

# ENIGMA-PRO

Open-Source Password-Manager in C#

AUTOR: RACHID MZANNAR

ERSTELLUNG: 02.02.2016

ABGABE: 18.05.2016

BETREUENDE LEHRKRAFT: M. PRAMANN

FACH: PRAXIS



Rachid Mzannar

# Enigma-Pro

Facharbeit im Fach Praxis

Erstellungsdatum: 01.02.2016 – 15.02.2016

Abgabedatum: 18.02.2016

Bei M. Pramann

## Inhaltsverzeichnis

Vorwort .....	IV
Einleitung.....	1
Einführung in das Projekt .....	1
Oberfläche .....	1
Programmierung .....	2
Klassenbibliothek .....	2
XmlHandler .....	2
Datenbank verschlüsseln und entschlüsseln.....	2
Importieren und Exportieren.....	3
Konfiguration .....	3
DialogManager .....	3
ListView .....	3
Einträge .....	4
Key-File .....	5
DatabaseHandler .....	6
Datenbank anlegen.....	6
Datenbank öffnen .....	6
Datenbank schließen .....	6
Implementierung .....	6
Haupt-Fenster.....	6
Einträge hinzufügen-Fenster.....	7
Einträge bearbeiten-Fenster .....	8
Info-Fenster .....	8
Dokumentation.....	8
Fazit.....	10
Ausblick .....	10
Quellenverzeichnis .....	11
Anhang .....	IV
Screenshots .....	IV

Software.....	VII
Eigenständigkeitserklärung .....	VII

## Vorwort

Die eigentliche Idee dieses Projektes war es eine Smartphone App zu erstellen zur Verwaltung von Passwörtern in verschlüsselten Datenbanken. Jedoch gibt es bereits genügend Tools, die dies schon können. Für Windows hingegen gibt es nicht allzu viele Password-Manager, daher habe ich mich entschieden das Programm für Microsoft Windows zu schreiben.

Schon immer habe ich mich mit Kryptographie beschäftigt und bin nach wie vor von diesem Konzept begeistert, sodass ich mir die Herausforderung gestellt habe, selbst eine Verschlüsselungsbibliothek zu programmieren und ein Passwort-manager für das Verwalten der Daten.

## Einleitung

### Einführung in das Projekt

Bei dieser Projektarbeit handelt es sich um eine Software zur Verwaltung von Passwörtern. Die Software wurde mithilfe von Microsoft „Visual Studio 2015 Enterprise“ mit der Programmiersprache C# und dem .NET Framework 4.5 entwickelt. Dadurch ist das Programm für Windows Vista >= kompatibel. Alle Einträge der Datenbank werden in einem verschlüsselten XML-Dokument mit AES-256 gespeichert. Das Projekt ist bereits schon Open-Source auf <https://github.com/> erhältlich und kann weiterhin unter Einhaltung der „GNU GENERAL PUBLIC LICENSE – GPLv3“ weiterentwickelt werden.

Auf die Datensätze kann man nur mit einem vorgenerierten Keyfile zugreifen und so die einzelnen Felder wie Benutzername, Passwort, URL usw. ansehen und verändern. Die Verwaltung dieser Einträge erfolgt über eine dynamische Klasse, welche auch für die Erzeugung der Benutzeroberfläche zuständig ist. Durch Tastenkombinationen bzw. einer Menüleiste können einzelne Einträge wie Benutzername oder Passwort in die Zwischenablage kopiert werden. Des Weiteren erlaubt eine Klasse das Importieren und Exportieren von nicht verschlüsselten XML-Dateien, welche zur Übersicht der Einträge dienen sollen.

Die Projektmappe besteht aus zwei Projekten, der Klassenbibliothek und dem Datenverwaltungsprogramm selbst, welche diese dynamische Bibliothek dann als DLL über eine Using-Direktive importiert.

### Oberfläche

Wie bereits erwähnt werden die einzelnen Elemente (Objekte) der Oberfläche mithilfe einer Klasse generiert und verwaltet. Das heißt, alle erstellten Objekte sind dynamisch und werden in der Laufzeit erstellt. Somit fällt das „Drag-and-Drop“ Prinzip der einzelnen Objekte im Visual Studio Form Designer aus.

Auch das Interface ist dynamisch konzipiert, d.h. die Größe aller Objekte wie Buttons, Labels und die ListView werden automatisch mit der Größe des Fensters angepasst. Weiterhin speichert das Programm die Größe des Fensters, die Position, den Ort und die zuletzt geöffnete Key-Datei (Key = Schlüssel) in XML Dateien.

## Programmierung

In den weiteren Aspekten gehe ich näher auf die Programmierung des Passwortmanagers und der Klassenbibliothek ein.

### Klassenbibliothek

Wie bereits erwähnt gibt es eine vom eigentlichen Programm dynamische Klassenbibliothek (DLL) namens „Enigma.Cryptography.Dll“, welche für die Verschlüsselung und Entschlüsselung der Daten tätig ist. Diese Klasse kann somit in allen anderen Projekten mit einem einfachen Referenzverweis hinzugefügt werden und auf die statischen Methoden zugegriffen werden.

