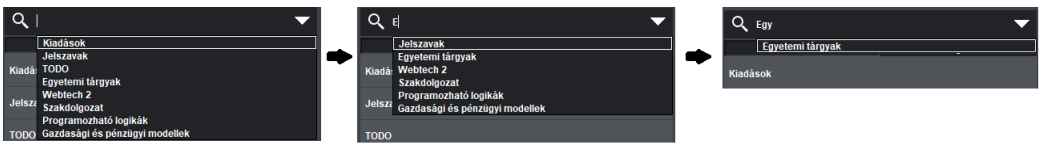
Az 6.10. ábrán látható a kereső menü felépítése, a pirossal számozott elemek leírása található a kép alatt.



6.10. ábra. Kereső menü működés közben.

1-essel jelölt elem a 'search bar' (kereső sor). Jegyzetek, lapok és lapok tartalma között keres. Ha jegyzet neve, lap neve, vagy lap tartalma tartalmazza a beírt kifejezést, akkor lesz találat.

Hogy a keresés könnyebb legyen, üres search bar-ba való kattintáskor az összes jegyzet és lap neve megjelenik egy listába. Ahogy a keresendő szöveget írjuk be a lista annak megfelelően fog változni (lásd 6.11. ábra).



6.11. ábra. Keresési folyamat szemléltetése.

Ha nincs egyezés, a lista eltűnik, nem jelenít meg egyetlen elemet sem.

Ha az egyező szöveg egy adott lap tartalmán belül található, tehát nem a nevében,

akkor is a listában a lap neve fog szerepelni.

A beírt szövegre keresni kijelölés után a nagyítóra való kattintással vagy enter megnyomásával lehetséges.

2-essel jelölt elem a Note és Page keresési találatok listája.

Alapvetően, ha futtatáskor először lépünk be ebbe a menübe egyik sem lesz kijelölve, viszont az első keresés után a jegyzetek listáját fogja megjeleníteni. Hogyha a page-re kattintunk, akkor a talált lapokat fogja megjeleníteni, így tudunk váltani a talált jegyzetek és lapok, illetve lapok tartalma között.

Ha a jegyzet találatok között az egyikre rákattintunk, akkor visszakerülünk a Notes Menu-be, hogy meg tudjuk tekinteni az adott jegyzet lapjait, átnevezni vagy törölni tudjunk.

Ha a lap találatok között az egyikre kattintunk, akkor nem kerülünk vissza a Notes Menu-be. Az adott lap tartalmát tudjuk szerkeszteni és megtekinteni egy Notes Menu-höz hasonló szövegdobozban **(3.elem)**. Ennek funkciói teljesen megegyeznek az említett szövegdobozéval és menteni is a ctrl+s-el kell a változtatásokat, vagy a mentés ikonra kattintva.

Ha esetleg átnevezni vagy törölni szeretnénk az adott lapot, a Notes Menu-ben tehetjük ezt meg.

6.4. Ismert bug-ok

Jelenleg kettő kisebb bug-ról tudok, ami biztosan megtalálható az alkalmazásban. Egyik bug sem 'töri meg' az alkalmazást, de a gördülékeny működéshez jó lenne a közeljövőben megoldani őket.
Mind a felhasználói felülethez köthető, azon belül is a kereső menühöz.

- Bármit írunk be a keresőbe, ha több mint egy találatot dob ki, és rányomunk az első lehetőségre, akkor az a keresősáv szövegdobozába mindig a 'New Note 0'-t fogja behelyettesíteni (a jegyzetek listájának első elemének nevét). Ez egyetlen egyező találat esetén nem érvényes.

- A másik keresősáv bug pedig ehhez, egyetlen egyezés esetéhez köthető. Tegyük fel rákeresünk a 'New Note 0'-ra, akkor a legördülő menü eltűnik, attól függetlenül, hogy van egyező találat. A szövegdobozra való újabb kattintással jelenik meg. Ha viszont nem teljes egyezőséggel keresünk rá valamire, például azt írjuk be, hogy 'New note 0', akkor megjelenik az egyező találat egyetlen egyezés esetén is.

