

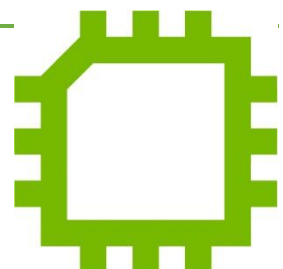
Hochschule für Technik und Wirtschaft Berlin

Belegarbeit Kleopatra

Modul: Software Engineering - 3. Fachsemester

Betreuer: Prof. Dr. Thomas Baar

Frank Schütze S0549557
Lukas Poeppel S0567279
10.6.2019



Inhaltsverzeichnis

1. Einführung	3
1.1 Kleopatra	3
2. Komplexitätsanalyse der Ausgangssoftware	3
2.1 Use-Case(s)	4
2.1.1 Generate Key Pair.....	5
Glossar Generate Key Pair.....	5
2.1.2 Export Public-Key	5
2.1.3 Export Private-Key.....	5
2.1.4 Import Public Key	5
2.1.5 Import Private Key.....	6
Glossar Import Public/Private Key	6
2.1.6 Sign Message/File.....	6
2.1.7 Encrypt Message/File	6
2.1.8 Verify Message/File	7
2.1.9 Decrypt Message/File.....	7
2.1.10 Generate Checksum	7
2.1.10 Verify Checksum.....	7
3. Programmstruktur	8
3.1 Änderungen	8
3.1.1 Anpassung Key-Darstellung	8
3.1.2 Struktur der Dateiausgabe	10
3.1.3 Änderung der Dateiausgabe	11
3.1.3 Option „Select Folder to Encrypt“	12
4. Herausforderungen und Herangehensweise.....	13
5. Fazit	14

Abbildung 1 Use-Case(s)	4
Abbildung 2 All Certificates	8
Abbildung 3 Foreign Certificates	8
Abbildung 4 Klassendiagramm keylists.....	9
Abbildung 5 Klasse ForeignCertificatesKeyFilter	9
Abbildung 6 Klassendiagramm Filesystem	10
Abbildung 7 Sequenzdiagramm Filesystem	10
Abbildung 8 Funktion createInfoFile.....	11
Abbildung 9 Beispiel Infodatei.....	11
Abbildung 10 Originalfunktion	12
Abbildung 11 Abgeänderte Funktion	12

1. Einführung

Im Rahmen des Pflichtmoduls *Softwaretechnik* wurde ein frei gewähltes open-source Computer Programm analysiert und geringfügig verändert. Dieses Projekt wurde während der Ausarbeitung von Prof. Dr. Thomas Baar betreut. Alle verwendeten Grafiken befinden sich im beiliegenden Ordner „Grafiken“.

1.1 Kleopatra

Das gewählte Programm Kleopatra der KDE e.V. steht für Anwender und Entwickler auf allen gängigen Plattformen kostenlos zur Verfügung. Der open-source Quelltext ist auf GitHub unter dem Link <https://github.com/KDE/kleopatra/tree/Applications/17.12> frei verfügbar.

Kleopatra ist eine C++ front-end GUI Implementation für eine Standard GnuPG Verschlüsselung. GnuPG ist eine open-source Implementierung des OpenPGP Standards welcher 1997 mit dem Dokument <https://www.ietf.org/rfc/rfc4880.txt> weltweit einheitlich definiert wurde. GnuPG bietet die Möglichkeit, Nachrichten bzw. Dateien zu verschlüsseln und zu signieren. Grundlage für diese Verschlüsselung ist ein umfangreiches Private/Public-Key-Management System.

Eine Datei, welche mit einem 128-Bit PGP Public Key verschlüsselt wurde, ist ohne den dazugehörigen PGP Privat Key so gut wie nicht zu entschlüsseln. Um alle möglichen 2^{128} Schlüssel zu generieren, würde der aktuelle Stand der Technik ca. 10^{13} ^[1] Jahre brauchen. Vertrauliche Nachrichten werden mittlerweile mit bis zu 4069 Bit verschlüsselt.

GnuPG ist ein reines Kommandozeilenprogramm und bereitet vielen Anwendern Probleme in der Handhabung. Kleopatra stellt eine grafische Oberfläche für diesen Standard bereit und erleichtern somit die Nutzung dieser Verschlüsselungsmethode.

2. Komplexitätsanalyse der Ausgangssoftware

Kleopatra ist ein sehr vielseitiges Programm. Es bietet die Möglichkeit Nachrichten zu verschlüsselt/entschlüsselt und Prüfsummen nach verschiedenen Standards zu erstellen und auszuwerten. Außerdem bietet es die Möglichkeit der Schlüsselgenerierung und des Importieren vorhandener Schlüssel.

Kleopatra ist in verschiedene Bibliotheken aufgebaut. Kernstück bildet *GpgME* und *Gpg++* sowie *MIME* und *Qt*, worauf Kleopatra aufsetzt. Zusätzlich entwickelt KDE weitere Software und bedient sich dabei eigener, wiederverwendbarer Bibliotheken; darunter *KF5*, *Libkleo* und weitere.

Die umfangreichen Anwendungsmöglichkeiten sind im folgenden Use-Case Diagramm dargestellt.