Die Klasse enthält die nötigen Methoden für das Generieren von einem Key und einem Initialisierungsvektor, welche für die AES-256 Verschlüsselung und Entschlüsselung der Datenbank benötigt werden. Des Weiteren enthält die Klasse eine Methode namens ZeroMemory, welche von der „kernel32.dll“ aufgerufen wird um den generierten Key aus dem Speicher zu leeren bzw. mit Nullen zu Überschreiben. Dies ist aus Sicherheitsgründen daher wichtig, da man sonst einfach mit einem Debugger den Speicher auslesen kann und so den Key an der Speicheradresse mit großer Wahrscheinlichkeit herausbekommt. Die weiteren zwei Methoden sind AES\_Encrypt und AES\_Decrypt, welche mit einem Key als Parameter die Daten verschlüsseln sollen.

### XmlHandler

Die statische XmlHandler-Klasse verfügt über mehrere öffentliche statische Methoden zum Laden und Speichern von jeweils verschlüsselten und nicht verschlüsselten Datenbanken in Form von XML-Dateien sowie den Verweis zur Cryptography DLL. Manche dieser Methoden haben ein ListView Objekt und ein SaveFileDialog oder ein OpenFileDialog Objekt als Argumente. So werden die Einträge aus einer ListView in XML-Dateien gespeichert.

### Datenbank verschlüsseln und entschlüsseln

Die weiteren Methoden ExportEncryptedToXml und ImportEncryptedFromXml verwenden die Keyfile und die Methoden aus der Klassenbibliothek um die Einträge mit AES-256 zu verschlüsseln bzw. zu entschlüsseln.

Zusätzlich hat die ExportEncryptedToXml Methode eine Überladung, welche statt einem OpenFileDialog Objekt ein String verwendet. Dies ist für das spätere Speichern von bereits erstellten Datenbanken nötig.

### Importieren und Exportieren

Damit auch nicht verschlüsselte XML-Dateien importiert und exportiert werden können, gibt es zwei Methoden die dies ermöglichen. Für das Exportieren wird die XmlWriter Klasse verwendet und für das Importieren die XmlReader Klasse (siehe: [XmlDocument Class](#)).

### Konfiguration

Für das Speichern und Laden von Fenster Eigenschaften, wie Größe und Position wird auch wie bereits erwähnt eine XML-Datei angelegt und mit den Methoden SaveConfigFile und LoadConfigFile verwaltet. Für das Speichern und Laden der Konfiguration wird die XmlWriter/XmlReader Klasse von Microsoft verwendet (siehe: [XmlDocument Class](#)).

Die letzte Funktion GetRecentKeyFilePath gibt den Pfad der zuletzt verwendeten Keyfile zurück, welche für weitere Klassen von Bedeutung ist.

### DialogManager

Die wichtigste Klasse im Projekt ist die DialogManager Klasse. Diese verfügt sowohl über öffentliche, private und statische Methoden. In dieser Klasse werden alle Objekte erstellt, die später an der Oberfläche dynamisch angelegt werden.

### ListView

In der Klasse befindet sich eine Methode InitializeListView in der die gesamte ListView samt Menü-Leiste gebaut wird und kann dann in einer Form-Klasse mit der eingestellten Größe erstellt werden. Die Objekte welche hier initialisiert werden sind alle private Member. Zusätzlich bekommen die Objekte welche vom Benutzer bedient werden ihre eigenen EventHandler, sogenannte Methoden die dann aufgerufen werden, wenn ein bestimmtes Event eintritt wie z.B. das Rechtsklicken in die ListView.

Hier werden die Spalten, die Menüleiste und das Kontextmenü initialisiert und ihren EventHandler zugeordnet. Es gibt insgesamt fünf Spalten: Titel, Benutzername, Passwort, URL und Notizen.

Auch die Icons für die Menü-Items werden hier aus den Ressourcen geladen. Am Ende werden alle Objekte in den Container von der Form hinzugefügt, sodass beim Aufrufen dieser Funktion die ListView in die Form erstellt wird, statt ein weiteres Fenster zu erstellen. Wichtig hierbei ist noch, dass ein gekapseltes öffentliches ListView Objekt vorhanden sein muss, da andere Klassen über die ListView zugreifen müssen, um bestimmte Eigenschaften auszulesen oder zu setzen.



## Einträge

Die folgenden Funktionen dienen dazu, Einträge zu verwalten, sprich Einträge hinzufügen, bearbeiten, kopieren, löschen und duplizieren.

### Einträge hinzufügen

Um Einträge zu der zuvor erstellten ListView hinzufügen zu können, muss hierfür ein Dialog erstellt werden der diese Einträge mit benutzerdefinierten Inhalten befüllt. Zum Befüllen der Einträge werden Textboxen verwendet.