Szóba kerültek az 5.5. és 5.6. alfejezetben a felhasználói felület létrehozásával kapcsolatos nehézségek, a keresősáv bug-jait is ilyen nehézségek közé sorolnám.
Úgy gondolom, ha nem Swing-et, hanem egy modernebb GUI fejlesztő függvény könyvtárat használtam volna, akkor nem találkoztam volna ezekkel a hibákkal.

7. fejezet

Összefoglalás

Úgy gondolom, hogy sikerült a dolgozat elméleti részének egész jól lefednie a témát és az adott problémákat.

Biztos vagyok benne, hogy sikerült kiválasztanom a megfelelő titkosítási algoritmust és módszert mind a tesztek és az alkalmazás szempontjából is, hiszen az AES egy nemzetközileg elismert és tesztelt, erős és biztonságos titkosítási algoritmus.

Az alkalmazás végső verziójával teljes mértékben meg vagyok elégedve. Sikerült kialakítanom szinte minden téren úgy az applikációt, ahogy azt témaválasztáskor elterveztem.

Úgy érzem, a szövegszerkesztőt még lehetne tovább fejleszteni és tökéletesíteni, szinte majdnem az elképzeléseknek megfelelően működik, de úgy gondolom több időt kellett volna a fejlesztésére szánnom.
Ugyanakkor észben kell tartani, hogy az alkalmazás elsődleges funkciója nem a szöveg szerkeszthetősége volt, hanem adatok jegyzet-szerű, struktúrált és biztonságos tárolása, valamint az elérhetőség, rendelkezésre állás, amiket szerintem sikerült jól kiviteleznem.

Összefoglalva meg vagyok elégedve a munkámmal és a dolgozattal, sok hasznos dolgot tanultam adatbiztonságtól kezdve idegen kód olvasásáig, amik a jövőben biztosan hasznomra válnak.

Ahogy az 5. fejezet végén írtam, az alkalmazást használni és fejlesztését folytatni fogom a jövőben.

A. függelék

Táblázatok

A.1. táblázat. 1MB-hoz tartozó titkosítási mérések

Titkosítás 1MB	**AES**	**DES**	**Triple DES**	**Blowfish**
1.mérés	55	70	105	60
2.mérés	51	63	98	65
3.mérés	57	68	109	57
4.mérés	58	67	102	58
5.mérés	55	68	110	62
6.mérés	54	71	106	58
7.mérés	53	63	102	59
8.mérés	57	65	105	64
9.mérés	51	70	107	53
10.mérés	51	72	100	56
Átlag	**54,2**	**67,7**	**104,4**	**59,2**

A.2. táblázat. 1MB-hoz tartozó visszafejtési mérések

Visszafejtés 1MB	**AES**	**DES**	**Triple DES**	**Blowfish**
1.mérés	13	29	59	18
2.mérés	12	30	58	21
3.mérés	11	30	68	18
4.mérés	12	31	58	19
5.mérés	12	30	59	21
6.mérés	12	31	58	18
7.mérés	12	32	59	20
8.mérés	11	29	59	20
9.mérés	12	27	59	18
10.mérés	13	30	56	19
Átlag	**12**	**29,9**	**59,3**	**19,2**

A.3. táblázat. 2MB-hoz tartozó titkosítási mérések

Titkosítás 2MB	**AES**	**DES**	**Triple DES**	**Blowfish**
1.mérés	52	86	149	71
2.mérés	58	85	148	68
3.mérés	51	90	157	65
4.mérés	56	86	149	66
5.mérés	51	85	145	71
6.mérés	60	97	161	76
7.mérés	55	86	166	69
8.mérés	60	95	153	68
9.mérés	70	89	157	70
10.mérés	54	87	154	74
Átlag	**56,7**	**88,6**	**153,9**	**69,8**