¹ Quelle <http://www.gpg.net/pgpnet/pgp-faq/pgp-faq-security-questions.html#security-against-brute-force>
02.06.2019

2.1 Use-Case(s)

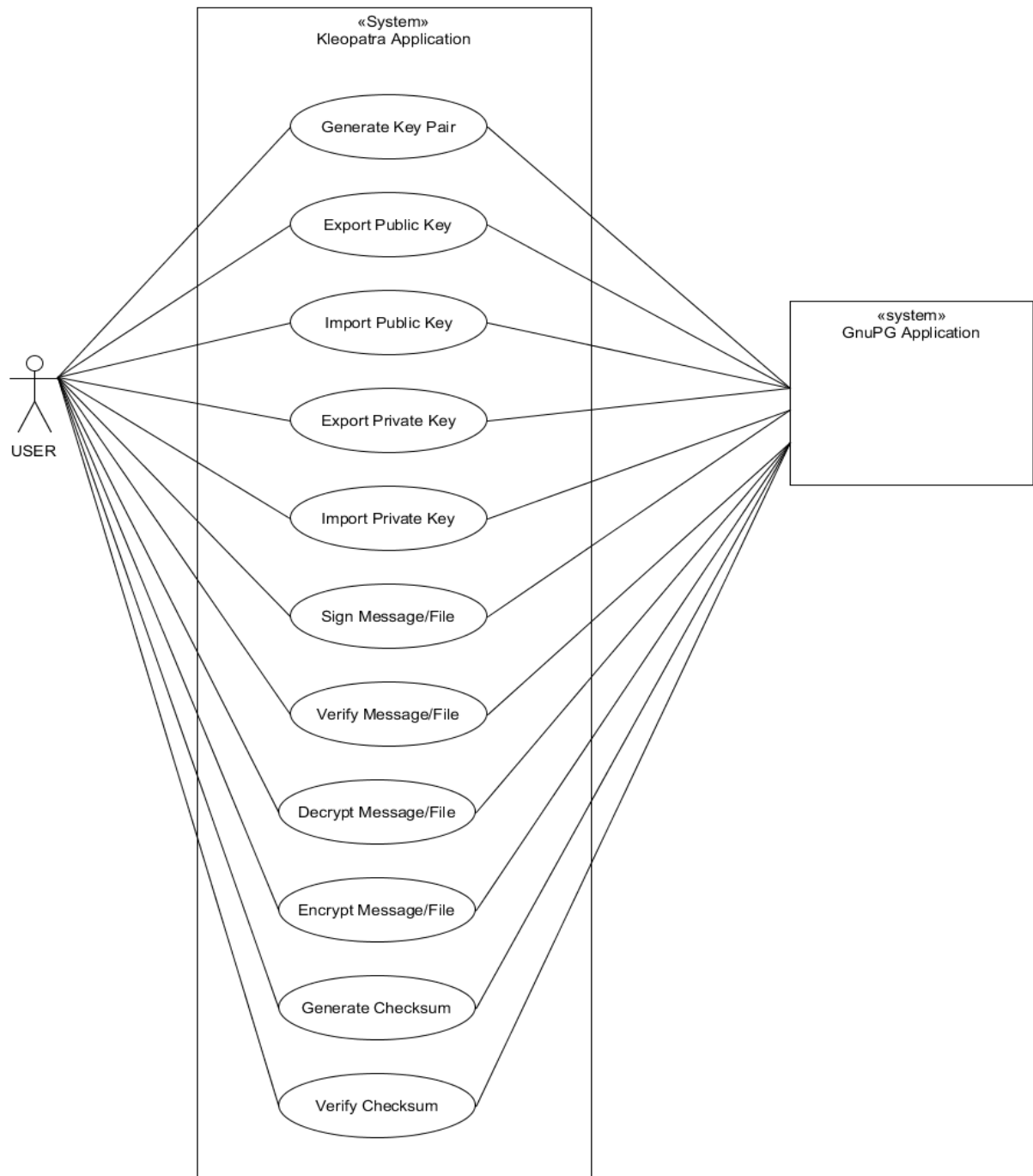


Abbildung 1 Use-Case(s)

Kleopatra stellt die Schnittstelle zwischen Anwender und kryptischer Verschlüsselung durch den GnuPG Standard bereit.

Im Folgenden befindet sich die Ausformulierung der einzelnen Operationen als Happy Day Szenario.

2.1.1 Generate Key Pair

1. **User** wählt Option „Neues Schlüsselpaar“ und gibt *Details* des zu erstellenden Schlüssels an.
2. **System** validiert Eingabe und zeigt eine Zusammenfassung der gesammelten Informationen
3. **User** validiert die angezeigten Daten und erstellt ein persönliches Passwort
4. **System** gibt Auftrag an **GnuPG application**
5. **GnuPG application** generiert Schlüsselpaar und übermittelt dieses an **System**
6. **System** gibt Meldung über erfolgreiche Schlüsselpaar Generierung aus und zeigt eine Zusammenfassung des Auftrages
7. **User** bestätigt die Zusammenfassung
8. **System** fügt Schlüsselpaar in die Übersicht „vorhandene Zertifikate“ hinzu
9. Das Use Case endet erfolgreich

Glossar Generate Key Pair

Details:

Die *Details* beinhalten Name, E-Mailadresse (optional), Verschiedene Verschlüsselungsarten (2048 Bit / 3072 Bit / 4096 Bit), Gültigkeitsdauer und verschiedene Verwendungsmöglichkeiten.