Hierbei für ist die private Methode AddNewEntry zuständig, welche ein ListViewItem erstellt um die Einträge mit den Inhalten der Parameter als String zu befüllen. Wenn der Benutzer beispielsweise eine URL ohne WWW-Subdomain eingibt, wird diese an dem String mit eingehängt. Dies ist daher nötig, da beim Öffnen einer URL eines Eintrages die WWW-Subdomain benötigt wird. Des Weiteren wird auch überprüft, ob das eingetragene Passwort mit dem des zu wiederholenden Passworts übereinstimmt. Stimmen die Inhalte, so werden sie dann in das gekapselte ListView Feld hinzugefügt.

### Einträge bearbeiten

Um schließlich später auch die Einträge bearbeiten zu können, wird ein weiteres Fenster benötigt. In dieser Methode werden bereits bei der Initialisierung die Einträge des ausgewählten Datensatzes in den Textboxen reingeschrieben. Dies erfolgt mit einer foreach-Schleife, welche jedes ListViewItem in die Textboxen befüllt. Die Methode kann im Übrigen auch nur aufgerufen werden, wenn sich Einträge in der ListView befinden. Dies lässt sich einfach mit dem Count der ausgewählten Items in der ListView feststellen: Ist der Count größer als 0, so ist mindestens ein Eintrag vorhanden und das Fenster wird angezeigt.

Auch hier ist wieder eine weitere private Methode beim Bearbeiten der Einträge zuständig, die EditEntry Funktion, welche auch einmal durch die ListViewItems iterieren muss um die Einträge zu setzen. Die beiden Methoden AddNewEntry und EditEntry sind deshalb nur innerhalb der Klasse aufzufinden, da sie wiederum von EventHandlerler aufgerufen werden müssen, welche auch alle privat sind.

### Einträge kopieren

Damit ein ausgewählter Eintrag kopiert werden kann, gibt es verschiedene Funktionen die dies ermöglichen. Dabei wird immer dasselbe Verfahren angewendet: Alle Einträge in der ListView werden durchiteriert und das jeweilige SubItem wird dann mit dem Index als String zurückgegeben bzw. mit Clipboard.SetText(eintrag) aufgerufen. So können dann ausgewählte Einträge in die Zwischenablage kopiert werden, jedoch nur wenn Inhalt vorhanden ist. Dies wird mit string.IsNullOrEmpty(eintrag) überprüft.

### Einträge löschen

Auch hier werden wieder alle ausgewählten Einträge der ListView durchiteriert und dabei das gesamte item aus dem Objekt gelöscht.

### Einträge duplizieren

Hierbei wird ähnlich wie bei der bereits zuvor erwähnten Funktion AddNewEntry vorgegangen. Es werden zunächst lokale Variablen für die späteren hinzuzufügenden Einträge deklariert. Bevor die Variablen mit den Inhalt der ausgewählten Einträgen zugewiesen werden, müssen die Einträge innerhalb einer foreach-Schleife mit einer for-Schleife überprüft werden ob diese nicht leer sind, sodass keine ArgumentNullException vorkommt.

Anschließend wird ein ListViewItem erstellt, die jetzt befüllten Variablen werden nun zu den Subltems hinzugefügt und schließlich das ListViewItem zur ListView selbst hinzugefügt.

### Key-File

Damit eine Datenbank letztendlich auch wieder entschlüsselt werden kann, muss ein Key-File erstellt werden. Diese Datei ist nichts anderes, als der Rückgabewert der GenerateKey Funktion innerhalb der Enigma.Cryptography.Dll. Dieser ist in Base64 kodiert und wird zum Entschlüsseln als auch zum Verschlüsseln benötigt. Bevor eine Datenbank angelegt wird, muss zunächst definiert werden wo die Datei mit dem Schlüssel gespeichert werden soll. Aus diesem Grund wird ein Dialog erstellt zum Generieren solch einer Datei. Logischerweise muss es auch eine Funktion geben zum Öffnen einer bereits vorhandenen Key-Datei. Diese Methoden werden hier näher beschrieben.

### Erstellen

Bei der Erstellung der Key-Datei wird ein SaveFileDialog erstellt um den Pfad und Namen der Datei anzugeben. Als Dateiendung kann nur .enigma festgelegt werden, um die Key-File von der eigentlichen Datenbank zu unterscheiden. Der eigentliche Teil findet in der privaten SaveKeyFile Methode statt. Die Komplette Funktion ist in einem Try-Catch Block abgekapselt, sodass die UnauthorizedAccessException, welche nur auftaucht wenn versucht wird die Datei in einem Ort zu speichern wo der Benutzer keine Rechte hat, verarbeitet wird. Wird das Programm als Administrator gestartet, gestattet diese Funktion auch das Speichern in allen anderen Ordnern.

Der Key wird schließlich mithilfe der Cryptography Klasse generiert und mit einem StreamWriter in dem angegebenen Pfad erstellt. Anschließend wird der Speicherbereich des Schlüssels aus zuvor erwähnten Gründen aus dem Speicher mit ZeroMemory gelöscht bzw. mit Nullen allokiert. Die statische private Variable \_mKeySet wird auf true

gesetzt wenn der Key erfolgreich generiert wurde. Damit können nur Datenbanken erstellt werden, wenn auch bereits ein Key gesetzt bzw. gespeichert wurde.