A.4. táblázat. 2MB-hoz tartozó visszafejtési mérések

Visszafejtés 2MB	**AES**	**DES**	**Triple DES**	**Blowfish**
1.mérés	18	48	109	32
2.mérés	17	47	110	33
3.mérés	17	48	111	32
4.mérés	17	48	110	32
5.mérés	17	48	110	32
6.mérés	20	49	136	33
7.mérés	18	49	137	35
8.mérés	20	50	111	35
9.mérés	18	49	111	32
10.mérés	18	50	112	34
Átlag	**18**	**48,5**	**115,7**	**33**

A.5. táblázat. 3MB-hoz tartozó titkosítási mérések

Titkosítás 3MB	**AES**	**DES**	**Triple DES**	**Blowfish**
1.mérés	55	109	205	96
2.mérés	64	108	216	80
3.mérés	63	114	241	80
4.mérés	56	98	209	78
5.mérés	60	113	204	84
6.mérés	73	117	209	79
7.mérés	59	103	238	78
8.mérés	64	117	206	81
9.mérés	58	113	242	86
10.mérés	59	113	204	85
Átlag	**61,1**	**110,5**	**217,4**	**82,7**

A.6. táblázat. 3MB-hoz tartozó visszafejtési mérések

Visszafejtés 3MB	**AES**	**DES**	**Triple DES**	**Blowfish**
1.mérés	20	68	162	41
2.mérés	22	67	163	41
3.mérés	21	67	199	42
4.mérés	21	68	163	42
5.mérés	21	69	162	41
6.mérés	23	67	164	45
7.mérés	21	68	201	42
8.mérés	20	67	162	43
9.mérés	22	70	200	42
10.mérés	21	68	164	45
Átlag	**21,2**	**67,9**	**174**	**42,4**

A.7. táblázat. 4MB-hoz tartozó titkosítási mérések

Titkosítás 4MB	**AES**	**DES**	**Triple DES**	**Blowfish**
1.mérés	66	135	258	93
2.mérés	63	115	254	95
3.mérés	60	137	258	96
4.mérés	56	135	255	93
5.mérés	58	131	303	94
6.mérés	72	137	262	91
7.mérés	62	133	263	91
8.mérés	66	137	310	95
9.mérés	67	123	259	97
10.mérés	62	137	270	92
Átlag	**63,2**	**132**	**269,2**	**93,7**

A.8. táblázat. 4MB-hoz tartozó visszafejtési mérések

Visszafejtés 4MB	**AES**	**DES**	**Triple DES**	**Blowfish**
1.mérés	23	84	212	51
2.mérés	23	85	213	53
3.mérés	22	83	213	52
4.mérés	23	83	213	50
5.mérés	22	83	262	40
6.mérés	24	91	217	54
7.mérés	24	84	216	52
8.mérés	23	86	265	52
9.mérés	23	88	214	51
10.mérés	23	83	266	52
Átlag	**23**	**85**	**229,1**	**50,7**

Irodalomjegyzék

[1] Wikipedia contributors. Personal information management — Wikipedia, the free encyclopedia, 2022. [Online; accessed 30-April-2022].

[2] Nurzhan Nurseitov, Michael Paulson, Randall Reynolds, and Clemente Izurieta. Comparison of json and xml data interchange formats: a case study. *Caine*, 9:157–162, 2009.

[3] Wikipedia contributors. Database — Wikipedia, the free encyclopedia, 2022.

[4] Deka Ganesh Chandra, Ravi Prakash, and Swati Lamdharia. A study on cloud database. In *2012 Fourth International Conference on Computational Intelligence and Communication Networks*, pages 513–519. IEEE, 2012.

[5] Antonio Varriale, Paolo Prinetto, Alberto Carelli, and Pascal Trotta. Secube (tm): Data at rest and data in motion protection. In *Proceedings of the International Conference on Security and Management (SAM)*, page 138. The Steering Committee of The World Congress in Computer Science, Computer . . . , 2016.

[6] Luc Bouganim and Yanli Guo. Database encryption, 2009.

[7] Wikipedia contributors. Filesystem-level encryption — Wikipedia, the free encyclopedia, 2021. [Online; accessed 29-April-2022].

[8] Wikipedia contributors. Disk encryption — Wikipedia, the free encyclopedia, 2022. [Online; accessed 29-April-2022].