2.1.2 Export Public-Key

1. **User** selektiert einen vorhandenen Schlüssel
2. **User** wählt Option „Exportieren“ und gibt einen Speicherpfad an
3. **System** fordert Public Key von **GnuPG application** und speichert Public Key als file (*.asc) im ausgewählten Pfad
4. Das Use Case endet erfolgreich

2.1.3 Export Private-Key

1. **User selektiert einen vorhandenen Schlüssel**
2. **User** wählt Option „Geheimen Schlüssel Exportieren“ und gibt einen Speicherpfad an
3. **System** fordert Private Key von **GnuPG application** und speichert Public Key als file (*.asc) im ausgewählten Pfad
4. Das Use Case endet erfolgreich

2.1.4 Import Public Key

1. **User** wählt Option „Importieren“ und selektiert eine Datei aus unterstützen Dateiformat
2. **System** übermittelt Public Key an **GnuPG application**
3. **GnuPG application** validiert Schlüssel
4. **System** zeigt detaillierte Ergebnisse des Schlüsselimportes an
5. **User** bestätigt die Ergebnisse
6. **System** fügt Schlüssel in die Übersicht „vorhandenen Zertifikate“ hinzu
7. Das Use Case endet erfolgreich

2.1.5 Import Private Key

1. **User** wählt Option „Importieren“ und selektiert eine Datei aus unterstützten Dateiformat
2. **System** übermittelt Public Key an **GnuPG application**
3. **GnuPG application** validiert Schlüssel
4. **User** gibt persönliches Passwort des Schlüsselpaares ein
5. **System** validiert das Passwort zeigt detaillierte Ergebnisse des Schlüsselimportes an
6. **User** bestätigt die Ergebnisse
7. **System** fügt Schlüssel in die Übersicht „vorhandenen Zertifikate“ hinzu
8. Das Use Case endet erfolgreich

Glossar Import Public/Private Key

Dateiformat:

Unterstützte Dateiformate = *.asc ; *.cer ; *.cert ; *.crt ; *.der ; *.pmg ; *.gpg ; *.p7c ; *.p12 ; *.pfx ; *.pgp

2.1.6 Sign Message/File

1. **User** wählt Option „Signieren/Verschlüsseln“ und selektiert eine Datei
2. **System** validiert Datei zeigt vorhandene Schlüsselpaare zur Signierung
3. **User** wählt ein vorhandenes Schlüsselpaar, ein Speicherpfad und bestätigt den Vorgang mit einer Passworтеingabe
4. **System** gibt Auftrag an **GnuPG application** weiter
5. **GnuPG application** validiert Passwort und speichert selektierte Datei + Signierung in einer neuen Datei
6. **System** zeigt detaillierte Ergebnisse der Signierung an
7. **User** bestätigt die Ergebnisse
8. Das Use Case endet erfolgreich

2.1.7 Encrypt Message/File

1. **User** wählt Option „Signieren/Verschlüsseln“ und selektiert eine Datei
2. **System** validiert Datei zeigt vorhandene Schlüsselpaare zur Verschlüsselung
3. **User** wählt ein vorhandenes Schlüsselpaar, ein Speicherpfad und bestätigt den Vorgang mit einer Passworтеingabe
4. **System** gibt Auftrag an **GnuPG application** weiter
5. **GnuPG application** validiert Passwort und verschlüsselt + speichert selektierte Datei in einer neuen Datei
6. **System** zeigt detaillierte Ergebnisse der Verschlüsselung an
7. **User** bestätigt die Ergebnisse
8. Das Use Case endet erfolgreich

2.1.8 Verify Message/File

1. **User** wählt Option „Entschlüsseln/Überprüfen“ und selektiert eine Datei
2. **System** gibt Auftrag an **GnuPG application** weiter
3. **GnuPG** validiert die Datei unter berücksichtig der vorhandenen Schlüssel
4. **System** und zeigt detaillierte Ergebnisse der Verifizierung
5. **User** bestätigt die Ergebnisse
6. Das Use Case endet erfolgreich

2.1.9 Decrypt Message/File

1. **User** wählt Option „Entschlüsseln/Überprüfen“ und selektiert eine Datei
2. **System** gibt Auftrag an **GnuPG application** weiter
3. **GnuPG** validiert die Datei unter berücksichtig der vorhandenen Schlüssel
4. **User** gibt Passwort des zugehörigen Schlüsselpaars ein
5. **GnuPG** validiert Passwort entschlüsselt und speichert selektierte Datei als neue Datei
6. **System** und zeigt detaillierte Ergebnisse Entschlüsselung
7. **User** bestätigt die Ergebnisse
8. Das Use Case endet erfolgreich

2.1.10 Generate Checksum

1. **User** wählt Option „Prüfsumme erstellen“ und selektiert eine Datei
2. **System** gibt Auftrag an **GnuPG application** weiter
3. **GnuPG** erstellt Prüfsumme
4. **System** speichert Prüfsumme in neuer Datei zeigt detaillierte Ergebnisse Entschlüsselung
5. **User** bestätigt die Ergebnisse
6. Das Use Case endet erfolgreich

2.1.10 Verify Checksum

1. **User** wählt Option „Prüfsumme verifizieren“ und selektiert eine Datei
2. **System** gibt Auftrag an **GnuPG application** weiter
3. **GnuPG** validiert Prüfsumme
4. **System** zeigt detaillierte Ergebnisse der Prüfung
5. **User** bestätigt die Ergebnisse
6. Das Use Case endet erfolgreich

3. Programmstruktur

Kleopatra besteht aus verschiedenen Bibliotheken mit insgesamt 327 einzelnen Klassen. Ein Großteil ist hierbei auf die Bibliothek *Qapplications*, *GpgME*, *Libkleo* und *Kleopatra* aufgebaut.

