Examensarbete, Högskolan på Åland, Utbildningsprogrammet för Informationsteknik

PLATTFORMSOBEROENDE LÖSENORDSHANTERARE

Joacim Bondas

2021:33

Datum för godkännande: 12.05.2021
Handledare: Agneta Eriksson-Granskog

EXAMENSARBETE
Högskolan på Åland

Utbildningsprogram:	Informationsteknik
Författare:	Joacim Bondas
Arbetets namn:	Plattformsoberoende lösenordshanterare
Handledare:	Agneta Eriksson-Granskog
Uppdragsgivare:	-

Abstrakt

Syftet med det här examensarbetet var att utveckla en plattformsoberoende lösenordshanterare i utbildningssyfte. I det här arbetet har jag fått bygga vidare på den kunskap jag har fått från skolan samt lära mig helt nya tekniker.

Utveckling av applikationen har skett i C++ med hjälp av biblioteken Boost, wxWidgets och CryptoPP.

Resultatet är en plattformsoberoende lösenordshanterare med grafiskt gränssnitt som sparar lösenord i en krypterad fil.

Nyckelord (sökord)

C++, CryptoPP, wxWidgets, kryptografi, lösenord, hashning

Högskolans serienummer:	ISSN:	Språk:	Sidantal:
2021:33	1458-1531	Svenska	24 sidor

Inlämningsdatum:	Presentationsdatum:	Datum för godkännande:
30.04.2021	12.05.2021	12.05.2021

DEGREE THESIS
Åland University of Applied Sciences

Study program:	Information Technology
Author:	Joacim Bondas
Title:	Cross-platform Password Manager
Academic Supervisor:	Agneta Eriksson-Granskog
Technical Supervisor:	-

Abstract

The purpose of the assignment was to develop a cross-platform password manager for educational purposes. During the assignment I have been able to improve on the knowledge I have acquired from school learning new techniques.

The application has been developed in C++ with the use of the Boost, wxWidgets and CryptoPP libraries.

The result is a cross-platform password manager with a graphical user interface that saves passwords in an encrypted file.

Keywords

C++, CryptoPP, wxWidgets, cryptography, password, hashing

Serial number:	ISSN:	Language:	Number of pages:
2021:33	1458-1531	Swedish	24 pages

Handed in:	Date of presentation:	Approved on:
30.04.2021	12.05.2021	12.05.2021

INNEHÅLLSFÖRTECKNING

1. INLEDNING

1.1 Syfte

Syftet med examensarbetet är att designa och implementera en enkel lösenordshanterare som även kan generera lösenord i önskad längd och komplexitet. Filen med lösenord sparas lokalt på datorn i ett krypterat format och kan låsas upp av användaren med rätt lösenord. Designen på användargränssnittet ska vara lättförstådd och programmet ska vara plattformsoberoende. Syftet med detta projekt är dock inte att skapa en kommersiell produkt, utan det är endast i utbildningssyfte jag utvecklar applikationen.

1.2 Metod

Utvecklingsarbetet görs i programmeringsspråket C++ och det grafiska gränssnittet skapas med hjälp av det öppna källkodsbiblioteket wxWidgets. Krypteringen av lösenordsfilen görs med hjälp av Chacha20 och hashningen av lösenord med hjälp av hkdf.

Programmeringsspråket C++ har jag studerat på flera kurser inom IT på högskolan. Övriga tekniker har jag fått viss kännedom om på högskolan och studerar i detalj genom självstudier inom ramen för detta projekt. För en användarvänlig design använder jag synpunkter från utomstående, i första hand från vänner och bekanta. Som projektmetod använder jag ett inkrementellt arbetssätt.

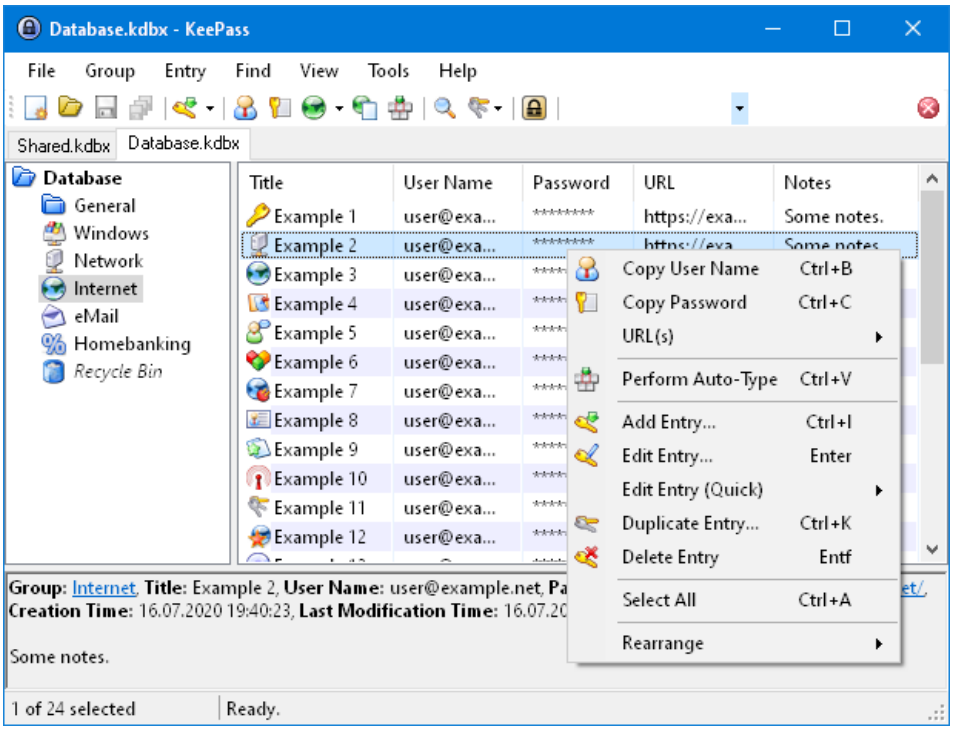
1.3 Avgränsningar

Lagringen av lösenorden kommer endast vara lokalt på datorn. Lösenordshanteraren kommer inte uppdatera lösenord automatiskt efter att användaren bytt lösenord på sidan/applikationen.