### Laden

Um nun den zuvor generierten oder auch andere Keys zum Öffnen einer Datenbank zu laden, wird ein OpenFileDialog erstellt. Der Pfad des Schlüssels wird in der privaten statischen Variable `_mKeyPath` festgelegt. Dies ist praktisch, da der Anwender nach dem Auswählen der Datenbank den Pfad der Key-File angezeigt bekommt. Auch hier wird eine private Variable auf `true` gesetzt, damit andere Klassen wissen ob ein Key beim Öffnen von Datenbanken ausgewählt wurde.

### DatabaseHandler

Diese Klasse enthält nur öffentliche statische Methoden zur Verwaltung von Datenbanken. In der Klasse werden Datenbanken erstellt, geöffnet und geschlossen. Sie greift im Übrigen auf die XmlHandler und DialogManager Klasse über.

### Datenbank anlegen

Als Parameter wird hier die DialogManager Klasse genommen und die Form, sodass beim Erstellen eine ListView mit der Größe der Form entsteht.

### Datenbank öffnen

Hierfür wird ein zusätzlicher Parameter benötigt, ein OpenFileDialog Objekt. Die Datenbank wird dann über die XmlHandler Klasse entschlüsselt in die ListView geladen.

### Datenbank schließen

Damit das ListView Objekt auch wirklich vom Speicher geleert wird, muss die Methode Dispose aus der Form geladen werden. Davor wird jedoch noch die ListView aus dem Container der Form gelöscht.

### Implementierung

Die zuvor beschriebenen Klassen werden in der MainWindow Klasse, welche das Hauptfenster darstellen soll, implementiert. Dieses Fenster „wächst“ bzw. „schrumpft“ dann im Prinzip dynamisch zur Laufzeit des Programmes.

### Haupt-Fenster

In der Klassenform MainWindow ist nur eine MenuBar vorhanden, welche wiederum andere Fenster öffnen kann bzw. die ListView welche dynamisch von der DialogManager Klasse erstellt wird, in der Form abgelagert wird. In dem Konstruktor der Klasse wird ein DialogManager Objekt `_mDialog` angelegt. Sie ruft eine Methode zur Erstellung eines Begrüßungslabels auf und platziert dieses Label in die Mitte der Form. Somit ist auch

dieses Label dynamisch mit der Größe des Fensters abhängig, sodass es beim Vergrößern mittig positioniert wird.

Über die Menüleiste lässt sich eine Datenbank anlegen, öffnen, schließen, speichern, XML-Dokumente importieren und exportieren, sowie das Programm selbst schließen. Darüber hinaus können Einträge hinzugefügt, editiert, gelöscht, geklont und weitere Einträge kopiert werden. Zuletzt lässt sich noch eine Info zum Programm anzeigen. Die Unterpunkte der Menüleiste können auch mit Tastenkombinationen erreicht werden. Sobald die Form geladen wurde, wird die Konfigurationsdatei geladen wenn diese vorhanden ist. Beim Schließen der Form wird eine neue Konfigurationsdatei mit den Eigenschaften der Form angelegt.

Jeder Menüunterpunkt hat sein eigenes ClickEvent, welches aufgerufen wird wenn man die Unterpunkte anklickt. Bei der Erstellung einer neuen Datenbank wird die InitializeSetKeyFile Methode aus dem DialogManager aufgerufen. Nach dem Festlegen der Keyfile wird die ListView mit der Methode NewDatabase aus dem DatabaseHandler angelegt und das Begrüßungslabel verschwindet. Jetzt werden die Unterpunkte der Menüleiste zum Verwalten der Einträge aktiviert. Ob sich ein Menüpunkt anklicken lässt, wird über die Methode entriesMenuItem\_Select mithilfe des Item Counts festgestellt. Auch lässt sich jetzt die Datenbank speichern und schließen, sowie importieren und exportieren.

Beim Klicken auf dem „Open Database“ Unterpunkt wird zunächst der Anwender aufgefordert die „.edb“ Datenbank auszuwählen und dann die InitializeGetKeyFile Methode aufgerufen. Hierbei wählt man die Keyfile aus und dann wird die Datenbank geöffnet. Befindet sich bereits eine Datenbank in einer ListView und es wird versucht eine neue Datenbank zu öffnen, kann sich der Benutzer entscheiden ob er die derzeitige Datenbank schließen möchte. Dieses Haupt-Fenster lässt sich über dem Menüunterpunkt „Quit“ oder der Tastenkombination STRG + Q schließen.

### Einträge hinzufügen-Fenster

Dieses Fenster wird über den Menüunterpunkt „Add Entry“ aufgerufen. Das Fenster besteht aus einer PictureBox, sechs Labels, fünf Textboxen, eine RichEditBox und drei Buttons. Dort können die Einträge mit dem Button „OK“ hinzugefügt werden und das Passwort auch durch Platzhalter versteckt werden. Man kann Titel, Benutzername, Passwort, eine URL und auch Notizen hinzufügen. Nicht jedes Feld muss eingetragen werden.