Qapplications ist eine ständig wachsende Entwicklungsumgebung und stellt das komplette GUI-Framework zur Verfügung ^[2]. *GpgME* stellt den eigentlichen Verschlüsselungspart und *Libkleo* sowie *Kleopatra* sind entwickelt von KDE selbst.

Das Scope des Projekts und dieser Ausarbeitung beschränkt sich auf die vorgesehen Änderungen bestimmter Klassen, die in den Haupt-Bibliotheken *Kleopatra* und *Libkleo* angesiedelt sind.

3.1 Änderungen

Ein wichtiger Bestandteil der Projektaufgabe war es, geringfügige Änderungen am Programmablauf/-verhalten vorzunehmen. Es wurde sich darauf geeinigt, die Darstellung der vorhandenen Schlüssel sowie die Dateiausgabe zu verändern.

3.1.1 Anpassung Key-Darstellung

Für die Anpassung der Darstellung sollten die Schlüssel zusätzlich zur Auflistung aller Keys in eigene Private-Keys und fremde, importierte Public-Keys differenziert werden können. Die Funktion für eigene Schlüssel gab es bereits, somit musste das Programm noch um die Fremd-Schlüssel Anzeige erweitert werden.



Abbildung 2 All Certificates



Abbildung 3 Foreign Certificates

² Weitere Informationen: <https://www.qt.io/>

```

classDiagram
    class Libkleo {
        namespace /src/models/
        class KeyListModel {
            <<abstract>>
        }
        class KeyListModelInterface
        class KeyListSortProxyModel {
            <<abstract>>
        }
        namespace /src/kleo/
        class KeyFilter {
            MatchContext: enum
        }
        class DefaultKeysFilter {
            mName: string
            mHasSecret: ?
            setName(string): void
            setSpecificity(int): void
            setId(string): void
        }
        class KeyFilterManager {
            <<interface>>
            class CertificatesKeysFilter {
                ?CertificatesKeysFilter
                setHasSecret(?): void
            }
        }
    }

    class Kleopatra {
        namespace /src/
        class Mainwindow
        namespace /src/view/
        class TabWidget {
            addView(*Page): void
            setKeyFilter(*KeyFilter): void
            setSourceColumns(vector<int>): void
        }
        class Page
        class KeyTreeView {
            m_view: *QTreeView
            m_keyfilter: *KeyFilter
        }
    }

    KeyListModel <|-- KeyListModelInterface
    KeyListSortProxyModel <|-- KeyListModelInterface
    KeyFilter <|-- DefaultKeysFilter
    KeyFilterManager <|-- CertificatesKeysFilter

    KeyListModelInterface ..> KeyListSortProxyModel : <<setSourceColumn>>
    KeyFilterManager ..> CertificatesKeysFilter : <<setKeyFilter>>
    KeyFilterManager ..> KeyFilter : <<gets key information from>>

    Mainwindow "1" --> "1" TabWidget
    TabWidget "1" --> "1..n" Page : shows
    Page "1" o--> "1" KeyTreeView : sorts by
  
```

The diagram illustrates the architecture of Libkleo and Kleopatra. Libkleo contains the core interfaces and abstract classes, while Kleopatra implements these in the GUI. The diagram shows dependencies, inheritance, and associations between components like KeyListModelInterface, KeyListSortProxyModel, KeyFilter, Page, and KeyTreeView.

Libkleo Package Structure:

- /src/models/**
 - «abstract» KeyListModel**
 - KeyListModelInterface** (Inherits from KeyListModel)
 - «abstract» KeyListSortProxyModel** (Inherits from KeyListModelInterface)
- /src/kleo/**
 - «abstract» KeyFilter** (MatchContext: enum)
 - DefaultKeysFilter** (Inherits from KeyFilter)
 - mName: string
 - mHasSecret: ?
 - setName(string): void
 - setSpecificity(int): void
 - setId(string): void
 - KeyFilterManager** (Interface for front-end for current display)
 - ?CertificatesKeysFilter** (Inherits from KeyFilterManager)
 - setHasSecret(?): void

Kleopatra Package Structure:

- /src/**
 - Mainwindow**
- /src/view/**
 - TabWidget**
 - addView(*Page): void
 - setKeyFilter(*KeyFilter): void
 - setSourceColumns(vector<int>): void
 - Page**
 - sorts by ▶ (Association with KeyTreeView)
 - KeyTreeView**
 - m_view: *QTreeView
 - m_keyfilter: *KeyFilter

Relationships:

- KeyListModelInterface** (Libkleo) has a dashed association with **KeyListSortProxyModel** (Libkleo) labeled **«setSourceColumn»**.
- KeyFilterManager** (Libkleo) has a dashed association with **?CertificatesKeysFilter** (Libkleo) labeled **«setKeyFilter»**.
- KeyFilterManager** (Libkleo) has a dashed association with **KeyFilter** (Libkleo) labeled **«gets key information from»**.
- Mainwindow** (Kleopatra) has a solid association with **TabWidget** (Kleopatra) with multiplicity 1 at Mainwindow and 1 at TabWidget.
- TabWidget** (Kleopatra) has a solid association with **Page** (Kleopatra) labeled **shows** with multiplicity 1 at TabWidget and 1..n at Page.
- Page** (Kleopatra) has a solid association with **KeyTreeView** (Kleopatra) labeled **sorts by** with multiplicity 1 at Page and 1 at KeyTreeView.