2. KRAV

2.1 Vilken funktionalitet skall ingå i applikationen?

Vid valet av funktionalitet som skall ingå så var den viktigaste faktorn praktikalitet. Eftersom jag är den enda utvecklaren så är det omöjligt att ha all funktionalitet som en kommersiell produkt har. Jag valde således att endast ta med det absolut viktigaste för att få en fungerande applikation. Lösenord ska sparas på ett säkert sätt. En krypterad fil sparas på användarens hårddisk. För att komma åt innehållet på filen ska användaren öppna upp filen i applikationen och skriva in sitt lösenord för öppning av filen. Då avkrypteras filen och användaren kan redigera sin fil och lägga till/ta bort information. Då programmet stängs så krypteras filen och innehållet är återigen säkert. För att lättare hålla koll på alla lösenord som sparas på filen så har jag också lagt till möjligheten att dela upp lösenorden i olika kategorier. Den största inspirationen till min applikation är KeePass, en gratis lösenordshanterare med öppen källkod. I Figur 1 ser vi en bild på KeePass.



Figur 1. Ett exempel på en lösenordshanterare, i detta fall KeePass. (Reichl, n.d.)

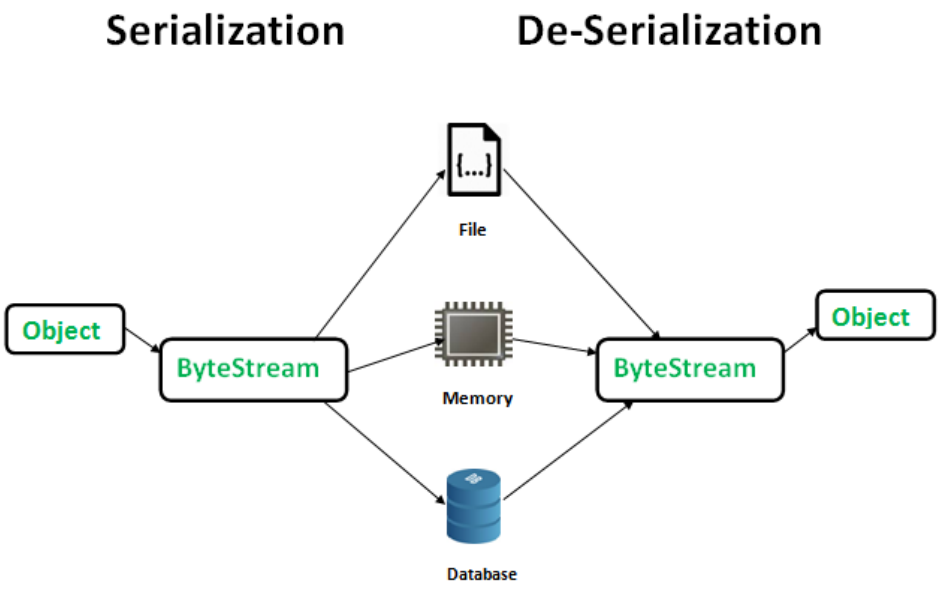
3. TEKNIK

3.1 Programmeringsspråket C++

C++ är ett programmeringsspråk skapat av Bjarne Stroustrup. I början var tanken att språket skulle vara C med klasser men har med tiden utvecklats till ett helt eget språk även om syntaxen kan likna C-kod (Albatross, n.d.). Orsaken till att jag valde C++ som programmeringsspråk var att jag har använt det i skolan i flera kurser och är väldigt bekant med språket.

3.2 Boost

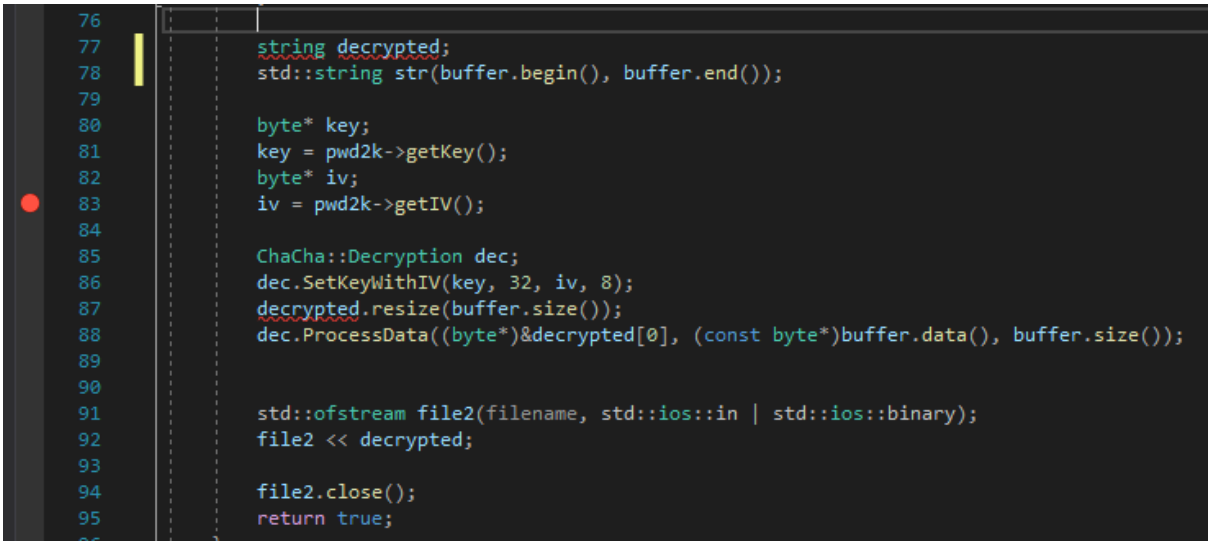
Boost är en samling bibliotek för C++. Boost är också öppen källkod. Till projektet använde jag Boosts Serialization (på svenska Serialisering) bibliotek. Serialisering konverterar ett objekt till en byte ström och den strömmen sparar jag till en fil (*Serialization*, n.d.). I Figur 2 ser vi en illustration av flödet under serialisering.