[9] Amitabh Saxena, Vikrant Kaulgud, and Vibhu Sharma. Application layer encryption for cloud. In *2015 Asia-Pacific Software Engineering Conference (APSEC)*, pages 377–384. IEEE, 2015.

[10] Kazuo Nakatani, Ta-Tao Chuang, and Duanning Zhou. Data synchronization technology: standards, business values and implications. *Communications of the Association for Information Systems*, 17(1):44, 2006.

[11] Balaji N. One-way vs two-way syncing: What's the difference?, Feb 2020.

[12] Sandro Rafaeli and David Hutchison. A survey of key management for secure group communication. *ACM Computing Surveys (CSUR)*, 35(3):309–329, 2003.

[13] Jay Thakkar. Types of encryption: 5 encryption algorithms & how to choose the right one, Mar 2022.

[14] Diaa Salama Abd Elminaam, Hatem Mohamed Abdual-Kader, and Mohiy Mohamed Hadhoud. Evaluating the performance of symmetric encryption algorithms. *Int. J. Netw. Secur.*, 10(3):216–222, 2010.

[15] Symmetric vs. asymmetric encryption - what are differences?, Jun 2021.

[16] James Nechvatal, Elaine Barker, Lawrence Bassham, William Burr, Morris Dworkin, James Foti, and Edward Roback. Report on the development of the advanced encryption standard (aes). *Journal of Research of the National Institute of Standards and Technology*, 106(3):511, 2001.

[17] Data Encryption Standard et al. Data encryption standard. *Federal Information Processing Standards Publication*, 112, 1999.

[18] Wikipedia contributors. Triple des — Wikipedia, the free encyclopedia, 2022. [Online; accessed 30-April-2022].

[19] Tingyuan Nie and Teng Zhang. A study of des and blowfish encryption algorithm. In *Tencon 2009-2009 IEEE Region 10 Conference*, pages 1–4. IEEE, 2009.

[20] Muneer Bani Yassein, Shadi Aljawarneh, Ethar Qawasmeh, Wail Mardini, and Yaser Khamayseh. Comprehensive study of symmetric key and asymmetric key encryption algorithms. In *2017 international conference on engineering and technology (ICET)*, pages 1–7. IEEE, 2017.

[21] Kamlesh Gupta and Sanjay Silakari. Ecc over rsa for asymmetric encryption: A review. *International Journal of Computer Science Issues (IJCSI)*, 8(3):370, 2011.

[22] Quynh Dang et al. *Recommendation for applications using approved hash algorithms*. US Department of Commerce, National Institute of Standards and Technology ..., 2008.

[23] Surbhi Gupta, Neha Goyal, and Kirti Aggarwal. A review of comparative study of md5 and ssh security algorithm. *International Journal of Computer Applications*, 104(14), 2014.

CD Használati útmutató

A CD 2 fő jegyzéket tartalmaz, az Alkalmazás és a Dokumentumok jegyzéket.

Az Alkalmazás mappában megtalálható a

- User_manual mappa, amiben az alkalmazás kezeléséhez szükséges használati útmutató pdf-je található.
- Source_code mappa, ami az elkészített alkalmazás forráskódját tartalmazza. A mappán belül a src/Main/MainClass.java fájlban található az alkalmazást indító main függvény.
  Az elkészített programok telepítéséhez, futtatásához tartozó feltételek a használati útmutatóban vannak leírva.
- Release mappa, melyben az alkalmazás található futtatható .jar kiterjesztésben.

A Dokumentumok mappában megtalálható(k) a

- Adatszerkezetek, Tervezés és Titkosítási_dokumentumok mappák, amelyek az összes, a dolgozattal kapcsolatosan összegyűjtött anyagot tartalmazzák .docx, .pdf és .txt kiterjesztésű fájlokban.
- CD_User_Manual mappa, a CD használati útmutatójának pdf-ét tartalmazza.
- Szakdolgozat mappa, amiben a dolgozat LaTeX forráskódja található.
- Szakdolgozat_pdf mappa, ami a szakdolgozat pdf verzióját tartalmazza.