Für die Anpassung wurde im *KeyFilterManager* Container der Bibliothek „Libkleo“ eine eigene Klasse ***ForeignCertificatesKeyFilter***, abgeleitet von *DefaultKeyFilter*, erstellt und der entsprechende Parameter zur Filterung *hasSecret* gesetzt:

Abbildung 5 Klasse ForeignCertificatesKeyFilter

Seite | 9

3.1.2 Struktur der Dateiausgabe

Als weiterer Teil der Projektaufgabe sollte die Dateiausgabe verändert werden. Anforderung war es, den Dateinamen so zu verändern, dass aus ihm ersichtlich wird, mit welchem Schlüssel die Datei signiert wurde und für welchen Empfänger diese Verschlüsselt worden ist.

Für die Umsetzung der Teilaufgabe wurde eine Klassen- und Sequenzdiagramm erstellt, um einen Überblick über die Dateiausgabe zu bekommen:

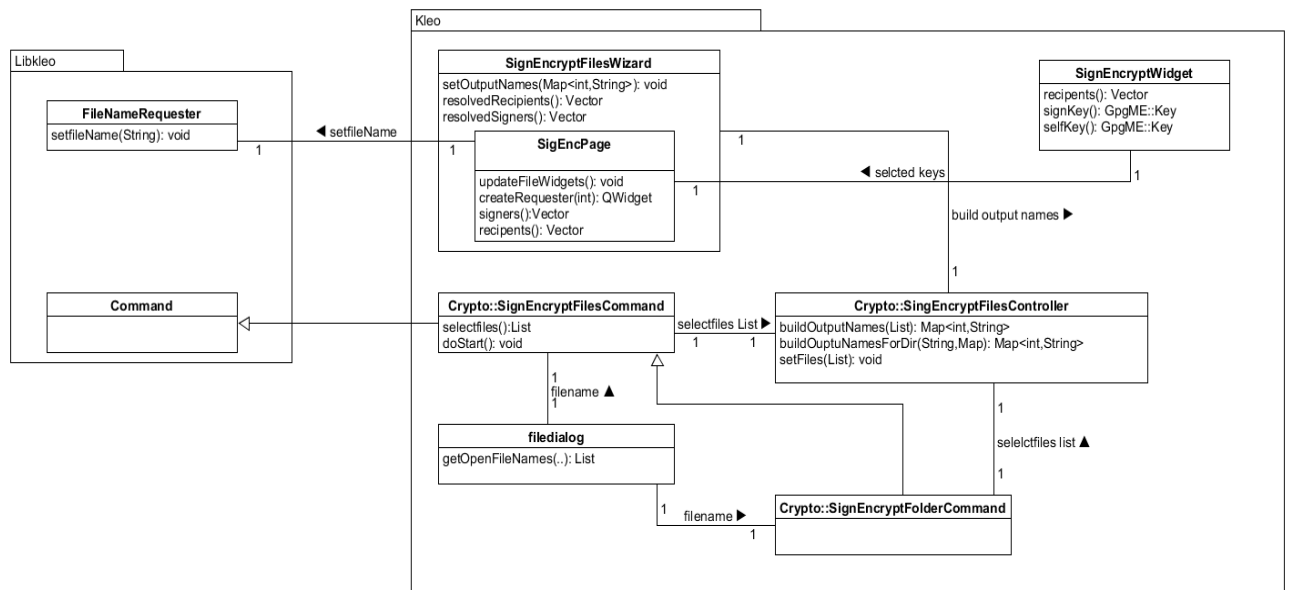


Abbildung 6 Klassendiagramm Filesystem

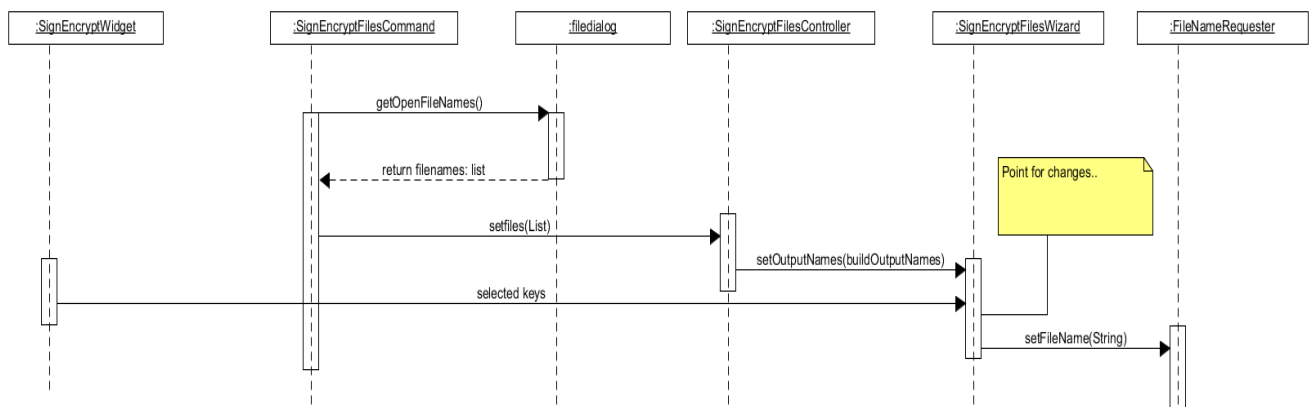


Abbildung 7 Sequenzdiagramm Filesystem