Figur 2. Flödet under serialisering (Serialization and Deserialization in Java with Example - GeeksforGeeks, 2016).

3.3 Visual Studio

Visual Studio är en programutvecklingsmiljö från Microsoft. I Visual Studio ingår en kompilator för C++. En kompilator översätter människo-skriven kod till maskinkod som datorn förstår. Visual Studios kod redigerare har syntaxmarkering som visar källkod i olika färger vilket hjälper väldigt mycket med läsbarheten och visar syntaxfel. Visual Studio har även en debugger. Debuggern är ett verktyg som används under programmets körning. Man kan se variablers värden och följa flödet av programmet vilket är väldigt användbart då man felsöker ett program. I Figur 3 kan vi se ett exempel på hur C++-kod kan se ut i Visual Studio. I exemplet syns även Visual Studios syntaxmarkeringar.



Figur 3. Visual Studio med syntaxmarkering, syntaxfel markeras med röd understrykning, och en breakpoint för avlusning.

3.4 Git, Git Bash och GitHub

Git är ett versionshanteringssystem vilket innebär att man kan spara tidigare versioner av ett program och gå tillbaka till en tidigare version ifall någonting går fel i den senaste versionen. I grunden är Git ett UNIXterminalbaserat program. (*About*, n.d.)

Visual Studio erbjuder inbyggda verktyg för Git men jag är mer van med terminalmiljön så jag bestämde mig för att fortsätta med det. Eftersom jag använder Microsoft Windows så behövde jag ett verktyg för att emulera en UNIXterminal. Git Bash är ett sånt verktyg. Git Bash installerar både Git och Bash på Windows datorn och gör det möjligt att använda Git-kommandon (Atlassian, n.d.).

GitHub är en webbapplikation som erbjuder lagring av versionshistorik för programmeringsprojekt. De flesta projekt på GitHub är publika så vem som helst kan ladda ner källkoden och kolla på den. Jag använde GitHub för att spara mitt projekt där (*About Repositories*, n.d.).

Versionshantering är en bra försäkring ifall jag skulle råka ta bort något viktigt eller om min dator skulle sluta fungera mitt i projektet.

4. GRAFISKT GRÄNSSNITT

4.1 wxWidgets

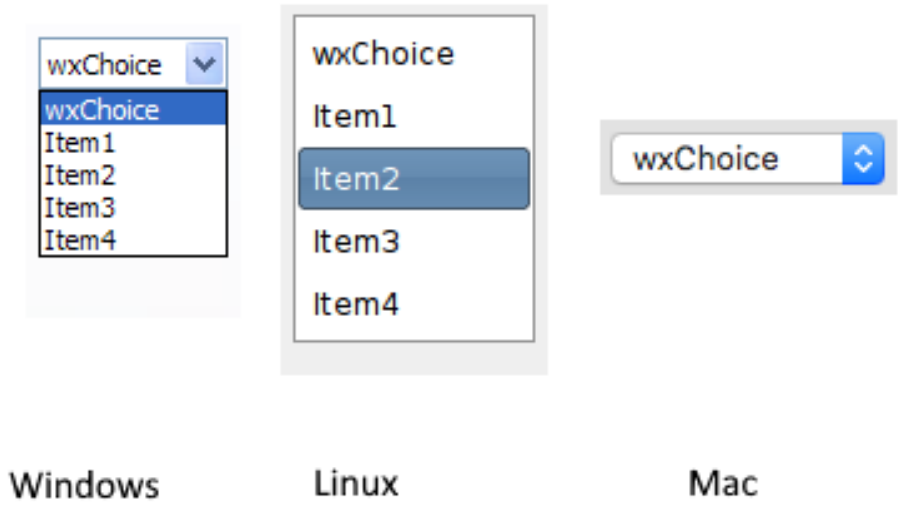
wxWidgets(wxWidgets Team, n.d.) är ett öppet källkodbibliotek för C++ som ger möjlighet till s.k. plattformsoberoende grafiska gränssnitt.

4.1.1 Open source

Öppen källkod är mjukvara där skaparna valt att lägga upp källkoden publikt. Detta innebär att i princip vem som helst kan bidra med förbättringar till mjukvaran. Mjukvaran är då också gratis att använda till privata projekt, om licensen tillåter så kan mjukvaran också användas i kommersiella projekt. wxWidgets-licens (GNU Library General Public Licence) (*The wxWindows Library Licence (WXwindows)*, n.d.) tillåter användning och modifiering i kommersiella projekt.