### Einträge bearbeiten-Fenster

Dieses Fenster wird über den Menüunterpunkt „Edit/View Entry“ aufgerufen. Das Fenster ist genau gleich aufgebaut wie das vorherige. Der einzige Unterschied ist, dass die Einträge in die Textboxen eingetragen werden, sodass das Bearbeiten leichter fällt.

### Info-Fenster

In dem Fenster werden lediglich nur Informationen zur Lizenz angezeigt und andere diverse Informationen wie Versionsnummer und Github URL. Dieses Fenster kann über dem Menüunterpunkt „About“ aufgerufen werden.

## Dokumentation

Im Folgenden erläutere ich die statischen öffentlichen Methoden der Klassenbibliothek näher und deren Argumente, sodass man weiß wie man mit der Klasse umgehen soll.

<code>public static extern bool ZeroMemory(IntPtr destination, int length)</code>	Externe Methode aus der Kernel32.dll. destination steht für die Adresse des Objektes und length für die Länge.
<code>public static string GenerateKey()</code>	Generiert automatisch einen 8-byte Key (64-bit) und gibt ihn als Base64 String zurück.
<code>public static string AES_Encrypt(string sPlainText, string sKey)</code>	Verschlüsselt einen String sPlainText, welcher in einem <code>byte[]</code> Buffer umgewandelt wird. sKey entspricht dem Key der AES-256 Klasse.
<code>public static string AES_Decrypt(string sEncryptedData, string sKey)</code>	Entschlüsselt einen String sEncryptedData, welcher in einem <code>byte[]</code> Buffer umgewandelt wird. sKey entspricht dem Key der AES-256 Klasse.

Andere interessante Methoden der DialogManager werden hier erläutert.

<code>public static string AddNewPicture(Form window, Size imageSize, string imagePath)</code>	Erstellt eine neue <code>PictureBox</code> mit den Initialisierungsparametern <code>imageSize</code> und <code>imagePath</code> . Die <code>PictureBox</code> wird dann in den Container der <code>Form</code> hinzugefügt.
<code>public static bool CheckUrlValid(string urlInput)</code>	Überprüft eine URL als String <code>urlInput</code> und gibt mithilfe der Klasse <code>Uri</code> <code>true</code> zurück, wenn das Format von <code>urlInput</code> <code>Uri.UriSchemeHttp</code> oder <code>Uri.UriSchemeHttps</code> entspricht.
<code>public void FillListViewItemColors()</code>	Die <code>ListViewItems</code> einer <code>ListView</code> werden mit einer <code>foreach</code> -Schleife durchgangen und jedes zweite Item bekommt eine beigeartige Hintergrundfarbe.
<code>public void AddNewLabel(Form window, string caption)</code>	Erstellt ein neues <code>Label</code> mit den Initialisierungsparametern <code>window</code> und <code>caption</code> . Die Eigenschaft <code>TextAlign</code> ist <code>ContentAlignment.MiddleCenter</code> , <code>Dock</code> = <code>DockStyle.Fill</code> und <code>AutoSize</code> = <code>false</code> . Das <code>Label</code> wird dann in den Container der <code>Form</code> hinzugefügt.

## Fazit

Die Programmierung der Klassenbibliothek war relativ einfach, jedoch hat mir die DialogManager kleine bis große Probleme bereitet. Sie ist die aufwendigste Klasse im Programm gewesen und ich habe bei der Programmierung und beim Debuggen dieser Klasse am meisten meine Zeit verbracht. Deshalb habe ich mich auch anfangs entschieden diese Klasse als erstes zu erstellen, da das Handling (Handhaben) im Vordergrund steht und nicht die Verschlüsselung. Das Verwalten der Daten soll angenehm sein und einfach verständlich.

Dazu muss ich sagen, dass das Projekt nicht nach Zeitplan fertiggestellt wurde. Das liegt daran, dass das Design dynamisch konzipiert wurde und deshalb viele Änderungen aufgetreten sind. Die Arbeitszeit betrug inklusive Programmierung ca. 90 Stunden, d.h. ich würde ca. 1250 Euro für diese Projektarbeit bekommen, wenn dies bezahlt wäre. Insgesamt bin ich mit dem Resultat des Projektes zufrieden.

## Ausblick

Es sind einige versprochene Funktionen weggefallen und stattdessen durch andere oder neue ersetzt worden. Deshalb plane ich noch folgende Implementierungen im Programm vorzunehmen bzw. Fehler zu beheben.