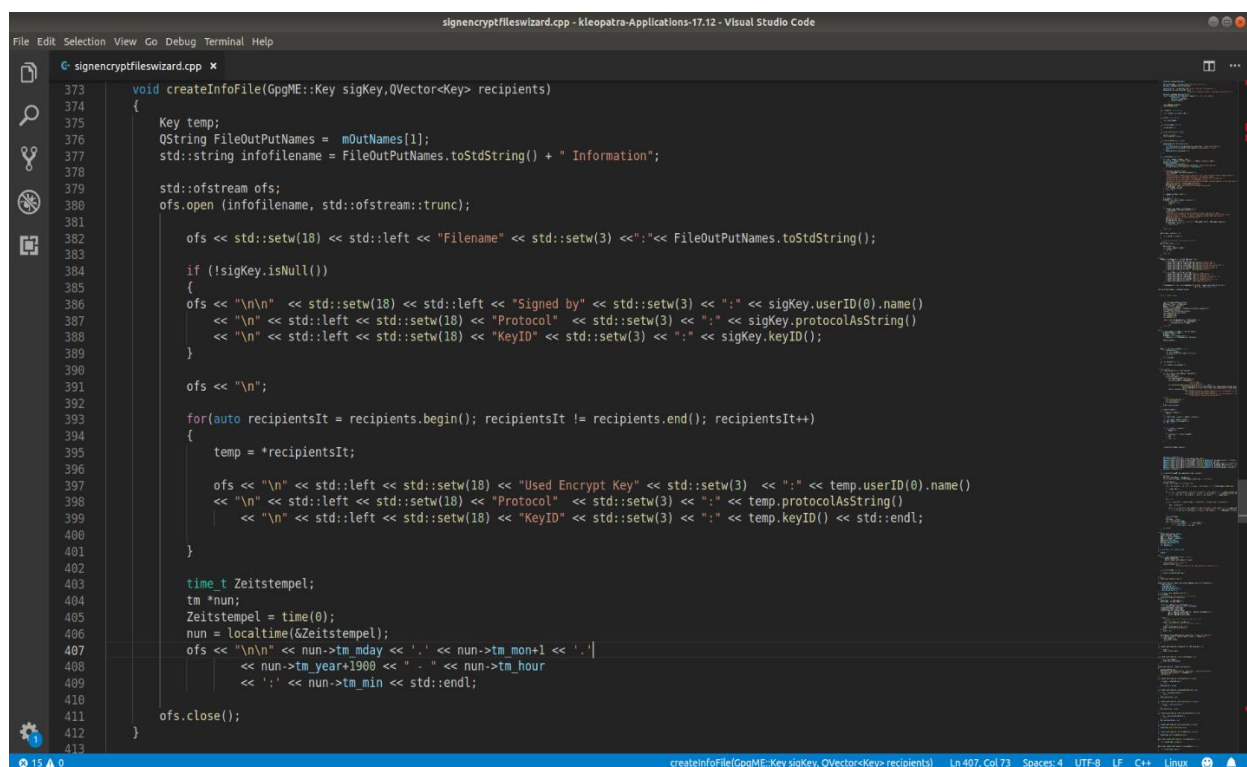
3.1.3 Änderung der Dateiausgabe

Ein geeigneter Punkt für die Änderung ist die Klasse **SignEncryptFilesWizard**, eine Unterklasse von **SigEncPage**, da dort die selektierten Keys sowie die erzeugten Dateinamen zusammenkommen.

Da bei der Verschlüsselung unendlich viele Empfänger ausgewählt werden können und die Option besteht, die Datei zusätzlich zu signieren, würde der erzeugte Dateiname unter bestimmten Umständen sehr lang und unübersichtlich werden. Es wurde sich dafür entschieden, eine zweite Datei zu erzeugen, welche alle wichtigen Informationen beinhaltet. Die erzeugte Datei wird im selben Verzeichnis als „gewaehlterDateiname.information“ abgelegt.

Es wurde eine Funktion **createInfoFile(sigKey, recipients)** erstellt, welche innerhalb der Funktion **updateFileWidgets()** aufgerufen wird. Somit wird sichergestellt, dass diese Funktion immer mit dem aktuell selektierten Key versorgt wird.