4.1.2 Plattformsoberoende

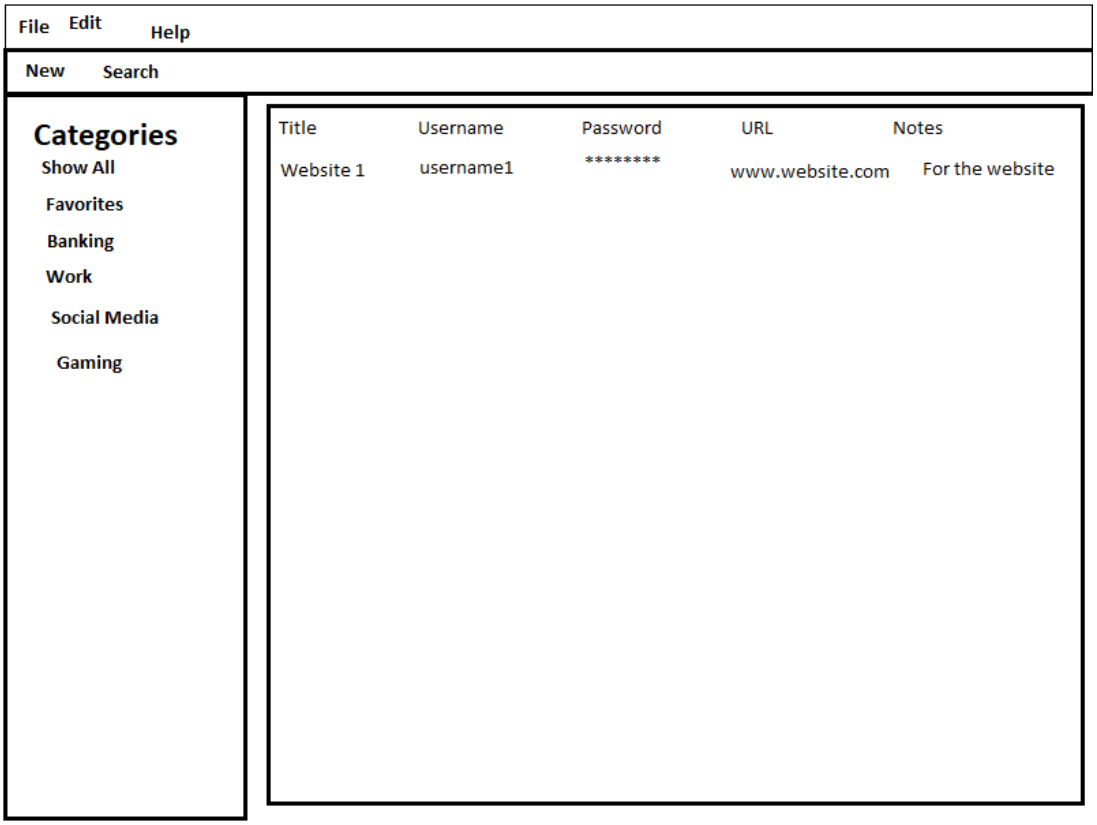
Detta betyder att utseendet på knappar, menyer osv bestäms av operativsystemet. Då biblioteket är plattformsoberoende räcker det med att skriva programmet en gång för att det ska fungera på Windows-, Mac- och Linuxdatorer. I Figur 4 kan vi se exempel på hur en rullgardinsmeny ser ut i olika operativsystem.



Figur 4. Exempel på wxChoice, en rullgardinsmeny, i olika operativsystem. (wxWidgets: Screenshots of Different Controls, n.d.)

4.2 Design

Då det kommer till designen av det grafiska gränssnittet så valde jag att använda MS Paint istället för något specifikt anpassat verktyg för design av grafiska användargränssnitt (GUI). Orsaken är att jag ville ha en väldigt enkel design för applikationen för att göra den så användarvänlig som möjligt och jag ville inte ödsla tid på att lära mig använda ett komplicerat program då jag i princip får samma resultat från MS Paint. För att få en användarvänlig design har jag bett vänner och bekanta ge förslag på vad som kan förbättras, och vad de skulle vilja se i en lösenordshanterare. I Figur 5 ser vi en skiss på den önskvärda designen jag gjorde i MS Paint vilket gjorde det mycket lättare att bygga upp ett grafiskt gränssnitt i programmeringsfasen.



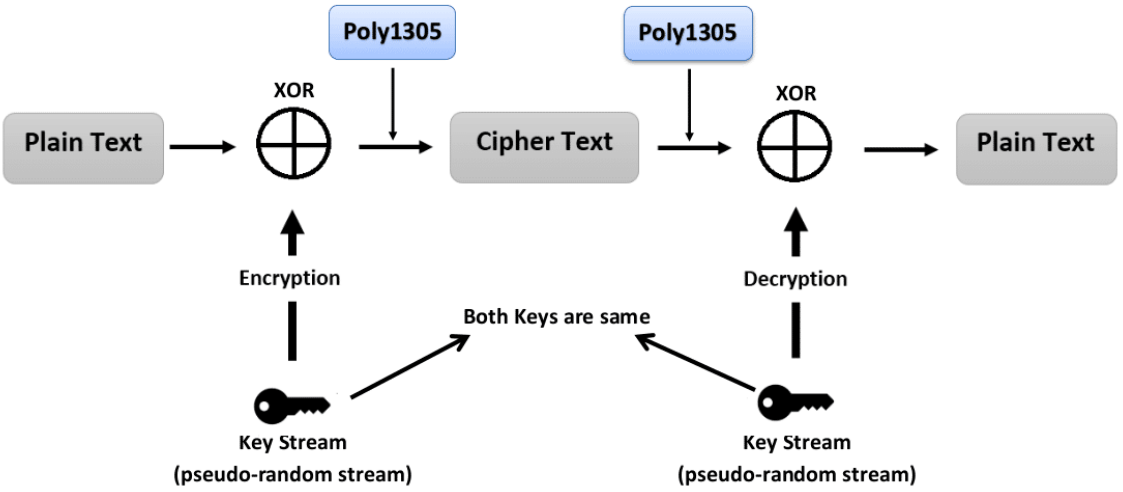
Figur 5. Första designen i MS Paint

5. KRYPTERING

Det viktigaste med en lösenordshanterare är att lösenorden är i säkert förvar. Därför måste filen med alla lösenord krypteras för att undvika att obehöriga personer kommer åt den privata informationen.