- Benutzerdefinierbarer Master-Key in SHA-256 (8 Bytes)
- Keyfile überprüfen ([CryptographicException](#): Bad data beheben)
- DialogManager Klasse als Basis Klasse, einzelne Sub-Klassen für Objekte
- Password Generator

## Quellenverzeichnis

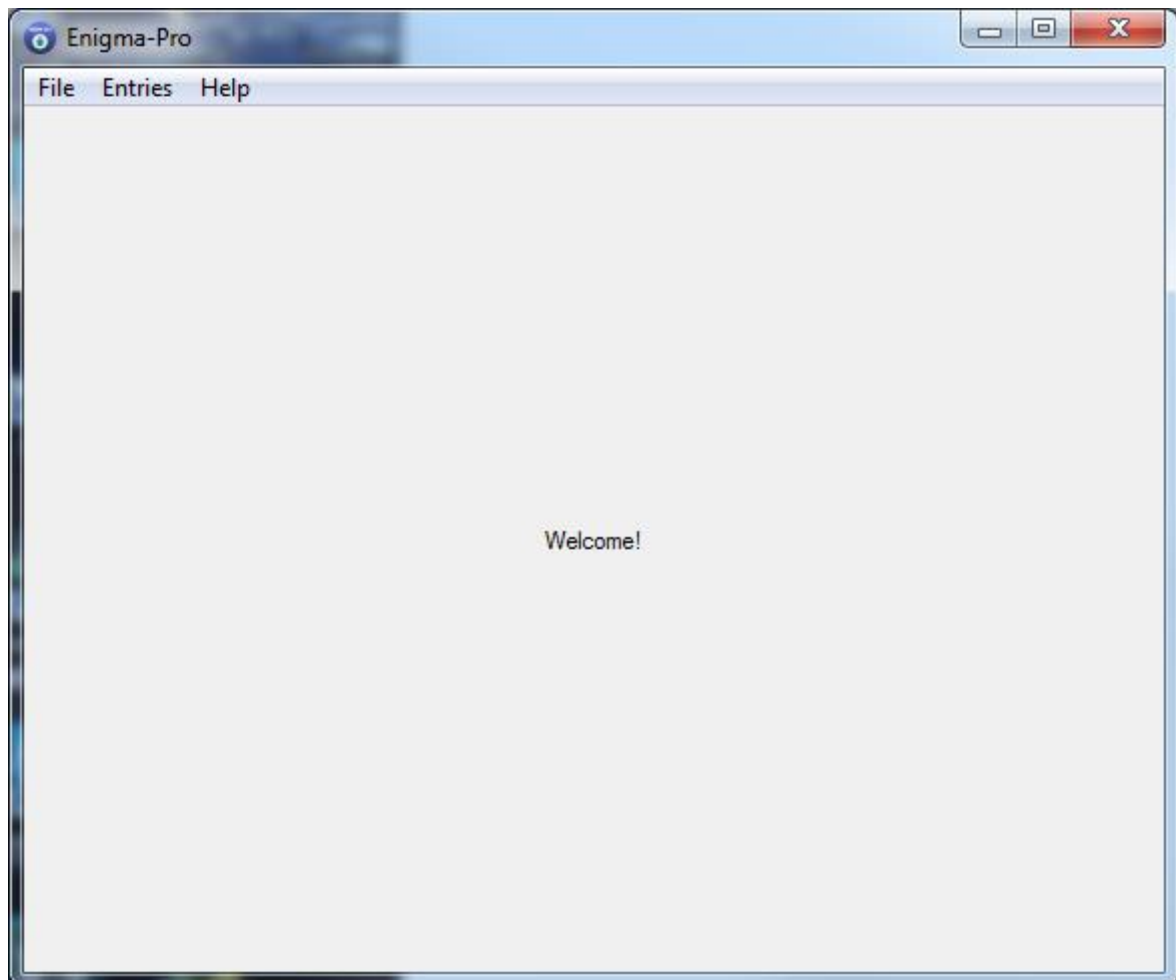
- 1) <http://darkblackwords.deviantart.com/art/Sword-Art-Online-Font-Download-426603647>, (Stand: 13.05.2016)
- 2) <https://support.microsoft.com/en-us/kb/319266>, (Stand: 02.02.2016)
- 3) <http://www.fatcow.com/free-icons>, (Stand: 02.05.2016)
- 4) [https://msdn.microsoft.com/en-us/library/system.security.cryptography.descryptoserviceprovider\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/system.security.cryptography.descryptoserviceprovider(v=vs.110).aspx), (Stand: 15.05.2016)
- 5) [https://msdn.microsoft.com/en-us/library/system.xml.xmlreader\(v=vs.95\).aspx](https://msdn.microsoft.com/en-us/library/system.xml.xmlreader(v=vs.95).aspx), (Stand: 02.02.2016)
- 6) [https://msdn.microsoft.com/en-us/library/system.xml.xmlwriter\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/system.xml.xmlwriter(v=vs.110).aspx), (Stand: 02.02.2016)
- 7) [https://msdn.microsoft.com/en-us/library/system.xml.linq.xdocument\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/system.xml.linq.xdocument(v=vs.110).aspx), (Stand: 15.02.2016)
- 8) <https://github.com/>, (Stand: 02.02.2016)