Im Folgenden befindet sich die erstellte Funktion:



```
373 void createInfoFile(GpgME::Key sigKey, QVector<Key> recipients)
374 {
375     Key temp;
376     QString FileOutPutNames = mOutNames[1];
377     std::string infofilename = FileOutPutNames.toStdString() + " Information";
378
379     std::ofstream ofs;
380     ofs.open (infofilename, std::ofstream::trunc);
381
382     ofs << std::setw(18) << std::left << "Filename" << std::setw(3) << ":" << FileOutPutNames.toStdString();
383
384     if (!sigKey.isNull())
385     {
386         ofs << "\n\n" << std::setw(18) << std::left << "Signed by" << std::setw(3) << ":" << sigKey.userID(0).name()
387         << "\n" << std::left << std::setw(18) << "Protocol" << std::setw(3) << ":" << sigKey.protocolAsString()
388         << "\n" << std::left << std::setw(18) << "KeyID" << std::setw(3) << ":" << sigKey.keyID();
389     }
390
391     ofs << "\n";
392
393     for(auto recipientsIt = recipients.begin(); recipientsIt != recipients.end(); recipientsIt++)
394     {
395         temp = *recipientsIt;
396
397         ofs << "\n" << std::setw(18) << "Used Encrypt Key" << std::setw(3) << ":" << temp.userID(0).name()
398         << "\n" << std::left << std::setw(18) << "Protocol" << std::setw(3) << ":" << temp.protocolAsString()
399         << "\n" << std::left << std::setw(18) << "KeyID" << std::setw(3) << ":" << temp.keyID() << std::endl;
400     }
401
402     time_t Zeitstempel;
403     tm *nun;
404     Zeitstempel = time(0);
405     nun = localtime(&Zeitstempel);
406     ofs << "\n\n" << nun->tm_mday << "." << nun->tm_mon+1 << "." <<
407     << nun->tm_year+1900 << "-" << nun->tm_hour
408     << ":" << nun->tm_min << std::endl;
409
410     ofs.close();
411 }
412
413
```

Abbildung 8 Funktion createInfoFile

Datei	Bearbeiten	Format	Ansicht	Hilfe
Filename	:	/home/frank/kalender.gpg		
Signed by	:	frankaoem		
Protocol	:	OpenPGP		
KeyID	:	8DE2C60A37B0C2FC		
Used Encrypt Key	:	frankaoem		
Protocol	:	OpenPGP		
KeyID	:	8DE2C60A37B0C2FC		
6.6.2019 - 22:52				

Abbildung 9 Beispiel Infodatei

3.1.3 Option „Select Folder to Encrypt“

Eine weitere Aufgabenstellung war es, eine Option zum Implementieren, welche die Selektion eines kompletten Ordners zur Verschlüsselung/Signierung ermöglicht. Während der Analyse des Programmcodes wurde festgestellt, dass es diese Option bereits gab und dafür eine Klasse *SignEncryptFolderCommand* existierte. Diese Klasse wurde bereits in Abbildung 6 dargestellt.

SignEncryptFolderCommand ist eine Ableitung der Klasse *SignEncryptFilesCommand* und hat denselben sequenziellen Ablauf wie diese.

Der Aufruf eines Objektes dieser Klasse im Programmменю „*Sign/Encrypt Folder...*“ ruft regulär einen Programmabsturz hervor.

Obwohl kein syntaktischer Fehler im Programmcodes vorhanden ist, wurde die Funktion abgeändert und der Aufruf brachte keine Probleme mehr. Im abgeänderten Programm steht die Funktion „***Sign/Encrypt Folder...***“ ohne Einschränkungen zur Verfügung, zusätzlich wird die Info-File für diese Option generiert.

```
QStringList SignEncryptFolderCommand::selectFiles() const
{
    const QString dir = QFileDialog::getExistingDirectory(qApp->activeWindow(),
                                                         i18n("Select Folder to Sign and/or Encrypt"));
    if (dir.isNull()) {
        return QStringList();
    }
    return QStringList() << dir;
}
```

Abbildung 10 Originalfunktion

```
QStringList SignEncryptFolderCommand::selectFiles() const
{
    const QString dir = QFileDialog::getExistingDirectory(qApp->activeWindow(),
                                                         i18n("Select Folder to Sign and/or Encrypt"));
    if (dir.isNull()) {
        return QStringList();
    }

    QStringList liste;
    liste << dir;
    return liste;
}
```

Abbildung 11 Abgeänderte Funktion

4. Herausforderungen und Herangehensweise

Die Historie der Terminal-Befehle zur Installation des Programms samt aller Bibliotheken findet sich im Anhang. Dies dokumentiert sehr akkurat die Installation, da nichts über eine grafische Oberfläche installiert wurde, ausgenommen der Downloads für GpgME, jedoch wurden auch diese via Terminal installiert.

Als erstes Projekt dieser Art war die größte Herausforderung die mangelnde Erfahrung. Das gesamte Programm „Kleopatra“ mit seinen verschiedenen Bibliotheken ging weit über die Vorgabe hinaus. Um dies auszugleichen, wurde sich entsprechend darauf geeinigt, sich auf eine bestimmte Bibliothek („Kleopatra“, die das eigentliche Kernstück darstellt) zu beschränken.

Nachdem die Zielstellung definiert war, stellte sich jedoch schnell heraus, dass Kleopatra als Programm nicht wie angenommen aufgebaut und strukturiert war. Somit erweiterte sich die Aufarbeitung der Programmstruktur enorm, da nun auch andere Bibliotheken analysiert und verstanden werden mussten – so wurden aus geplanten 20 plötzlich 140 Klassen mit 200-300 Zeilen Code im Durchschnitt.

Dass der Code insgesamt schlecht bis gar nicht kommentiert und dokumentiert ist, war keine Hilfe (insbesondere Variabel-Namen wie „magic“ oder Kommentare wie „todo: adjust“ waren nicht hilfreich).