5.1 ChaCha20

Krypteringsalgoritmen jag använde kallas ChaCha20, som egentligen är ett strömkrypto utvecklat av Daniel J. Bernstein(Bernstein, 2008). ChaCha20 är baserat på Salsa20 familjen som också är utvecklat av Bernstein. ChaCha20 är en ARX (Add, Rotate, XOR(på svenska modulär addition, rotation och exklusiv disjunktion)) baserad hashfunktion som är väldigt snabb och lätt, vilket var orsaken till att jag använde den i lösenordshanteraren. Algoritmen använder sig av en nyckelströmsgenerator som tar in en 256 bitars nyckel som användaren bestämmer, ett nummer som används en gång(nonce, number used once), och ett block nummer. Nyckelströmsgeneratorn genererar då nyckelbitar som kombineras med det okrypterade meddelandet genom exklusiv disjunktion. Vi får då vårt chiffer. Processen för att avkryptera ett meddelande går till på samma sätt, man skickar in chiffret istället för det okrypterade meddelandet och får ut det okrypterade meddelandet istället. I Figur 6 ser vi ett exempel på flödet vid en ChaCha20 kryptering. Poly1305 är en autentiseringskod som garanterar meddelandets integritet.



Figur 6. Bild på flödet vid en ChaCha20 kryptering (javainterviewpoint, 2019).

5.2 HKDF

För att kryptera och avkryptera filen behöver jag då en 256 bitars nyckel. Att kräva att en användare av lösenordsgeneratorn bara ska kunna använda sig av ett lösenord som är 256 bitar långt är inte speciellt användarvänligt. För att tillåta att användaren använder ett lösenord av valfri längd så behövs en nyckelgenereringsfunktion.

Min lösning blev HKDF. HKDF är en nyckelderiveringsfunktion skapad av Krawczyk och Eronen som är baserad på en hash-baserad autentiseringskod (HMAC). HKDF extraherar en pseudoslumpmässig nyckel och expanderar den till flera slumpmässiga nycklar. HKDF används bland annat av protokollet IPsec som är säkert nätverksprotokoll som autentiserar och krypterar datapaket mellan två datorer över ett internetprotokoll (*HKDF*, n.d.).

HKDF tillåter olika hashalgoritmer. Hashning transformerar data till ett säkert och oläsligt format och transformerar datat tillbaka till sitt ursprungliga format med hjälp av en nyckel. För att kryptera lösenorden använde jag SHA-256. SHA2 står för Secure Hash Algorithm 2 och används bland annat av den amerikanska underrättelseverksamheten. Det är näst intill omöjligt att återskapa det ursprungliga data från det hashade värdet, det skulle kräva 2^{256} försök (*SHA-256 Algorithm*, 2019).

5.3 CryptoPP

Kryptografi är ett brett och väldigt komplicerat område. Det krävs en djup förståelse för området och lång erfarenhet för att vara säker på att krypteringen faktiskt är säker. Inom informationsteknologin sätts allt större fokus på säkerhet och kapplöpningen mellan de som vill skydda sin information och de som vill komma åt den rusar ständigt vidare. Att självständigt implementera krypteringsalgoritmer på ett säkert sätt är då väldigt komplicerat, speciellt utan lång erfarenhet och en utbildning specifikt inriktad på säkerhet.

Som tur är finns det bibliotek som sköter största delen av implementeringen. Jag använder mig av CryptoPP som är ett bibliotek med öppen källkod. Biblioteket erbjuder många av de populäraste samt mindre populära kryptografiska algoritmer, bland annat ChaCha20(*ChaCha20*, n.d.). En stor del av jobbet är då redan gjort och implementeringen går mycket lättare. I Figur 7 är den kod jag använder för att kryptera innehållet i en fil. Variabeltypen för variablerna key(nyckel) och IV(initialiseringsvektorn) är SecBlock som erbjuds av biblioteket CryptoPP. SecBlock erbjuder säker lagring av nycklar som fullständigt raderas då variabeln tas bort eller töms (*SecBlock*, n.d.).

```
ChaCha::Encryption enc;
enc.SetKeyWithIV(key, 32, iv, 8);
cipher.resize(str.size());
enc.ProcessData((byte*)&cipher[0], (const byte*)str.data(), str.size());
```

Figur 7. All kod som behövs för att kryptera en sträng med ChaCha20 algoritmen i CryptoPP.

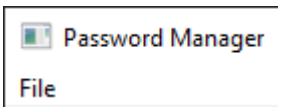
&cipher[0] ger det icke-konstanta värdet från en std::string (standardsträng i C++). Nyckeln och initialiseringsvektorn skapar jag i en annan klass och min krypteringsklass hämtar dessa värden därifrån. Strängen str är det okrypterade innehållet från filen och cipher är en tom sträng där jag allokerar tillräckligt med minne för att rymma det krypterade innehållet. Strängarna konverteras till CryptoPPs version av byte. Funktionen ProcessDats sköter då själva krypteringen och matar in det krypterade värdet i ciphersträngen.