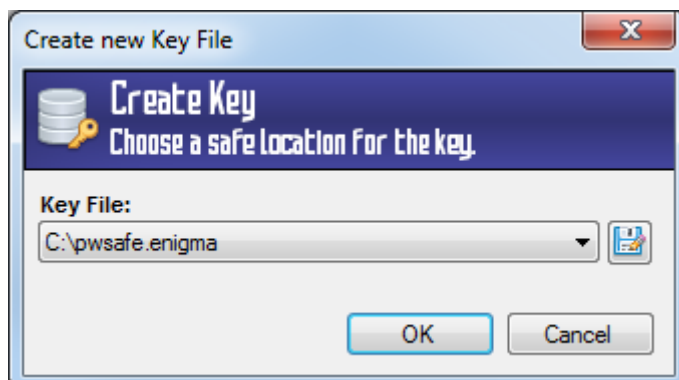
## Anhang

### Screenshots

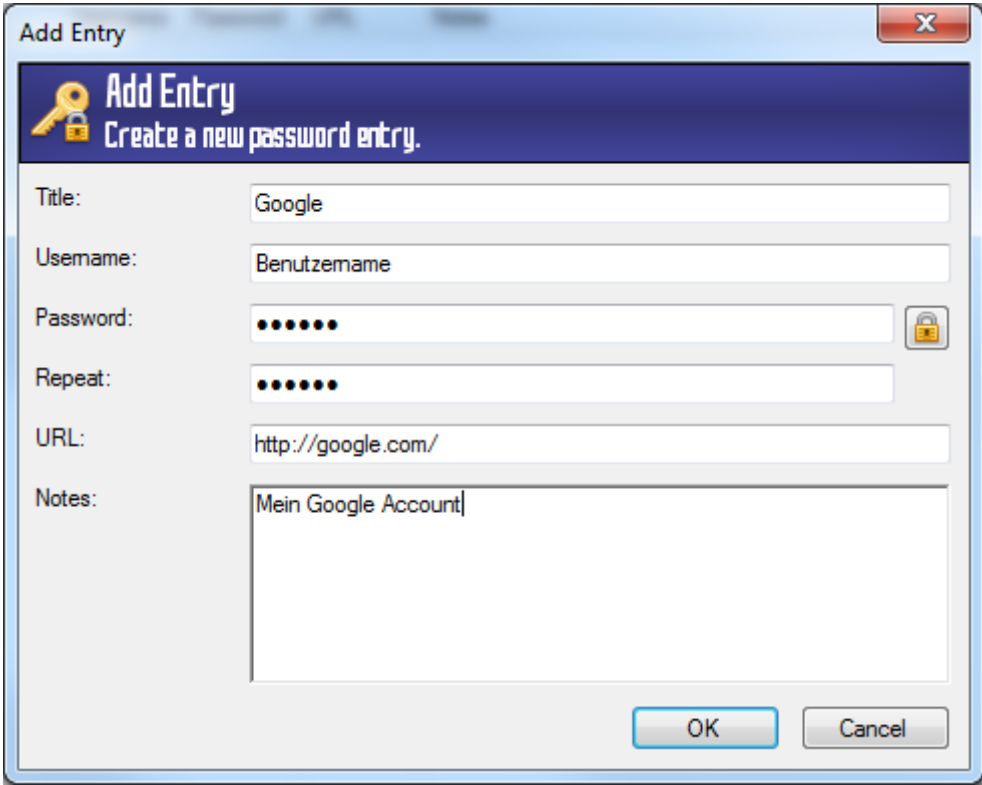
#### *Haupt-Fenster*



#### *Keyfile-Dialog*



### Einträge hinzufügen



The 'Add Entry' dialog box is used to create a new password entry. It features a title bar with a close button (X). The main area has a dark blue header with a key icon and the text 'Add Entry Create a new password entry.' Below this, there are several input fields: 'Title' (containing 'Google'), 'Username' (containing 'Benutzername'), 'Password' (masked with dots and a lock icon), 'Repeat' (masked with dots), 'URL' (containing 'http://google.com/'), and 'Notes' (containing 'Mein Google Account'). At the bottom right, there are 'OK' and 'Cancel' buttons.