Auch die Installation des Programms erwies sich als Hürde. Während es vorgepackte Programmversionen für MacOS, die gängigen Linux-Distributionen und Windows gibt, wird für Entwickler der Github-Download mit der Installation über cmake als Installation vorgegeben. Dafür mussten jedoch viele fehlende Pakete und Bibliotheken, die zwar im vorgepackten Programm, jedoch nicht im Git enthalten waren, über andere Quellen heruntergeladen und installiert werden. Dabei die entsprechend kohärenten und funktionierenden Versionen zu finden war sehr mühsam.

Als dann im Verlauf der eigentlichen Anpassung offensichtlich wurde, dass auch andere Bibliotheken als das Kernstück „Kleopatra“ verändert werden mussten, stellte sich schnell ein noch größeres Problem ein – die zusätzlich geladenen Pakete und so durchgeführte Installation verhinderte die Anpassung der anderen Bibliotheken.

So begann ein neues Suchen der passenden Pakete, das noch mühsamer und umfangreicher war als das Ursprüngliche. Pakete, die früher mal abänderbar waren und jetzt nicht mehr bzw. der umgekehrte Fall mussten jetzt zusätzlich mit anderen Paketen auf Kompatibilität geprüft werden. Einige Bündel an Paketen, die als Ganzes heruntergeladen werden konnten, mussten nun jedoch als einzelne Pakete in verschiedenen Versionen heruntergeladen werden.

Nachdem all dies erfolgreich abgeschlossen war, begann das Aufarbeiten der nun anderen Programm- und Klassenstruktur. Einige Klassen waren nun in anderen Ordnern oder sogar Bibliotheken gegenüber der ursprünglich genutzten Installation. Dies machte insgesamt das Ausarbeiten eines Klassendiagramms sehr aufwendig, allerdings half es auch insofern, dass man tatsächlich auf ein Klassendiagramm angewiesen war, um die verschiedenen Abhängigkeiten im Überblick zu halten.

Sourcetrail und Doxygen erwiesen sich am Anfang als gute Tools, um einen ersten Einblick in Aufbau und Struktur zu erhalten. Leider waren die Abbildungen nicht vollständig, die Sprünge zwischen den

Bibliotheken gingen verloren. Schnell wurden selbsterstellte Klassendiagramme die Basis der Recherche. Die beste Lösung hierfür war eigenständiges durcharbeiten und dokumentieren.

Dabei half, dass man händisch nahezu alle Pakete der Bibliotheken selbst herausgesucht hatte und auch einige logische Abhängigkeiten kannte.

Frustrierend waren vor allem die „rabbits-holes“, bei denen man bestimmten Aufrufen und Klassen-Sprüngen folgte, um dann nach mehreren dutzend Zeilen Code festzustellen, dass dies einen nicht ans gewünschte Ziel bringt.

Am Ende hat sich jedoch ein sehr gutes Verständnis für die Funktionsweise, die Abhängigkeiten und den gesamten Aufbau des Programms mit seinen diversen Bibliotheken entwickelt.

Die dann notwendigen Anpassungen selbst waren der leichteste Teil und erforderten nur wenige Zeilen Code.

Hier wurde es kurz noch einmal spannend, ausgelagerte Eigenschaften oder Funktionen richtig zu nutzen. (Denn bei 3 oder mehr *namespaces* war nicht unbedingt immer ersichtlich, wo genau diese Funktion jetzt herkommt.) Die Eigenschafts- und Parameternamen waren oftmals auch nicht sehr aufschlussreich.

Hier half wieder die Erfahrung, sich durch die meisten der Klassen gelesen zu haben, aber auch die vorher erwähnten „rabbit-holes“, durch die inzwischen einige Aufrufe oder Abhängigkeiten bekannt waren.

Es gab Funktionen und Klassen, welche zwar implementiert waren, aber nicht ordnungsgemäß funktionierten, wie beispielsweise die in Punkt 3.1.3 erwähnte Funktion *SingEncryptFolderCommand*. Außerdem gab es viele gleichartige Klassen, welche von einer abstrakten Klasse hätten abgeleitet werden können und den Programmcode übersichtlicher gemacht hätten. Auch das essenzielle, programmspezifische Funktionen in einer Bibliothek definiert/deklariert und ausgelagert wurden, welche besser in einer abgeleiteten Klasse im Programmcode hätten verfügbar sein sollen, erschwerte die Arbeit erheblich.

5. Fazit

Die Arbeit mit fremden Programmcode erwies sich als schwieriger als angenommen. Es hat viel Zeit in Anspruch genommen sich eine detaillierte Übersicht über die komplexe Programmstruktur und das Environment zu anzueignen. Es wurde festgestellt, dass sich andere Programmierer nicht immer an gängige Konventionen halten und viele Fehler wurden entdeckt. Reengineering-Tools wie Sourcetrail gaben einen guten anfänglichen Überblick, doch stieß man damit schnell an ihre Grenzen. Letztendlich war es nötig, eigenen Klassen und Sequenzdiagramme zu entwerfen, um den Zusammenhang zu verstehen. Ohne diese Klassendiagramme wäre eine konstruktive Arbeit nicht möglich gewesen.

Das Programmieren selbst erwies sich als der leichteste Teil, sobald die Abhängigkeiten verstanden waren.

Trotz all der Schwierigkeiten wurde die Aufgabenstellung erfüllt und alle Änderungen erfolgreich implementiert. Die Gruppenarbeit erwies sich als äußerst produktiv und hilfreich.

Somit konnten Änderungen an einem größeren Projekt wie Kleopatra erfolgreich umgesetzt werden.