6. UTVECKLING AV APPLIKATIONEN

6.1 Skapa ett grafiskt användargränssnitt

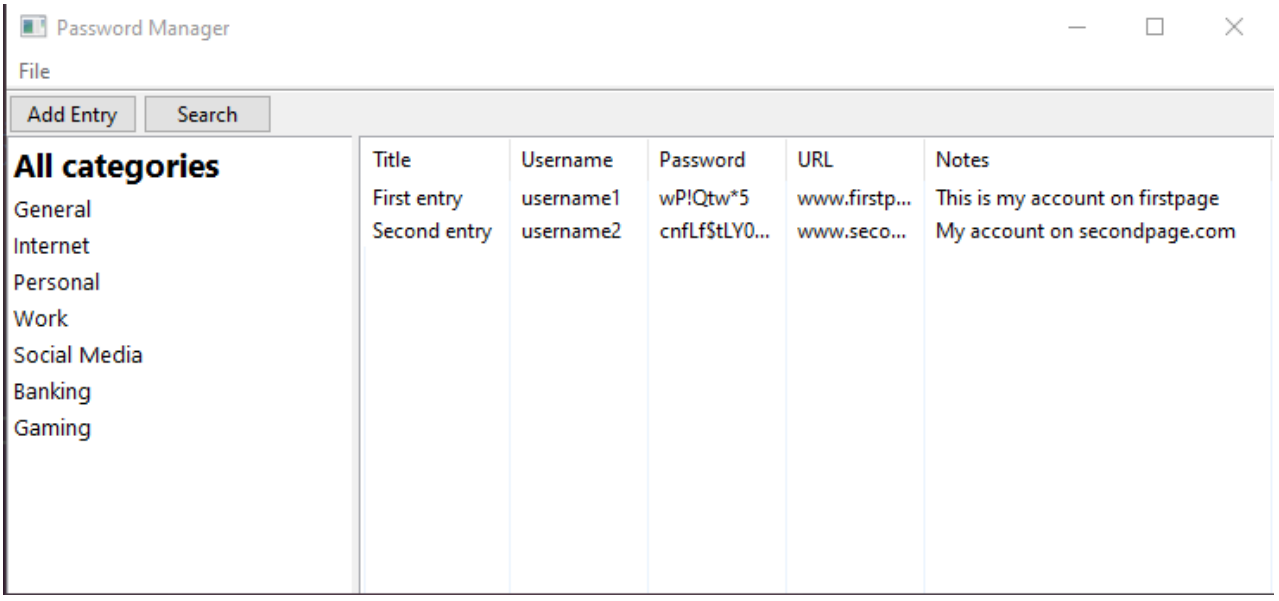
Jag använde wxWidgets för att få ett grafiskt gränssnitt till min applikation. Det första som behöver skapas upp är ett fönster. Klassen wxFrame är ett fönster som användaren kan ändra storlek på. I Figur 8 ser vi ett exempel på den kod jag använder för att lägg till en filmeny på applikationsfönstret.

```
m_MenuBar = new wxMenuBar();
this->SetMenuBar(m_MenuBar);
wxMenu* menuFile = new wxMenu();
menuFile->Append(10001, "New");
menuFile->Append(10002, "Open");
menuFile->Append(10003, "Save");
menuFile->Append(10004, "Exit");

m_MenuBar->Append(menuFile, "File");
```



Figur 8. Exempel på hur man lägger till en filmeny med hjälp av wxWidgets wxMenuBary funktion.

Huvudfönstret är uppdelat i två paneler. Den mindre panelen innehåller en lista på de olika kategorierna, den större innehåller en lista på alla sparade lösenord. I Figur 9 ser vi en bild på huvudfönstret.



Figur 9. Huvudfönstret i applikationen. En lista på kategorier till vänster och en lista på sparade lösenord till höger.

wxWidgets erbjuder en funktion som delar upp fönstret dynamiskt så man kan ändra på förhållandet mellan de två sidorna. Kategorierna är en lista av formatet wxSimpleHtmlListBox. wxSimpleHtmlListBox tillåter olika html stilformateringar som storlek, färg och typsnitt.

6.1.2 Dialoger

Jag har flitigt använt mig av olika dialoger i applikationen. Fildialogen i kapitel 6.2 är ett exempel på en dialog. En dialog är ett litet fönster med begränsad funktionalitet som jag använder för att t.ex visa ett lösenordsobjekt eller generera ett lösenord. Då en dialog öppnas så är huvudfönstret i ett inaktivt läge tills dialogen stängs.

6.2 Filhantering

För att applikationen skall komma ihåg alla lösenord tills nästa gång man startar upp den sparar jag allt i en fil. Filens format är binärt, med filändelsen .dat. I praktiken spelar filändelsen ingen roll, då innehållet är binärt kan programmet läsa filen oavsett ändelse men för binära filer är dat och bin de vanligaste ändelserna. För att låta programmet veta vilken fil användaren vill jobba mot använde jag wxFileDialog. wxFileDialog öppnar utforskaren, ett filhanteringsverktyg som låter användaren öppna, spara och skapa nya filer. Då användare