**Add Entry**  
Create a new password entry.

Title: Google

Username: Benutzername

Password: •••••

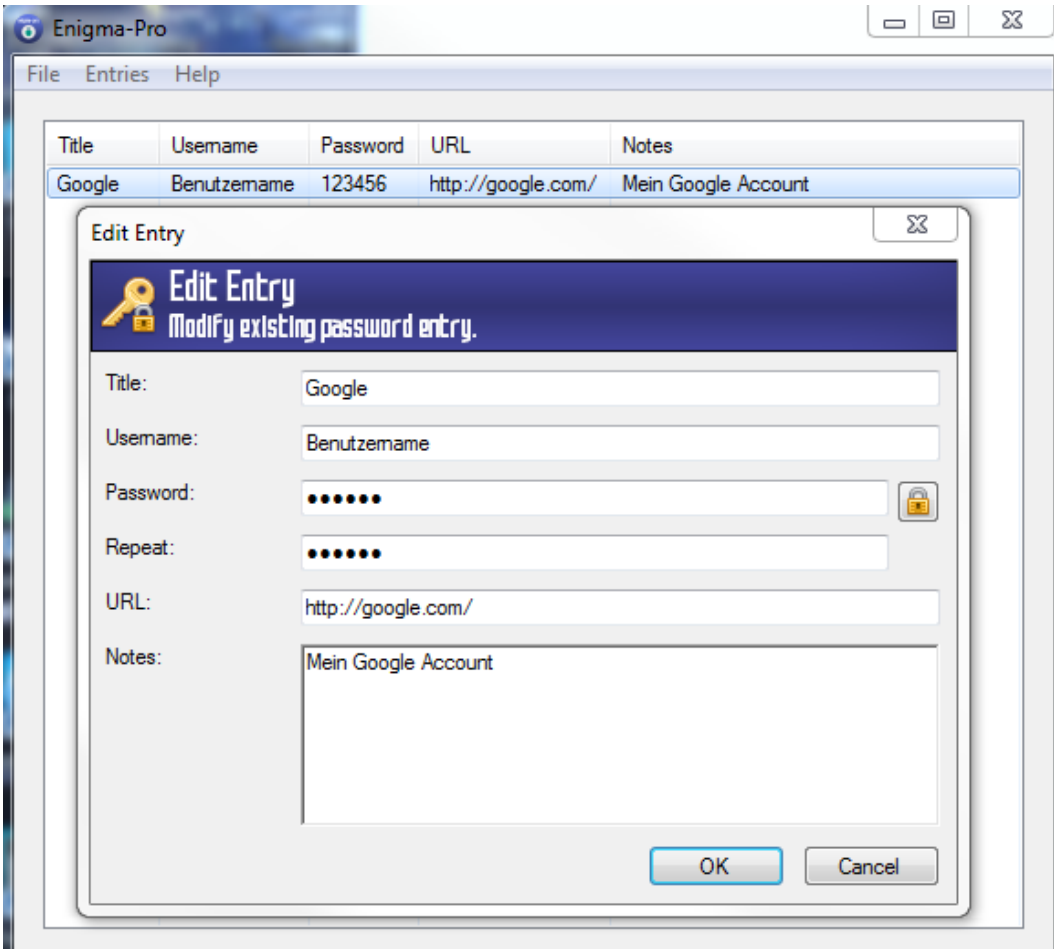
Repeat: •••••

URL: http://google.com/

Notes: Mein Google Account

OK Cancel

### Einträge bearbeiten:



The main window of 'Enigma-Pro' shows a table of entries. The 'Edit Entry' dialog box is open, allowing modification of an existing entry. The dialog has a title bar with a close button (X). The main area has a dark blue header with a key icon and the text 'Edit Entry Modify existing password entry.' Below this, there are several input fields: 'Title' (containing 'Google'), 'Username' (containing 'Benutzername'), 'Password' (masked with dots and a lock icon), 'Repeat' (masked with dots), 'URL' (containing 'http://google.com/'), and 'Notes' (containing 'Mein Google Account'). At the bottom right, there are 'OK' and 'Cancel' buttons.

**Enigma-Pro**

File Entries Help

Title	Username	Password	URL	Notes
Google	Benutzername	123456	http://google.com/	Mein Google Account

**Edit Entry**  
Modify existing password entry.

Title: Google

Username: Benutzername

Password: •••••

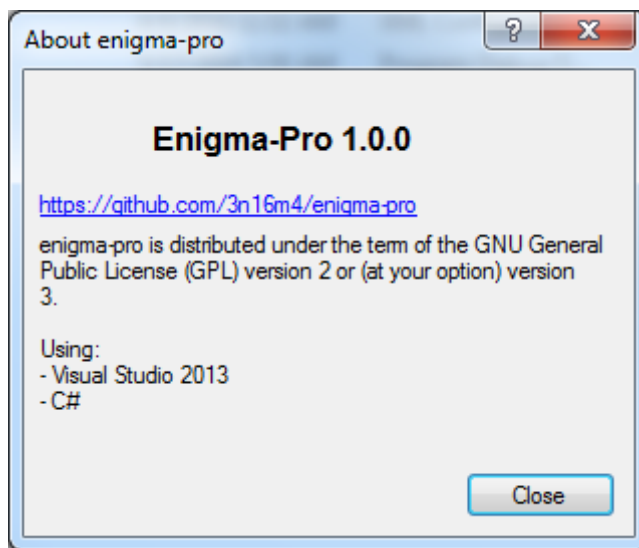
Repeat: •••••

URL: http://google.com/

Notes: Mein Google Account

OK Cancel

*Info-Fenster*



## Software

Die fertiggestellte Software befindet sich in der beiliegenden CD mit der Facharbeit als originales Dokument und als pdf. Datei.

## Eigenständigkeitserklärung

Hiermit bestätige ich, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Die Stellen der Arbeit, die dem Wortlaut oder dem Sinn nach anderen Werken (dazu zählen auch Internetquellen) entnommen sind, wurden unter Angabe der Quelle kenntlich gemacht.