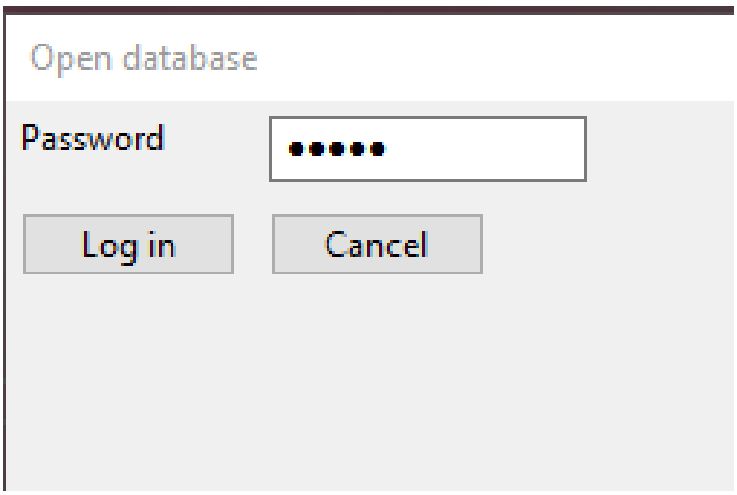
valt vilken fil applikationen ska jobba med kopierar jag filens innehåll till en temporär fil. Orsaken till att jag valt att arbeta med en temporär fil är att ifall programmet skulle avslutas på ett oväntat sätt så går inte all information i filen förlorad. Originalfilen är på detta sätt alltid krypterad. Då programmet avslutas kopieras all data från den temporära filen till originalfilen. I Figur 10 ser vi ett exempel på kod för att öppna utforskaren.

```
wxFileDialog
    openFileDialog(this, _("Open dat file"), "", "",
        "dat files (*.dat)|*.dat", wxFD_OPEN | wxFD_FILE_MUST_EXIST);
if (openFileDialog.ShowModal() == wxID_CANCEL)
    return;

wxFileInputStream input_stream(openFileDialog.GetPath());
```

Figur 10. Kod för att öppna utforskaren.

6.3 Kryptering/Avkryptering av fil

Om användaren väljer att öppna en existerande fil så öppnar applikationen en dialog där användaren får skriva in sitt lösenord. Programmet kollar då om lösenordet är korrekt och filen kan avkrypteras, om så är fallet öppnas huvudfönstret. Om lösenordet är fel och filen inte kan avkrypteras så öppnas en till dialog som frågar användaren om hen vill försöka med ett annat lösenord eller avbryta. Ett exempel på inloggningsdialogen ser vi i Figur 11.



Figur 11. Bild på inloggningsdialog då användaren valt att öppna en existerande databas.

För att avkryptera filen behövs en 256 bitars-nyckel samt en initialiseringsvektor. Jag har skapat en klass som genererar dessa nycklar utgående från lösenordet användaren skriver in. Förutom lösenordet använder nyckelgenereringen även ett salt, en hemlighet och info. Info och salt kan vara allmänt kända men hemligheten kan inte vara känd då det riskerar att någon obehörig kan demontera nyckelgenereringen. Det är också en god idé att använda olika parametrar vid generering av nyckel och initialiseringsvektor ifall en av dom blir äventyrade. Om nyckeln är rätt så avkrypteras filen till ett läsligt format. Om inte så returneras ett meddelande att lösenordet är inkorrekt. I Figur 12 ser vi ett kodexempel på nyckelgenerering med standardvärden för salt, hemlighet och info.

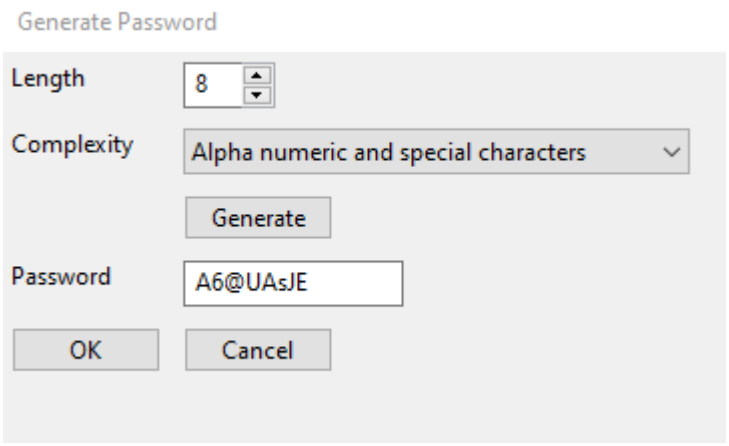
```
const byte* bytePassword = (const byte*)password.data();
size_t plen = strlen((const char*)bytePassword);
byte salt[] = "salt";
size_t slen = strlen((const char*)salt);

byte info1[] = "HKDF key derivation";
size_t ilen1 = strlen((const char*)info1);

byte info2[] = "HKDF iv derivation";
size_t ilen2 = strlen((const char*)info2);

HKDF<SHA256> hkdf;
hkdf.DeriveKey(key, sizeof(key), bytePassword, plen, salt, slen, info1, ilen1);
hkdf.DeriveKey(iv, sizeof(iv), bytePassword, plen, salt, slen, info2, ilen2);
```

Figur 12. Kodexempel på nyckelgenerering där parametrarna inte är speciellt hemliga.

6.4 Lösenordsgenerering

Då det kommer till lösenord och säkerhet så är det viktigt att inte använda samma lösenord till allt, men det kan vara svårt att komma på unika lösenord till alla olika webbplatser. Därför har jag också valt att implementera en lösenordsgenerator till applikationen. Olika sidor kan ha olika krav på hur långa och komplicerade lösenorden ska vara. I min lösenordsgenerator kan man välja tre olika kategorier på lösenordets komplexitet; alfanumeriskt med små bokstäver, alfanumeriskt med små och stora bokstäver samt alfanumeriskt med små och stora bokstäver samt specialtecken(!@#$%&*). Längden på lösenordet väljer man själv i en rullningsruta.

För att få ett pseudoslumpmässigt genererat lösenord använder jag mig av Boost-bibliotekets random_device funktion som är en slumpgenerator. Slumpgeneratorn ger slumpmässigt utvalda heltal inom ett förbestämt intervall som jag använder för att plocka ut slumpmässiga värden ur en räcka som innehåller de tecken användaren vill ha med i lösenordet. Före lösenordet returneras görs en kontroll på att de angivna lösenordskraven är uppfyllda. I Figur 13 ser vi ett exempel på lösenordsgeneratorn i applikationen.



Figur 13. Lösenordsgeneratorn som har genererat ett lösenord som är åtta karaktärer långt.

6.5 Lösenordsobjekt

Vad jag valt att kalla lösenordsobjekt i brist på bättre namn är egentligen en serialiseringsbar klass som innehåller en titel, användarnamn, lösenord, anteckningar, url, id, index och kategori. IDt är ett så kallat UUID, en universell, unik identifierare som är ett 128 bitars nummer som jag får från Boost-bibliotekets UUID-generator. Detta använder jag för att unikt identifiera alla objekt ifall användaren av någon orsak skulle skapa två objekt med samma titel, vilket annars skulle leda till att all information i det gamla objektet skulle raderas. Dessa objekt är serialiseringsbara och kan således konverteras till en byteström som sedan läses in till en fil. I Figur 14 ser vi ett exempel på hur en okrypterad lösenordsfil ser ut. De långa strängarna i formatet xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx är det unika IDt.

```
22 serialization::archive 18 0 0 0 0 2 0 1 1 0
0 11 First entry 9 username1 8 wP!Qtw*5 31 This is my account on firstpage
17 www.firstpage.org 36 9deb303e-4290-4b0e-95b3-26ac6548b67b 0 8 Internet
1
1 12 Second entry 9 username2 16 cnfLf$tLY08DM0GH 28 My account on
secondpage.com 18 www.secondpage.com 36 31cf34e4-8df9-4f24-ba5e-
2dd4da2decab -842150451 8 Internet
```

Figur 14. Exempel på en okrypterad lösenordsfil.

Kategorierna är mest ett hjälpmedel ifall användaren har väldigt många lösenord och behöver kunna sortera efter kategori. Kategorierna jag valde att ta med är allmänt, internet, personligt, arbete, sociala media, bankrelaterat och underhållning.

7. SLUTSATSER

Syftet med detta examensarbete var att utveckla en lösenordshanterare i utbildningssyfte och det är vad jag har gjort. Det går att spara lösenord på ett säkert sätt och utan rätt lösenord kommer man inte åt den krypterade informationen. Detta är vad jag gjort under arbetets gång:

- Skapat upp ett grafiskt användargränssnitt med hjälp av wxWidgets
- Serialiserat informationen som skall sparas med hjälp av Boost
- Krypterat informationen med algoritmen ChaCha20
- Genererat nyckel från ett lösenord skapat av användaren för att kryptera/avkryptera informationen

Jag lyckades med att arbeta inkrementellt och dela upp arbetet i mindre delar för att senare bygga ihop allt till en enda applikation. Självklart fick jag ibland hoppa tillbaka till ett tidigare moment och förbättra saker jag kom på i efterskott men överlag gick det väldigt bra att arbeta på ett strukturerat sätt.

Under projektets gång har jag fått lära mig mycket om kryptografi, att arbeta med grafiska gränssnitt i C++ och i andra bibliotek jag har fått använda. En stor del av tiden gick åt till att läsa på om olika krypteringsalgoritmer, vilka som är säkra och vilka det avråds från att använda i dagens läge.

Det finns mycket jag skulle vilja fortsätta på och förbättra i framtiden. Utseendet på applikationen kunde helt klart förbättras, kortkommandon från tangentbordet hade också varit en bra funktion. Överlag blev jag väldigt nöjd med vad jag lyckades producera på egen hand och mina kunskaper inom området har definitivt förstärkts. Mitt intresse för kryptografi och IT-säkerhet har också ökat markant efter att ha fått jobba med det.

KÄLL- OCH LITTERATURFÖRTECKNING

About. (n.d.). Branching and Merging. Retrieved April 13, 2021, from https://git-scm.com/about

About repositories. (n.d.). Github. Retrieved April 13, 2021, from https://docs.github.com/en/github/creating-cloning-and-archiving-repositories/about-repositories

Albatross. (n.d.). *History of C++*. Cplusplus. Retrieved April 13, 2021, from https://www.cplusplus.com/info/history/

Atlassian. (n.d.). *[No title]*. Retrieved April 13, 2021, from https://www.atlassian.com/git/tutorials/git-bash

Bernstein, D. J. (2008). *ChaCha, a variant of Salsa20*. The University of Illinois at Chicago.

ChaCha20. (n.d.). Retrieved March 11, 2021, from https://www.cryptopp.com/wiki/ChaCha20

HKDF. (n.d.). Retrieved April 21, 2021, from https://www.cryptopp.com/wiki/HKDF

javainterviewpoint. (2019, April 29). *Java ChaCha20 Poly1305 Encryption and Decryption Example*. https://www.javainterviewpoint.com/chacha20-poly1305-encryption-and-decryption/

Reichl, D. (n.d.). *KeePass password safe*. Retrieved April 28, 2021, from https://keepass.info/

SecBlock. (n.d.). Retrieved April 21, 2021, from https://www.cryptopp.com/wiki/SecBlock

Serialization. (n.d.). Retrieved April 13, 2021, from https://www.boost.org/doc/libs/1_75_0/libs/serialization/doc/index.html

Serialization and Deserialization in Java with Example - GeeksforGeeks. (2016, January 13). https://www.geeksforgeeks.org/serialization-in-java/

SHA-256 Algorithm. (2019, September 12). https://www.n-able.com/blog/sha-256-encryption

The wxWindows Library Licence (WXwindows). (n.d.). Retrieved March 11, 2021, from https://opensource.org/licenses/wxwindows.php

wxWidgets: Screenshots of Different Controls. (n.d.). Retrieved April 13, 2021, from https://docs.wxwidgets.org/trunk/page_screenshots.html

wxWidgets Team. (n.d.). *[No title]*. Retrieved March 11, 2021, from https://www.wxwidgets.org/